Gesellschaft für Informatik (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into the fields of
- Seminar
- Proceedings
- Dissertations
- Thematics

current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English.

Information: http://www.gi-ev.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-238-3

The BTW 2009 in Münster is the 13th conference of its kind reflecting the broad range of academic research and industrial development work within the German database community.This year's conference focuses on a broad range of database topics covering query languages and their processing, integration, data warehousing, new applications, indexing, caching,, metadata, and data streams.This volume contains contributions from the refereed scientific program, the refereed industrial program, and the demo program.

J.-C. Freytag, T. Ruf, W. Lehner, G. Vossen (Hrsg.): BTW 2009

144

# GI-Edition

## Lecture Notes in Informatics

**Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)**

# Datenbanksysteme in Business, Technologie und Web (BTW)

**2.–6. März 2009**
**Münster**

# Proceedings

Johann-Christoph Freytag, Thomas Ruf,
Wolfgang Lehner, Gottfried Vossen (Hrsg.)

# Datenbanksysteme in Business, Technologie und Web (BTW)

## 13. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

**2.-6. März 2009**
**in Münster, Germany**

Gesellschaft für Informatik e.V. (GI)

**Lecture Notes in Informatics (LNI) - Proceedings**
Series of the Gesellschaft für Informatik (GI)

Volume P-144

ISBN 978-3-88579-238-3
ISSN 1617-5468

**Volume Editors**
Prof. Johann-Christoph Freytag, Ph.D.
Institut für Informatik
Humboldt-Universität zu Berlin, 10099 Berlin, Germany
E-Mail: freytag@dbis.informatik.hu-berlin.de
Prof. Dr. Thomas Ruf
GfK Retail and Technology GmbH, 90319 Nürnberg, Germany
E-Mail: thomas.ruf@gfk.com
Prof. Dr. Wolfgang Lehner
Department of Computer Science
Technische Universität Dresden, 01187 Dresden, Germany
E-Mail: wolfgang.lehner@tu-dresden.de
Prof. Dr. Gottfried Vossen
Institut für Wirtschaftsinformatik
Universität Münster, 48149 Münster, Germany
E-Mail: vossen@uni-muenster.de

**Series Editorial Board**
Heinrich C. Mayr, Universität Klagenfurt (Chairman)
Jörg Becker, Universität Münster
Hinrich Bonin, Leuphana-Universität Lüneburg
Dieter Fellner, Technische Universität Darmstadt
Ulrich Flegel, SAP
Johann-Christoph Freytag, Humboldt-Universität zu Berlin
Ulrich Furbach, Universität Koblenz
Michael Koch, Universität der Bundeswehr München
Axel Lehmann, Universität der Bundeswehr, München
Peter Liggesmeyer, Universität Kaiserslautern
Ernst W. Mayr, Technische Universität München
Heinrich Müller, Universität Dortmund
Sigrid Schubert, Universität Siegen
Martin Warnke, Leuphana-Universität Lüneburg

**Dissertations**
Dorothea Wagner, Universität Karlsruhe, Germany

**Seminars**
Reinhard Wilhelm, Universität des Saarlandes, Germany

**Thematics**
Andreas Oberweis, Universität Karlsruhe (TH)

© Gesellschaft für Informatik, Bonn 2009
**printed by** Köllen Druck+Verlag GmbH, Bonn

# Vorwort

Alle zwei Jahre findet die BTW-Konferenz der Gesellschaft für Informatik (GI) als die nationale Datenbankkonferenz an einem ausgezeichneten Ort in Deutschland statt. So ist die Westfälischen-Wilhelms Universität in Münster vom 4. bis 6. März 2009 Gastgeberin die 13. BTW-Konferenz. Münster mit seiner mehr als 1200-jährigen Geschichte im Zentrum Westfalens gilt vielen als Ort der Tradition und der Kultur; hier wurde 1648 der Westfälische Frieden geschlossen, der das Ende des 30-jährigen Krieges in Europa markierte. Heute sind die Stadt, traditionell Sitz von Bildungs- und Verwaltungseinrichtungen sowie Finanzdienstleistern, und das umliegende Münsterland eine Region mit wachsender Wirtschafts- und Innovationskraft, die sie zum großen Teil der hiesigen Universität und ihren angegliederten Instituten verdankt.

Die BTW-Tagung ist seit über 20 Jahren das zentrale Forum der deutschsprachigen Datenbankgemeinde, die zahlen- und publikationsmäßig zu einer der stärksten weltweit gehört. Auf dieser Tagung treffen sich alle zwei Jahre nicht nur Wissenschaftler, sondern auch Praktiker und Anwender. Die Tradition der BTW hat ihren Anfang in der ersten Tagung 1985 in Karlsruhe zu einer Zeit, in der sich Datenbanksysteme von den klassischen betrieblichen Einsatzfeldern zu Anwendungen in Büro, Technik und Wissenschaft entwickelten, daher der ursprüngliche Name BTW.

Heute ist Datenbanktechnologie der wichtigste Stützpfeiler der IT-Branche generell; sie ist unverzichtbar für organisationsübergreifende Kooperationen und elektronische Prozesse, als Infrastruktur in der Telekommunikation und anderen eingebetteten Technologien, als skalierbares Rückgrat digitaler Bibliotheken, vieler Data-Mining-Werkzeuge sowie für viele Arten von Web-Anwendungen und der Realisierung service-orientierter Architekturen (SOAs).

Im Zeitalter der Informationsexplosion, Virtualisierung und Web-Orientierung kommen auf die Datenbank- und Informationssystemtechnologie kontinuierlich neue Herausforderungen zu. Diese spiegeln sich in Themen wie Informationsintegration aus heterogenen, verteilten Datenquellen mit verschiedenen Medien und variierendem Strukturierungsgrad, internetweites, kollaboratives Informationsmanagement, Grid/Cloud-Computing und E-Science-Kollaborationen oder der Gestaltung der Vision eines "Semantic Web" wider. Hinzu kommen Fragen aus den Bereichen der Datenqualität, der Datenstromverarbeitung sowie Einbindung von Ontologien und anderen Konzepten des Web 2.0, um den ständig komplexer werdenden Anforderungen aus verschiedenen Anwendungsgebieten gerecht zu werden. Im Bereich der Geschäftsprozessmodellierung und -realisierung müssen prozess- und dienstorientierte Architekturen weiterentwickelt und ihre Anforderungen auf Datenbank- und Informationssysteme hin formuliert werden.

In seiner Struktur hält sich die diesjährige BTW-Tagung an ihre Tradition. Sie umfasst auch diesmal ein wissenschaftliches Programm, ein Industrieprogramm und ein Demonstrationsprogramm. Am Rande der Tagung finden zusätzlich verschiedene Workshops, Tutorien im Rahmen der Datenbank-Tutorientage sowie ein Studierendenpro-

gramm statt. Für das wissenschaftliche Programm wurden Beiträge über die Weiterentwicklung der Datenbanktechnologie, ihrer Grundlagen und ihrer Wechselwirkungen mit benachbarten Gebieten sowie ihrer Anwendungen ausgewählt. Beiträge über den kommerziellen Einsatz von Datenbanktechnologie, Erfahrungen und aktuelle Industrietrends wurden für das Industrieprogramm zusammengestellt. Wie bei den bisherigen BTW-Tagungen waren sowohl Langbeiträge (Original- oder Übersichtsarbeiten) als auch Kurzbeiträge (über neuere Projekte oder erste Zwischenergebnisse laufender Forschungsarbeiten) erwünscht. Im wissenschaftlichen Bereich werden die Themen *Anfragesprachen und Anfragebearbeitung*, *Datenintegration und Metadaten*, *Data Warehousing und Caching*, *Datenströme* sowie *Neue Anwendungen* adressiert. Das Industrieprogramm behandelt die Bereiche *Neue Technologien für Datenbanken*, *Optimierungstechniken für Datenbankanfragen* sowie *Business Intelligence*.

Zusätzlich umfasst die BTW drei eingeladene Vorträge:

- Ricardo Baeza-Yates (Yahoo!Research Barcelona) entwirft in seinem Vortrag „Towards a Distributed Search Engine“ seine Vision zur Weiterentwicklung von Suchmaschinen.
- Juliana Freire (University of Utah, USA) untersucht in ihrer Keynote “Provenance Management: Challenges and Opportunities” das immer wichtiger werdende Feld der Protokollierung von Daten-Herkunft.
- Sergey Melnik (Google Inc.) stellt als BTW-Dissertationspreisgewinner 2003 in seinem Vortrag “The Frontiers of Data Programmability” seine neuesten Forschungsarbeiten bei Google vor.

Nach guter Tradition werden im Rahmen der BTW auch drei Preise für hervorragende Dissertationen im Datenbankbereich vergeben:

- Ira Assent (RWTH Aachen): Efficient Adaptive Retrieval and Mining in Large Multimedia Databases (Betreuer: Thomas Seidl);
- Sebastian Michel (Universität Saarbrücken & MPI Saarbrücken): Top-k Aggregation Queries in Large-Scale Distributed System (Betreuer: Gerhard Weikum);
- Jürgen Krämer (Universität Marburg): Continuous Queries over Data Streams - Semantics and Implementation (Betreuer: Bernhard Seeger)

Die Organisation einer großen Tagung wie der BTW ist nicht ohne zahlreiche Partner und Helfer möglich. Sie sind auf den folgenden Seiten aufgeführt, und ihnen gilt unser herzlicher Dank ebenso wie den Sponsoren der Tagung und der GI-Geschäftsstelle.

Berlin, Erlangen, Dresden, Münster, im Januar 2009

Johann-Christoph Freytag, Vorsitzender des Programmkomitees
Thomas Ruf, Vorsitzender des Industriekomitees
Wolfgang Lehner, Vorsitzender des Demokomitees
Gottfried Vossen, Tagungsleitung

## Tagungsleitung:

Gottfried Vossen, Univ. Münster

## Programmkomiteeleitung:

Johann-Christoph Freytag, Humboldt-Univ. zu Berlin

## Programmkomitee:

Hans-Jürgen Appelrath, Univ. Oldenburg
Christian Böhm, LMU München
Stefan Conrad, Univ. Düsseldorf
Stefan Dessloch, Univ. Kaiserslautern
Jens Dittrich, ETH Zürich
Silke Eckstein, Univ. Braunschweig
Burkhard Freitag, Univ. Passau
Torsten Grust, TU München
Theo Härder, Univ. Kaiserslautern
Andreas Henrich, Univ. Bamberg
Melanie Herschel, IBM Almaden
Carl Christian Kanne, Univ. Mannheim
Meike Klettke, Univ. Rostock
Birgitta König-Ries, Univ. Jena
Klaus Küspert, Univ. Jena
Jens Lechtenbörger, Univ. Münster
Wolfgang Lehner, TU Dresden
Ulf Leser, HU Berlin
Volker Linnemann, Univ. Lübeck
Thomas Mandl, Univ. Hildesheim
Stefan Manegold, CWI Amsterdam
Volker Markl, TU Berlin
Wolfgang May, Univ. Göttingen
Klaus Meyer-Wegener, Univ. Erlangen
Heiko Müller, Edinburgh
Felix Naumann, HPI Potsdam
Thomas Neumann ,MPI Saarbrücken
Daniela Nicklas, Univ. Oldenburg
Peter Peinl, FH Fulda
Erhard Rahm, Univ. Leipzig
Manfred Reichert, Univ. Ulm
Norbert Ritter, Univ. Hamburg
Gunter Saake, Univ. Madgeburg
Kai-Uwe Sattler, TU Ilmenau

Eike Schallehn, Univ. Magdeburg
Ralf Schenkel, MPI Saarbrücken
R. Ingo Schmitt, TU Cottbus
Harald Schöning, Software AG
Holger Schwarz, Univ. Stuttgart
Bernhard Seeger, Univ. Marburg
Thomas Seidl, RWTH Aachen
Günther Specht, Univ. Innsbruck
Myra Spiliopoulou, Univ. Magdeburg
Knuth Stolze, IBM Böblingen
Uta Störl, Hochschule Darmstadt
Jens Teubner, ETH Zürich
Can Türker, ETH Zürich
Klaus Turowski, Univ. Augsburg
Agnes Voisard, Fraunhofer Berlin
Mechtild Wallrath, BA Karlsruhe
Mathias Weske, HPI Potsdam
Christa Womser-Hacker, Univ. Hildesheim

## Industriekomitee:

Bärbel Bohr, UBS AG
Götz Graefe, Hewlett Packard
Christian König, Microsoft Research
Albert Maier, IBM Research
Thomas Ruf, GfK Retail and Technology GmbH, (Vorsitz)
Carsten Sapia, BMW AG
Stefan Sigg, SAP AG

## Studierendenprogramm:

Hagen Höpfner, International University, Bruchsal

## Tutorienprogramm:

Wolfgang Lehner, TU Dresden

## Organisationskomitee:

Gottfried Vossen (Leitung)
Ralf Farke
Till Haselmann
Jens Lechtenbörger
Joachim Schwieren
Jens Sieberg
Gunnar Thies
Barbara Wicher

## Gutachter für Dissertationspreise:

Stefan Conrad, Univ. Düsseldorf
Johann-Christoph Freytag, HU Berlin, (Vorsitz)
Theo Härder, Univ. Kaiserslautern
Alfons Kemper, TU München
Donald Kossmann, ETH Zürich
Georg Lausen, Univ. Freiburg
Udo Lipeck, Univ. Hannover
Bernhard Seeger, Univ. Marburg
Gottfried Vossen, Univ. Münster
Gerhard Weikum, MPI Saarbrücken

## Externe Gutachter:

Sadet Alcic, Univ. Düsseldorf
Christian Beecks, RWTH Aachen
Stefan Bensch, Univ. Augsburg
Lukas Blunschi, ETH Zurich
Dietrich Boles, Univ. Oldenburg
Andr Bolles, Univ. Oldenburg
Stefan Brüggemann, OFFIS, Oldenburg
Stephan Ewen, TU Berlin
Frank Fiedler, LMU München
Marco Grawunder, Univ. Oldenburg
Fabian Grüning, Univ. Oldenburg
Stephan Günnemann, RWTH Aachen
Michael Hartung, Univ. Leipzig
Wilko Heuten, OFFIS, Oldenburg
Marc Holze, Univ. Hamburg
Fabian Hueske, TU Berlin
Jonas Jacobi, Univ. Oldenburg
Thomas Kabisch, TU Berlin

Andreas Kaiser, Univ. Passau
Toralf Kirsten, Univ. Leipzig
Kathleen Krebs, Univ. Hamburg
Hardy Kremer, RWTH Aachen
Annahita Oswald, LMU München
Fabian Panse, Univ. Hamburg
Philip Prange, Univ. Marburg
Michael von Riegen, Uni Hamburg
Karsten Schmidt, Univ. Kaiserslautern
Joachim Selke, TU Braunschweig
Sonny Vaupel, Univ. Marburg
Gottfried Vossen, Univ. Münster
Bianca Wackersreuther, LMU München
Andreas Weiner, Univ. Kaiserslautern

# Inhaltsverzeichnis

**Eingeladene Beiträge**

Ricardo Baeza-Yates (Yahoo!Research Barcelona): *Towards a Distributed Search Engine* 2

Juliana Freire (University of Utah, USA): *Provenance Management: Challenges and Opportunities* 4

Sergey Melnik (Google Inc.): *The Frontiers of Data Programmability* 5

**Wissenschaftliches Programm**

*Anfrageverarbeitung*

Th. Neumann (MPI Saarbrücken), G. Moerkotte (Uni Mannheim): *A Framework for Reasoning about Share Equivalence and Its Integration into a Plan Generator* 7

G. Graefe (HP Labs), R. Stonecipher (Microsoft): Efficient Verification of B-tree Integrity 27

M. Heimel (IBM Germany), V. Markl (TU Berlin), K. Murthy (IBM San Jose): *A Bayesian Approach to Estimating the Selectivity of Conjunctive Predicates* 47

Ch. Böhm, R. Noll (LMU München), C. Plant, A. Zherdin (TU München): *Index-supported Similarity Join on Graphics Processors* 57

*Integration*

M. Böhm (HTW Dresden), D. Habich, W. Lehner (TU Dresden), U. Wloka (HTW Dresden): *Systemübergreifende Kostennormalisierung für Integrationsprozesse* 67

S. Mir (EML Heidelberg), St. Staab (Uni Koblenz), I. Rojas (EML Heidelberg): *Web-Prospector – An Automatic, Site-Wide Wrapper Induction Approach for Scientific Deep-Web Databases* 87

Th. Mandl (Uni Hildesheim): *Easy Tasks Dominate Information Retrieval Evaluation Results* 107

*Anfragesprachen*

M. Rosenmüller, Ch. Kästner, N. Siegmund, S. Sunkle (Uni Magdeburg), S. Apel (Uni Passau), Th. Leich (METOP GmbH), G. Saake (Uni Magdeburg): *SQL á la Carte - Toward Tailor-made Data Management* 117

I. Schmitt, D. Zellhöfer (TU Cottbus): *Lernen nutzerspezifischer Gewichte innerhalb einer logikbasierten Anfragesprache* 137

K. Stolze (IBM Germany R&D), V. Raman, R. Sidle (IBM ARC), O. Draese (IBM Germany R&D): *Bringing BLINK Closer to the Full Power of SQL* 157

*Speicherung und Indizierung*

Th. Härder, K. Schmidt, Y. Ou, S. Bächle (Uni Kaiserslautern): *Towards Flash Disk Use in Databases – Keeping Performance While Saving Energy?* 167

I. Assent (Aalborg Univ., Dänemark), St. Günnemann, H. Kremer, Th. Seidl (RWTH Aachen): *High-Dimensional Indexing for Multimedia Features* 187

Ch. Beecks, M. Wichterich, Th. Seidl (RWTH Aachen): *Metrische Anpassung der Earth Mover's Distanz zur Ähnlichkeitssuche in Multimedia-Datenbanken* 207

*Neue Anwendungen*

S. Schulze, M. Pukall, G. Saake, T. Hoppe, J. Dittmann (Uni Magdeburg): *On the Need of Data Management in Automotive Systems* 217

F. Irmert, Ch. Neumann, M. Daum, N. Pollner, K. Meyer-Wegener (Uni Erlangen): *Technische Grundlagen für eine laufzeitadaptierbare Transaktionsverwaltung* 227

D. Aumüller (Uni Leipzig): *Towards web supported identification of top affiliations from scholarly papers* 237

S. Tönnies, B. Köhncke (L3S Hannover), O. Köpler (Uni Hannover), W.-T. Balke (L3S Hannover): *Building Chemical Information Systems – the ViFaChem II Project* 247

*Metadaten*

J. Goeres, Th. Jörg, B. Stumm, St. Dessloch (Uni Kaiserslautern): *GEM: A Generic Visualization and Editing Facility for Heterogeneous Metadata* 257

A. Thor, M. Hartung, A. Gross, T. Kirsten, E. Rahm (Uni Leipzig): *An Evolution-based Approach for Assessing Ontology Mappings - A Case Study in the Life Sciences* 277

Ch. Böhm (LMU München), L. Läer, C. Plant, A. Zherdin (TU München): *Model-based Classification of Data with Time Series-valued Attributes* 287

N. Siegmund, Ch. Kästner, M. Rosenmüller (Uni Magdeburg), F. Heidenreich (TU Dresden), S. Apel (Uni Passau), G. Saake (Uni Magdeburg): *Bridging the Gap between Variability in Client Application and Database Schema* 297

*Data Warehousing und Caching*

M. Thiele, A. Bader, W. Lehner (TU Dresden): *Multi-Objective Scheduling for Real-Time Data Warehouses* 307

Th. Jörg, St. Dessloch (Uni Kaiserslautern): *Formalizing ETL Jobs for Incremental Loading of Data Warehouses* 327

J. Klein, S. Braun, G. Machado (Uni Kaiserslautern): *Selektives Laden und Entladen von Prädikatsextensionen beim Constraint-basierten Datenbank-Caching* 347

*Datenströme*

C. Franke (UC Davis), M. Karnstedt, D. Klan (TU Ilmenau), M. Gertz (Uni Heidelberg), K.-U. Sattler (TU Ilmenau), W. Kattanek (IMMS GmbH): *In-Network Detection of Anomaly Regions in Sensor Networks with Obstacles* 367

J. Jacobi, A. Bolles, M. Grawunder, D. Nicklas, H.-J. Appelrath (Uni Oldenburg): *Priorisierte Verarbeitung von Datenstromelementen* 387

St. Preißler, H. Voigt, D. Habich, W. Lehner (TU Dresden): *Stream-Based Web Service Invocation* 407

**Dissertationspreise**

S. Michel (Uni Saarbrücken & MPI Saarbrücken): *Top ‐ k Aggegation Queries in Large ‐ Scale Distributed Systems* 418

I. Assent (RWTH Aachen): *Efficient Adaptive Retrieval and Mining in Large Multimedia Databases* 428

J. Krämer (Uni Marburg): *Continuous Queries over Data Streams - Semantics and lmplementation* 438

**Industrieprogramm**

*Business Intelligence*

M. Oberhofer, E. Nijkamp (IBM Böblingen): *Embedded Analytics in Front Office Applications* 449

U. Christ (SAP Walldorf): *An Architecture for Integrated Operational Business Intelligence* 460

A. Lang, M. Ortiz, St. Abraham (IBM Böblingen): *Enhancing Business Intelligence with Unstructured Data* 469

*Optimierungstechniken*

Ch. Lemke (SAP Walldorf), K.-U. Sattler (TU Ilmenau), F. Färber (SAP Walldorf): *Kompressionstechniken für spaltenorientierte BI-Accelerator-Lösungen* 486

M. Fiedler, J. Albrecht, Th. Ruf, J. Görlich, M. Lemm (GfK Nürnberg): *Pre-Caching hochdimensionaler Aggregate mit relationaler Technologie* 498

H. Loeser (IBM Böblingen), M. Nicola, J. Fitzgerald (IBM San Jose): *Index Challenges in Native XML Database Systems* 508

*Neue Technologien*

St. Buchwald, Th. Bauer (Daimler AG Ulm), R. Pryss (Uni Ulm): *IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung* 526

St. Aulbach (TU München), D. Jacobs, J. Primsch (SAP Walldorf), A. Kemper (TU München): *Anforderungen an Datenbanksysteme für Multi-Tenancy- und Software-as-a-Service-Applikationen* 544

U. Hohenstein, M. Jäger (Siemens München): *Die Migration von Hibernate nach OpenJPA: Ein Erfahrungsbericht* 556

**Demo-Programm**

D. Aumüller (Uni Leipzig): *Retrieving Metadata for Your Local Scholarly Papers* 577

K. Benecke, M. Schnabel (Uni Magdeburg): *OttoQL* 580

A. Behrend, Ch. Dorau, R. Manthey (Uni Bonn): *TinTO: A Tool for View-Based Analysis of Stock Market Data Streams* 584

A. Brodt, N. Cipriani (Uni Stuttgart): *NexusWeb – eine kontextbasierte Webanwendung im World Wide Space* 588

D. Fesenmeyer, T. Rafreider, J. Wäsch (HTWG Konstanz): *Ein Tool-Set zur Datenbank-Analyse und –Normalisierung* 592

B. Jäksch, R. Lembke, B. Stortz, St. Haas, A. Gerstmair, F. Färber (SAP Walldorf): *Guided Navigation basierend auf SAP Netweaver BIA* 596

F. Gropengießer, K. Hose, K.-U. Sattler (TU Ilmenau): *Ein kooperativer XML-Editor für Workgroups* 600

H. Höpfner, J. Schad, S. Wendland, E. Mansour (IU Bruchsal): *MyMIDP and MyMIDP-Client: Direct Access to MySQL Databases from Cell Phones* 604

St. Preißler, H. Voigt, D. Habich, W. Lehner (TU Dresden): *Streaming Web Services and Standing Processes* 608

St. Scherzinger, H. Karn, T. Steinbach (IBM Böblingen): *End-to-End Performance Monitoring of Databases in Distributed Environments* 612

A. M. Weiner, Ch. Mathis, Th. Härder, C. R. F. Hoppen (Uni Kaiserslautern): *Now it's Obvious to The Eye – Visually Explaining XQuery Evaluation in a Native XML Database Management System* 616

D. Wiese, G. Rabinovitch, M. Reichert, St. Arenswald (Uni Jena / IBM Böblingen): *ATE: Workload-Oriented DB2 Tuning in Action* 620

E. Nijkamp, M. Oberhofer, A. Maier (IBM Böblingen): *Value Demonstration of Embedded Analytics for Front Office Applications* 624

Martin Oberhofer, Albert Maier (IBM Böblingen): *Support 2.0: An Optimized Product Support System Exploiting Master Data, Data Warehousing and Web 2.0 Technologies* 628

**DBTT-Tutorial**

Dean Jacobs (SAP AG): *Software as a Service: Do It Yourself or Use the Cloud* 633

# Eingeladene Beiträge

# Towards a Distributed Web Search Engine

Ricardo Baeza-Yates
Yahoo! Research
Barcelona, Spain
rbaeza@acm.org

**Abstract:** We present recent and on-going research towards the design of a distributed Web search engine. The main goal is to be able to mimic a centralized search engine with similar quality of results and performance, but using less computational resources. The main problem is the network latency when different servers have to process the queries. Our preliminary findings mix several techniques, such as caching, locality prediction and distributed query processing, that try to maximize the fraction of queries that can be solved locally.

## 1 Summary

Designing a distributed Web search engine is a challenging problem [BYCJ+07], because there are many external factors that affect the different tasks of a search engine: crawling, indexing and query processing. On the other hand, local crawling profits with the proximity to Web servers, potentially increasing the Web coverage and freshness [CPJT08]. Local content can be indexed locally, communicating later local statistics that can be helpful at the global level. So the natural distributed index is a document partitioned index [BYRN99].

Query processing is very efficient for queries that can be answered locally, but too slow if we need to request answers from remote servers. One way to improve the performance is to increase the fraction of queries that look like local queries. This can be achieved by caching results [BYGJ+08a], caching partial indexes [SJPBY08] and caching documents [BYGJ+08b], with different degree of effectiveness. A complementary technique is to predict if a query will need remote results and request in parallel local and remote results, instead of doing a sequential process [BYMH08]. Putting all these ideas together we can have a distributed search engine that has similar performance to a centralized search engine but that needs less computational resources and maintenance cost than the equivalent centralized Web search engine [BYGJ+08b].

Future research must study how all these techniques can be integrated and optimized, as we have learned that the optimal solution changes depending on the interaction of the different subsystems. For example, caching the index will have a different behavior if we are caching results or not.

## References


[BYCJ+07] Ricardo Baeza-Yates, Carlos Castillo, Flavio Junqueira, Vassilis Plachouras and Fabrizio Silvestri. Challenges on Distributed Web Retrieval. In *ICDE*, 6–20, 2007.

[BYGJ+08a] Ricardo Baeza-Yates, Aristides Gionis, Flavio P. Junqueira, Vanessa Murdock, Vassilis Plachouras and Fabrizio Silvestri. Design trade-offs for search engine caching. *ACM Trans. Web*, 2(4):1–28, 2008.

[BYGJ+08b] Ricardo Baeza-Yates, Aristides Gionis, Flavio P. Junqueira, Vassilis Plachouras and Luca Telloli. On the feasibility of multi-site Web search engines. Submitted, 2008.

[BYMH08] Ricardo Baeza-Yates, Vanessa Murdock and Claudia Hauff. Speeding-Up Two-Tier Web Search Systems. Submitted, 2008.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.

[CPJT08] B. Barla Cambazoglu, Vassilis Plachouras, Flavio Junqueira and Luca Telloli. On the feasibility of geographically distributed web crawling. In *InfoScale '08: Proceedings of the 3rd international conference on Scalable information systems*, 1–10, ICST, Brussels, Belgium, Belgium, 2008.

[SJPBY08] Gleb Skobeltsyn, Flavio Junqueira, Vassilis Plachouras and Ricardo Baeza-Yates. ResIn: a combination of results caching and index pruning for high-performance web search engines. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 131–138, New York, NY, USA, 2008. ACM.

# Provenance Management: Challenges and Opportunities

Juliana Freire

School of Computing
University of Utah
Salt Lake City, Utah, USA

juliana@cs.utah.edu

**Abstract:** Computing has been an enormous accelerator to science and industry alike and it has led to an information explosion in many different fields. The unprecedented volume of data acquired from sensors, derived by simulations and data analysis processes, accumulated in warehouses, and often shared on the Web, has given rise to a new field of research: provenance management. Provenance (also referred to as audit trail, lineage, and pedigree) captures information about the steps used to generate a given data product. Such information provides important documentation that is key to preserve data, to determine the data's quality and authorship, to understand, reproduce, as well as validate results. Provenance solutions are needed in many different domains and applications, from environmental science and physics simulations, to business processes and data integration in warehouses. In this talk, we survey recent research results and outline challenges involved in building provenance management systems. We also discuss emerging applications that are enabled by provenance and outline open problems and new directions for database-related research.

# The Frontiers of Data Programmability

Sergey Melnik

Google, Inc.
Seattle-Kirkland R&D Center, USA

melnik@google.com

**Abstract:** Simplifying data programming is a core mission of data management research. The issue at stake is to help engineers build efficient and robust data-centric applications. The frontiers of data programmability extend from longstanding problems, such as the impedance mismatch between programming languages and databases, to more recent challenges of web programmability and large-scale data-intensive computing. In this talk I will review some fundamental technical issues faced by today's application developers. I will present recent data programmability solutions for the .NET platform that include Language-Integrated Querying, Entity Data Model, and advanced techniques for mapping between objects, relations, and XML.

# Wissenschaftliches Programm

# A Framework for Reasoning about Share Equivalence and Its Integration into a Plan Generator

Thomas Neumann #1, Guido Moerkotte *2

#*Max-Planck Institute for Informatics, Saarbrücken, Germany*
1neumann@mpi-inf.mpg.de

**University of Mannheim, Mannheim, Germany*
2moerkotte@informatik.uni-mannheim.de

**Abstract:** Very recently, Cao et al. presented the MAPLE approach, which accelerates queries with multiple instances of the same relation by sharing their scan operator. The principal idea is to derive, in a first phase, a non-shared tree-shaped plan via a traditional plan generator. In a second phase, common instances of a scan are detected and shared by turning the operator tree into an operator DAG (directed acyclic graph).

The limits of their approach are obvious. (1) Sharing more than scans is often possible and can lead to considerable performance benefits. (2) As sharing influences plan costs, a separation of the optimization into two phases comprises the danger of missing the optimal plan, since the first optimization phase does not know about sharing.

We remedy both points by introducing a general framework for reasoning about sharing: plans can be shared whenever they are *share equivalent* and not only if they are scans of the same relation. Second, we sketch how this framework can be integrated into a plan generator, which then constructs optimal DAG-structured plans.

## 1 Introduction

Standard query evaluation relies on tree-structured algebraic expressions which are generated by the plan generator and then evaluated by the query execution engine [Lor74]. Conceptually, the algebra consists of operators working on sets or bags. On the implementation side, they take one or more tuple (object) streams as input and produce a single output stream. The tree-structure thereby guarantees that every operator – except for the root – has exactly one consumer of its output. This flexible concept allows a nearly arbitrary combination of operators and highly efficient implementations.

However, this model has several limitations. Consider, e.g., the following SQL query:

```
select ckey
from   customer, order
where  ckey=ocustomer
group by ckey
having sum(price) = (select max(total)
   from (select ckey, sum(price) as total
         from   customer, order
         where  ckey=ocustomer
         group  by ckey))
```

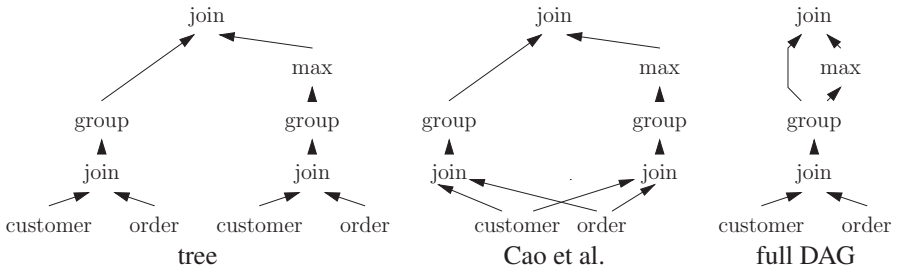



Figure 1: Example plans

This query leads to a plan like the one at the left of Fig. 1. We observe that (1) both relations are accessed twice, (2) the join and (3) the grouping are calculated twice. To (partially) remedy this situation, Cao et al. proposed to share scans of the same relation [CDCT08]. The plan resulting from their approach is shown in the middle of Fig. 1. Still, not all sharing possibilities are exploited. Obviously, only the plan at the right exploits sharing to its full potential.

Another disadvantage of the approach by Cao et al. is that optimization is separated into two phases. In a first phase, a traditional plan generator is used to generate tree-structured plans like the one on the left of Fig. 1. In a second step, this plan is transformed into the one at the middle of Fig. 1. This approach is very nice in the sense that it does not necessitate any modification to existing plan generators: just an additional phase needs to be implemented. However, as always when more than a single optimization phase is used, there is the danger of coming up with a suboptimal plan. In our case, this is due to the fact that adding sharing substantially alters the costs of a plan. As the plan generator is not aware of this cost change, it can come up with (from its perspective) best plan, which exhibits (after sharing) higher costs than the optimal plan.

In this paper, we remedy both disadvantages of the approach by Cao et al. First, we present a general framework that allows us to reason about share equivalences. This will allow us to exploit as much sharing as possible, if this leads to the best plan. Second, we sketch a plan generator that needs a single optimization phase to generate plans with sharing. Using a single optimization phase avoids the generation of suboptimal plans. The downside is that the plan generator has to be adopted to include our framework for reasoning about share equivalence. However, we are strongly convinced that this effort is worth it.

The rest of the paper is organized a follows. Section 2 discusses related work. Section 3 precisely defines the problem. Section 4 describes the theoretical foundation for reasoning about share equivalence. Section 5 sketches the plan generator. The detailed pseudocode and its discussion is given in [NM08]. Section 6 contains the evaluation. Section 7 concludes the paper.

## 2 Related Work

Let us start the discussion on related work with a general categorization. Papers discussing the generation of DAG-structured query execution plans fall into two broad categories. In the first category, a single optimal tree-structured plan is generated, which is then turned into a DAG by exploiting sharing. This approach is in danger of missing the optimal plan since the tree-structured plan is generated with costs which neglect sharing opportunities.

We call this *post plan generation share detection* (PPGSD). This approach is the most prevailing one in multi-query optimization, e.g. [Sel88]. In the second category, common subexpressions are detected in a first phase before the actual plan generation takes place. The shared subplans are generated independently and then replaced by an artificial single operator. This modified plan is then given to the plan generator. If several sharing alternatives exist, several calls to the plan generator will be made. Although this is a very expensive endeavor due to the (in the worst case exponentially many) calls to the plan generator. Since the partial plans below and above the materialization (temp) operator are generated separately, there is a slight chance that the optimal plan is missed. We term this *loose coupling* between the share detection component and the plan generator. In stark contrast, we present a tightly integrated approach that allows to detect sharing opportunities incrementally during plan generation.

A Starburst paper mentions that DAG-structured query graphs would be nice, but too complex [HFLP89]. A later paper about the DB2 query optimizer [GLSW93] explains that DAG-structured query plans are created when considering views, but this solution materializes results in a temporary relation. Besides, DB2 optimizes the parts above and below the temp operator independently, which can lead to suboptimal plans. Similar techniques are mentioned in [Cha98, GLJ01].

The Volcano query optimizer [Gra90] can generate DAGs by partitioning data and executing an operator in parallel on the different data sets, merging the result afterwards. Similar techniques are described in [Gra93], where algorithms like select, sort, and join are executed in parallel. However, these are very limited forms of DAGs, as they always use data partitioning (i.e., in fact, one tuple is always read by one operator) and sharing is only done within one logical operator.

Another approach using loose coupling is described in [Roy98]. A later publication by the same author [RSSB00] applies loose coupling to multi-query optimization. Another interesting approach is [DSRS01]. It also considers cost-based DAG construction for multi-query optimization. However, its focus is quite different. It concentrates on scheduling problems and uses greedy heuristics instead of constructing the optimal plan. Another loose coupling approach is described in [ZLFL07]. They run the optimizer repeatedly and use view matching mechanisms to construct DAGs by using solutions from the previous runs. Finally, there exist a number of papers that consider special cases of DAGs, e.g. [DSRS01, BBD$^+$04]. While they propose using DAGs, they either produce heuristical solutions or do not support DAGs in the generality of the approach presented here.

## 3 Problem Definition

Before going into detail, we provide a brief formal overview of the optimization problem we are going to solve in this paper. This section is intended as an illustration to understand the problem and the algorithm. Therefore, we ignore some details like the problem of operator selection here (i.e. the set of operators does not change during query optimization).

We first consider the classical tree optimization problem and then extend it to DAG optimization. Then, we distinguish this from similar DAG-related problems in the literature. Finally, we discuss further DAG-related problems that are not covered in this paper.

### 3.1 Optimizing Trees

It is the query optimizer's task to find the cheapest query execution plan that is equivalent to the given query. Usually this is done by algebraic optimization, which means the query optimizer tries to find the cheapest algebraic expression (e.g. in relational algebra) that is equivalent to the original query. For simplicity we ignore the distinction between physical and logical algebra in this section. Further, we assume that the query is already given as an algebric expression. As a consequence, we can safely assume that the query optimizer transforms one algebraic expression into another.

Nearly all optimizers use a tree-structured algebra, i.e. the algebraic expression can be written as a tree of operators. The operators themselves form the nodes of the tree, the edges represent the dataflow between the operators. In order to make the distinction between trees and DAGs apparent, we give their definitions. A *tree* is a directed, cycle-free graph $G = (V, E), |E| = |V| - 1$ with a distinguished root node $v_0 \in V$ such that all $v \in V \setminus \{v_0\}$ are reachable from $v_0$.

Now, given a query as a tree $G = (V, E)$ and a cost function $c$, the query optimizer tries to find a new tree $G' = (V, E')$ such that $G \equiv G'$ (concerning the produced output) and $c(G')$ is minimal (to distinguish the tree case from the DAG case we will call this equivalence $\equiv_T$). This can be done in different ways, either transformatively by transforming $G$ into $G'$ using known equivalences [Gra94, GM93, Gra95], or constructively by building $G'$ incrementally [Loh88, SAC$^+$79]. The optimal solution is usually found by using dynamic programming or memoization. If the search space is too large then heuristics are used to find good solutions.

An interesting special case is the join ordering problem where $V$ consists only of joins and relations. Here, the following statement holds: any tree $G'$ that satisfies the syntax constraints (binary tree, relations are leafs) is equivalent to $G$. This makes constructive optimization quite simple. However, this statement does no longer hold for DAGs (see Sec. 4).

### 3.2 Optimizing DAGs

DAGs are directed acyclic graphs, similar to trees with overlapping (shared) subtrees. Again, the operators form the nodes, and the edges represent the dataflow. In contrast to trees, multiple operators can depend on the same input operator. We are only interested in DAGs that can be used as execution plans, which leads to the following definition. A DAG is a directed, cycle-free graph $G = (V, E)$ with a denoted root node $v_0 \in V$ such that all $v \in V \setminus \{v_0\}$ are reachable from $v_0$. Note that this is the definition of trees without the condition $|E| = |V| - 1$. Hence, all trees are DAGs.

As stated above, nearly all optimizers use a tree algebra, with expressions that are equivalent to an operator tree. DAGs are no longer equivalent to such expressions. Therefore, the semantics of a DAG has to be defined. To make full use of DAGs, a DAG algebra would be required (and some techniques require such a semantics, e.g. [SPMK95]). However, the normal tree algebra can be lifted to DAGs quite easily: a DAG can be transformed into an equivalent tree by copying all vertices with multiple parents once for each parent. Of course this transformation is not really executed: it only defines the semantics. This trick allows us to lift tree operators to DAG operators, but it does not allow the lifting of

tree-based equivalences (see Sec. 4).

We define the problem of optimizing DAGs as follows. Given the query as a DAG $G = (V, E)$ and a cost function $c$, the query optimizer has to find any DAG $G' = (V' \subseteq V, E')$ such that $G \equiv G'$ and $c(G')$ is minimal. Thereby, we defined two DAG-structured expressions to be equivalent ($\equiv_D$) if and only if they produce the same output. Note that there are two differences between tree optimization and DAG optimization: First, the result is a DAG (obviously), and second, the result DAG possibly contains fewer operators than the input DAG.

Both differences are important and both are a significant step from trees! The significance of the latter is obvious as it means that the optimizer can choose to eliminate operators by reusing other operators. This requires a kind of reasoning that current query optimizers are not prepared for. Note that this decision is made during optimization time and not beforehand, as several possibilities for operator reuse might exist. Thus, a cost-based decision is required. But also the DAG construction itself is more than just reusing operators: a real DAG algebra (e.g. [SPMK95]) is vastly more expressive and cannot e.g. be simulated by deciding operator reuse beforehand and optimizing trees.

The algorithm described in this work solves the DAG construction problem in its full generality.

By this we mean that (1) it takes an arbitrary query DAG as input (2) constructs the optimal equivalent DAG, and (3) thereby applies equivalences, i.e. a rule-based description of the algebra. This discriminates it from the problems described below, which consider different kinds of DAG generation.

### 3.3 Problems Not Treated in Depth

In this work, we concentrate on the algebraic optimization of DAG-structured query graphs. However, using DAGs instead of trees produces some new problems in addition to the optimization itself.

One problem area is the execution of DAG-structured query plans. While a tree-structured plan can be executed directly using the iterator model, this is no longer possible for DAGs. One possibility is to materialize the intermediate results used by multiple operators, but this induces additional costs that reduce the benefit of DAGs. Ideally, the reuse of intermediate results should not cause any additional costs, and, in fact, this can be achieved in most cases. As the execution problem is common for all techniques that create DAGs as well as for multi-query optimization, many techniques have been proposed. A nice overview of different techniques can be found in [HSA05]. In addition to this generic approach, there are many special cases like e.g. application in parallel systems [Gra90] and sharing of scans only [CDCT08]. The more general usage of DAGs is considered in [Roy98] and [Neu05], which describe runtime systems for DAGs.

Another problem not discussed in detail is the cost model. This is related to the execution method, as the execution model determines the execution costs. Therefore, no general statement is possible. However, DAGs only make sense if the costs for sharing are low (ideally zero). This means that the input costs of an operator can no longer be determined by adding the costs of its input, as the input may overlap. This problem has not been studied as thoroughly as the execution itself. It is covered in [Neu05].

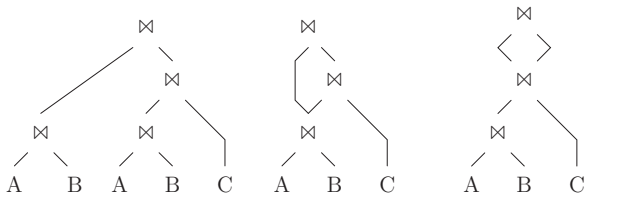



Figure 2: Invalid transformation for DAGs

# 4 Algebraic Optimization

In this section, we present a theoretical framework for DAG optimization. We first highlight three different aspects that differentiate DAG optimization from tree optimization. Then, we use these observations to formalize the reasoning over DAGs.

## 4.1 Using Tree Equivalences

Algebraic equivalences are fundamental to any plan generator: It uses them either directly by transforming algebraic expressions into equivalent ones, or indirectly by constructing expressions that are equivalent to the query. For tree-structured query graphs, many equivalences have been proposed (see e.g. [GMUW99, Mai83]). But when reusing them for DAGs, one has to be careful.

When only considering the join ordering problem, the joins are freely reorderable. This means that a join can be placed anywhere where its syntax constraints are satisfied (i.e. the join predicate can be evaluated). However, this is not true when partial results are shared. Let us demonstrate this by the example presented in Fig. 2. The query computes the same logical expression twice. In a) the join $A \bowtie B$ is evaluated twice and can be shared as shown in b). But the join with $C$ may not be executed before the split, as shown in c), which may happen when using a constructive approach to plan generation (e.g. dynamic programming or memoization) that aggressively tries to share relations and only considers syntax constraints. That is, a join can be build into a partial plan as soon as its join predicate is evaluable which in turn only requires that the referenced tables are present. This is the only check performed by a dynamic programming approach to join ordering. Intuitively, it is obvious that c) is not a valid alternative, as it means that $\bowtie C$ is executed on both branches. But in other situations, a similar transformation is valid, e.g. selections can often be applied multiple times without changing the result. As the plan generator must not rely on intuition, we now describe a formal method to reason about DAG transformations. Note that the problem mentioned above does not occur in current query optimization systems, as they treat multiple occurrences of the same relation in a query as distinct relations. But for DAG generation, the query optimizer wants to treat them as identical relations and thus potentially avoid redundant scans.

The reason why the transformation in Fig. 2 is invalid becomes clear if we look at the variable bindings. Let us denote by $A : a$ the successive binding of variable $a$ to members of a set $A$. In the relational context, $a$ would be bound to all tuples found in relation $A$. As shown in Fig. 3 a), the original expression consists of two different joins $A \bowtie B$

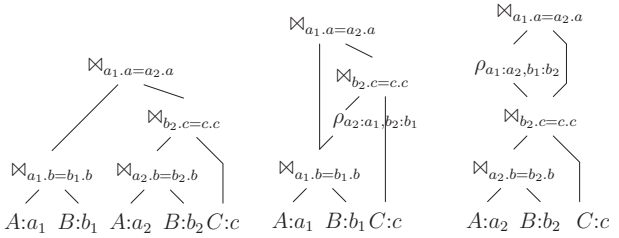



Figure 3: More verbose representation of Fig. 2

with different bindings. The join can be shared in b) by properly applying the renaming operator ($\rho$) to the output. While a similar rename can be used after the join $\bowtie C$ in c), this still means that the topmost join joins $C$ twice, which is different from the original expression.

This brings us to a rather surprising method to use normal algebra semantics:

> A binary operator must not construct a (*logical*) DAG.

Here, logical means that the same algebra expression is executed on both sides of its input. Further:

> What we do allow are *physical* DAGs, which means that we allow sharing operators to compute multiple logical expressions simultaneously.

As a consequence, we only share operators after proper renames: if an operator has more than one consumer, all but one of these must be rename operators. Thus, we use $\rho$ to pretend that the execution plan is a tree (which it is, logically) instead of the actual DAG.

### 4.2 Share Equivalence

Before going into more detail, we define whether two algebra expressions are *share equivalent*. This notion will express that *one expression can be computed by using the other expression and renaming the result*. Thus, given two algebra expressions $A$ and $B$, we define

$$A \equiv_S B \text{ iff } \exists_{\delta_{AB}:\mathcal{A}(A)\rightarrow\mathcal{A}(B),\, \delta_{AB}} \text{ bijective } \rho_{\delta_{AB}}(A) \equiv_D B.$$

where we denote by $\mathcal{A}(A)$ all the attributes provided in the result of $A$.

As this condition is difficult to test in general, we use a constructively defined sufficient condition of share equivalence instead. First, two scans of the same relation are share equivalent, since they produce exactly the same output (with different variable bindings):

$$scan_1(R) \equiv_S scan_2(R)$$

Note that in a constructive bottom-up approach, the mapping function $\delta_{A,B}$ is unique. Therefore, we always know how attributes are mapped.

$$
\begin{array}{rcll}
A \cup B & \equiv_S & C \cup D & \text{if } A \equiv_S C \wedge B \equiv_S D \\
A \cap B & \equiv_S & C \cap D & \text{if } A \equiv_S C \wedge B \equiv_S D \\
A \setminus B & \equiv_S & C \setminus D & \text{if } A \equiv_S C \wedge B \equiv_S D \\
\Pi_A(B) & \equiv_S & \Pi_C(D) & \text{if } B \equiv_S D \wedge \delta_{B,D}(A) = C \\
\rho_{a \to b}(A) & \equiv_S & \rho_{c \to d}(B) & \text{if } A \equiv_S B \wedge \delta_{A,B}(a) = c \wedge \delta_{A,B}(b) = d \\
\chi_{a:f}(A) & \equiv_S & \chi_{b:g}(B) & \text{if } A \equiv_S B \wedge \delta_{A,B}(a) = b \wedge \delta_{A,B}(f) = g \\
\sigma_{a=b}(A) & \equiv_S & \sigma_{c=d}(B) & \text{if } A \equiv_S B \wedge \delta_{A,B}(a) = c \wedge \delta_{A,B}(b) = d \\
A \times B & \equiv_S & C \times D & \text{if } A \equiv_S C \wedge B \equiv_S D \\
A \bowtie_{a=b}(B) & \equiv_S & C \bowtie_{c=d}(D) & \text{if } A \equiv_S C \wedge B \equiv_S D \wedge \delta_{A,C}(a) = c \wedge \delta_{B,D}(b) = d \\
\Gamma_{A;a:f}(B) & \equiv_S & \Gamma_{C;b:g}(D) & \text{if } B \equiv_S D \wedge \delta_{B,D}(A) = C \wedge \delta_{B,D}(a) = b \wedge \delta_{B,D}(f) = g
\end{array}
$$

Figure 4: Definition of share equivalence for common operators

Other operators are share equivalent if their input is share equivalent and their predicates are equivalent after applying the mapping function. The conditions for share equivalence for common operators are summarized in Fig. 4. They are much easier to check, especially when constructing plans bottom-up (as this follows the definition).

Note that share equivalence as calculated by the tests above is orthogonal to normal expression equivalence. For example, $\sigma_1(\sigma_2(R))$ and $\sigma_2(\sigma_1(R))$ are equivalent but *not* derivable as share equivalent by testing the sufficient conditions. This will not pose any problems to the plan generator, as it will consider both orderings. On the other hand, $scan_1(R)$ and $scan_2(R)$ are share equivalent, but not equivalent, as they may produce different attribute bindings.

Share equivalence is only used to detect if exactly the same operations occur twice in a plan and, therefore, cause costs only once. Logical equivalence of expressions is handled by the plan generator anyway, it is not DAG-specific.

Using this notion, the problem in Fig. 2 becomes clear: In part b), the expression $A \bowtie B$ is shared, which is ok, as $(A \bowtie B) \equiv_S (A \bowtie B)$. But in part c), the top-most join tries to also share the join with $C$, which is not ok, as $(A \bowtie B) \not\equiv_S ((A \bowtie B) \bowtie C)$. Note that while this might look obvious, it is not when e.g. constructing plans bottom up and assuming freely reorderable joins, as discussed in Section 3.1.

### 4.3 Optimizing DAGs

The easiest way to reuse existing equivalences is to hide the DAG structure completely: During query optimization, the query graph is represented as a tree, and only when determining the costs of a tree the share equivalent parts are determined and the costs adjusted accordingly. Only after the query optimization phase the query is converted into a DAG by merging share equivalent parts. While this reduces the changes required for DAG support to a minimum, it makes the cost function very expensive. Besides, if the query graph is already DAG-structured (e.g. for bypass plans), the corresponding tree-structured representation is much larger (e.g. exponentially for bypass plans), enlarging the search space accordingly.

A more general optimization can be done by sharing operators via rename operators. While somewhat difficult to do in a transformation-based plan generator, for a constructive plan generator it is easy to choose a share equivalent alternative and add a rename operator

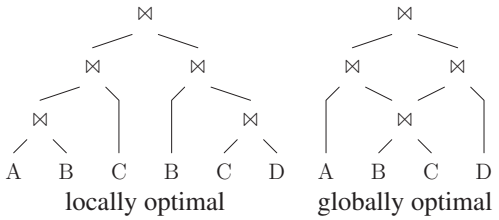



Figure 5: Possible non-optimal substructure for DAGs

as needed. Logically, the resulting plans behave as if the version without renaming was executed (i.e. as if the plan was a tree instead of a DAG). Therefore, the regular algebraic equivalences can be used for optimization. This issue will come up again when we discuss the plan generator.

### 4.4 (No) Optimal Substructure

Optimization techniques like dynamic programming and memoization rely on an optimal substructure of a problem (neglecting physical properties like sortedness or groupedness for a moment). This means that Bellmann's optimality principle holds and, thus, the optimal solution can be found by combining optimal solutions for subproblems. This is true for generating optimal tree-structured query graphs, but is not necessarily true for generating optimal DAGs. To see this, consider Fig. 5, which shows two plans for $A \bowtie B \bowtie C \bowtie B \bowtie C \bowtie D$. The plan on the left-hand side was constructed bottom-up, relying on the optimal substructure. Thus, $A \bowtie B \bowtie C$ was optimized, resulting in the optimal join ordering $(A \bowtie B) \bowtie C$. Besides, the optimal solution for $B \bowtie C \bowtie D$ was constructed, resulting in $B \bowtie (C \bowtie D)$. But when these two optimal partial solutions are combined, no partial join result can be reused. When choosing the suboptimal partial solutions $A \bowtie (B \bowtie C)$ and $(B \bowtie C) \bowtie D$, the expression $B \bowtie C$ can be shared, which might result in a better plan. Therefore, the optimal DAG cannot be constructed by just combining optimal partial solutions. Our approach avoids this problem by keeping track of sharing opportunities and considering them while pruning otherwise dominated plans.

### 4.5 Reasoning over DAGs

After looking at various aspects of DAG generation, we now give a formal model to reason about DAGs. More precisely, we specify when two DAGs are equivalent and when one DAG dominates another. Both operations are crucial for algebraic optimization.

As we want to lift tree equivalences to DAGs, we need some preliminaries: We name the equivalences for trees $\equiv_T$ and assume that the following conditions hold for all operators $\theta$, that is the equivalence can be checked on a per operator basis.

$$\begin{aligned} t \equiv_T t' &\Rightarrow \theta(t) \equiv_T \theta(t') \\ t_1 \equiv_T t_1' \wedge t_2 \equiv_T t_2' &\Rightarrow (t_1 \theta t_2) \equiv_T (t_1' \theta t_2') \end{aligned}$$

These conditions are a fundamental requirement of constructive plan generation, but as

seen in Sec. 4.1, they do no longer hold for DAGs in general. However, they hold for DAGs $(V, E)$ that logically are a tree ($LT$), i.e. that have non-overlapping input for all operators. For an expression $e$ let $\mathcal{A}(e)$ denote the set of attributes produced by $e$. We then define $LT((V, E))$ for a DAG $(V, E)$ as follows:

$$LT((V,E)) \quad \text{iff} \quad \forall v, v_1, v_2 \in V, (v, v_1) \in E, (v, v_2) \in E : \\ \mathcal{A}(v_1) \cap \mathcal{A}(v_2) = \emptyset$$

Note that this definition implies that all sharing of intermediate results must be done by renaming attributes.

Using this definition, we can now lift tree equivalences $\equiv_T$ to DAG equivalences $\equiv_D$ for DAGs $d$ and $d'$: We reuse the tree equivalences directly, but thereby must make sure that the input indeed behaves like a tree, as the equivalences were originally only defined on trees:

$$d \equiv_T d' \wedge LT(d) \wedge LT(d') \quad \Rightarrow \quad d \equiv_D d'$$

Note that the condition $LT(d) \Rightarrow LT(d' \subseteq d)$ holds, therefore partial DAGs that violate the logical tree constraint should be discarded immediately. As a consequence, tests for $LT(d)$ are required only when adding binary (or n-ary) operators, as unary operators cannot produce a new violation.

While lifting the tree equivalences to DAGs is important, they are not enough for DAG optimization, as they only create trees. DAGs can be created by using the notion of share equivalence defined above: If two DAGs $d$ and $d'$ are share equivalent, the two DAGs become equivalent by renaming the result suitably:

$$d \equiv_S d' \quad \Rightarrow \quad d \equiv_D \rho_{\mathcal{A}(d') \to \mathcal{A}(d)} d'$$

While these two implications are not exhaustive, lifting the tree equivalences to DAG equivalences and reusing intermediate results already allows for a wide range of DAG plans. Additional equivalences can be derived e.g. from [MFPR90, SPMK95].

In addition to checking if two plans are equivalent, the query optimizer has to decide which of two equivalent plans is better. Better usually means cheaper, according to a cost function. But sometimes two plans are incomparable, e.g. because one plan satisfies other ordering properties than the other. This is true for both, trees and DAGs. Here, we only look at the costs and DAG-specific limitations to keep the definitions short.

As shown in Sec. 4.4, DAGs cannot be compared by just looking at the costs, as one DAG might allow for more sharing of intermediate results than the other. To identify these plans, we require a labeling function that marks sharing opportunities. Here, we specify only the characteristics, see Section 5.3 for an explicit definition.

The DAGs are labeled using a function $S$. Its codomain is required to be partially ordered. We require that $S$ assigns the same label to share equivalent DAGs. Further the partial ordering between labels must express the fact that one DAG provides more sharing opportunities than another. Thus, for two DAGs $d_1 = (V_1, E_1)$ and $d_2 = (V_2, E_2)$ we require

the following formal properties for $S$:

$$S(d_1) = S(d_2) \quad \text{iff} \quad d_1 \equiv_S d_2$$
$$S(d_1) < S(d_2) \quad \text{iff} \quad \exists d_2' \subset_D d_2 : d_1 \equiv_S d_2'$$

Note that that $d_2' = (V_2', E_2') \subset_D d_2$ iff $V_2' \subset V_2 \wedge E_2' = E_{2|V_2'} \wedge d_2$ is a DAG).

Now a plan dominates another equivalent plan if it is cheaper and offers at least the same sharing opportunities:

$$d_1 \equiv_D d_2 \wedge costs(d_1) < costs(d_2) \wedge S(d_2) \leq S(d_1)$$
$$\Rightarrow \qquad d_1 \text{ dominates } d_2.$$

Note that the characterization of $S$ given above is overly conservative: one plan might offer more sharing opportunities than another, but these could be irrelevant for the current query. This is similar to order optimization, where the query optimizer only considers differences in interesting orderings [SAC+79, SSM96]. The algorithm presented in Section 5.3 improves the labeling by checking which of the operators could produce interesting shared intermediate results for the given query. As only whole subgraphs can be shared, the labeling function stops assigning new labels if any operator in the partial DAG cannot be reused. This greatly improves plan pruning, as more plans become comparable.

# 5 Plan Generator Skeleton

## 5.1 Overview

The main motivation for generating DAG structured execution plans is the ability to share intermediate results. Obviously, the fewer sharing opportunities are missed, the better. Hence, it is important to allow for a very aggressive sharing during plan generation.

In order to share intermediate results, the plan generator has to detect that partial plans produce the same (or rather equivalent) output. This is a problem for a rule-based plan generator, as the set of operators cannot be hard-coded and tests for equivalence are expensive. We overcome this problem by using a very abstract description of logical plan properties. Instead of using a complex property vector like, e.g., [Loh88], the plan generator uses a set of *logical properties*. The semantics of these logical properties is only known to the rules. They guarantee that two plans are equivalent if they have the same logical properties. A suitable representation is e.g. a bit vector with one bit for each property.

We assume that the logical properties are set properly by the optimization rules. This is all the plan generator has to know about the logical properties! However, to make things more concrete for the reader, we discuss some possible logical properties here. The most important ones are ”relation / attribute available” and ”operator applied”. In fact, these properties are already sufficient when optimizing only selections and joins as both are freely reorderable. We discuss more complex properties in Section 5.2.

As the logical properties describe equivalent plans, they can be used to partition the search space. This allows for a very generic formulation of a plan generator:

```
PLANGEN(goal)
1  bestPlan ← NIL
2  for each optimization rule r
3  do if r can produce a logical property in goal
4        then rem ← goal \ {p|p is produced by r}
5             part ← PLANGEN(rem)
6             p ← BUILDPLAN(r, part)
7             if bestPlan = NIL or p is cheaper
8               then bestPlan ← p
9  return bestPlan
```

This algorithm performs a top-down exploration of the search space. Starting with the complete problem (i.e. finding a plan that satisfies the query), it asks the optimization rules for partial solutions and solves the remaining (sub-) problem(s) recursively. For SPJ queries, this implies splitting the big join into smaller join problems (and selections) and solving them recursively. This algorithm is highly simplified, it only supports unary operators, does not perform memoization etc. Adding support for binary operators is easy but lengthy to describe, see [NM08] for a detailed discussion. Other missing and essential details will be discussed in Section 5.4. Let us summarize the main points of the above approach:

1. The search space is represented by abstract logical properties,
2. the plan generator recursively solves subproblems, which are fully specified by property combinations, and
3. property combinations are split into smaller problems by the optimization rules.

This approach has several advantages. First, it is very extensible, as the plan generator only reasons about abstract logical properties: the actual semantics is hidden in the rules. Second, it constructs DAGs naturally: if the same subproblem (i.e. the same logical property combination) occurs twice in a plan, the plan generator produces only one plan and thus creates a DAG. In reality, this is somewhat more complex, as queries usually do not contain exactly the same subproblem twice (with the same variable bindings etc.), but as we will see in Section 5.4, the notion of share equivalence can be used to identify sharable subplans.

### 5.2 Search Space Organization

An unusual feature of our plan generator is that it operates on an abstract search space, as the semantics of the logical properties are only known to concrete optimization rules. To give an intuition why this is a plausible concept, we now discuss constructing suitable properties for commonly used queries.

The most essential part of choosing properties is to express the query semantics properly. In particular, two plans must only have the same logical properties when they are equivalent. For the most common and (simplest) type of queries, the selection-projection-join (SPJ) queries using inner joins, it is sufficient to keep track of the available relations (and attributes) and the applied operators. For this kind of queries, two plans are equivalent if they operate on the same set of relations and they have applied the same operators. Therefore using the available attributes/relations and the applied operators as properties is sufficient for this kind of queries. Note that the properties are not only used by the plan

generator but also by the optimization rules, e.g. to check if all attributes required by a selection are available.

For more general queries involving e.g. outer joins, the same properties are sufficient, but the operator dependencies are more complex: Using the extended eligibility list concept from [RLL$^+$01], each operator specifies which other operators have to be applied before it becomes applicable. These operator dependencies allow for handling complex queries with relatively simple properties, and can be extended to express dependent subqueries etc. Note that we do not insist on a "dense" encoding of the search space that only allows for valid plans (which is desirable, but difficult to achieve in the presence of complex operator). Instead, we allow for a slightly sparse encoding and then guarantee that we only explore the valid search space.

Note that these properties are just examples, which are suitable for complex queries but not mandatory otherwise. The plan generator makes only two assumptions about the logical properties: Two plans with the same properties are equivalent and properties can be split to form subproblems. Further, not all information about plans have to be encoded into the logical properties. In our implementation, ordering/grouping properties are handled separately, as the number of orderings/groupings can be very large. We use the data structure described in [NM04] to represent them. This implies that multiple plans with the same logical properties might be relevant during the search, as they could differ in some additional aspects. We will see another example for additional properties in the next section.

### 5.3 Sharing Properties

Apart from the logical properties used to span the search space, each plan contains a *sharing* bit set to indicate potentially shared operators. It can be considered as the materialization of the labeling function $S$ used in Section 4.5: The partial ordering on $S$ is defined by the subset relation, that is, one plan can only dominate another plan if it offers at least the same sharing opportunities. We now give a constructive approach for computing $S$.

When considering a set of share equivalent plans, it is sufficient to keep one representative, as the other plans can be constructed by using the representative and adding a rename (note that all share equivalent plans have the same costs). Analogously, the plan generator determines all operators that are share equivalent (more precisely: could produce share equivalent plans if their subproblems had share equivalent solutions) and places them in equivalence classes. As a consequence, two plans can only be share equivalent if their top-most operators are in the same equivalence class, which makes it easier to detect share equivalence. The equivalence classes that contain only a single operator are discarded, as they do not affect plan sharing. For the remaining equivalence classes, one representative is selected and one bit in the *sharing* bit set is assigned to it.

For example, the query in Fig. 5 consists of 11 operators: $A$, $B_1$, $C_1$, $B_2$, $C_2$, $D$, $\bowtie_1$ ($A$ and $B_1$), $\bowtie_2$ (between $B_1$ and $C_1$), $\bowtie_3$ (between $B_2$ and $C_2$), $\bowtie_4$ (between $C_2$ and $D$) and $\bowtie_5$ (between $A$ and $D$). Then three equivalence classes with more than one element can be constructed: $B_1 \equiv_S B_2$, $C_1 \equiv_S C_2$ and $\bowtie_2 \equiv_S \bowtie_3$. We assume that the operator with the smallest subscript was chosen as representative for each equivalence class. Then the plan generator would set the *sharing* bit $B_1$ for the plan $B_1$, but not for the plan $B_2$. The plan $A \bowtie_1 (B_1 \bowtie_2 C_1)$ would set sharing bits for $B_1$, $C_1$ and $\bowtie_2$, as the subplan can be shared, while the plan $(A \bowtie_1 B_1) \bowtie_2 C_1$ would only set the bits $B_1$ and $C_1$, as the join $\bowtie_2$

cannot be shared here (only whole subgraphs can be shared). The sharing bit set allows the plan generator to detect that the first plan is not dominated by the second, as the first plan allows for more sharing. This solves the problem discussed in Section 4.4.

The equivalence classes are also used for another purpose: When an optimization rule requests a plan with the logical properties produced by an operator, the plan generator first checks if a share equivalent equivalence class representative exists. For example, if a rule requests a plan with $B_2$, $C_2$ and $\bowtie_3$, the plan generator first tries to build a plan with $B_1$, $C_1$ and $\bowtie_2$, as these are the representatives. If this rewrite is possible (i.e. a plan could be constructed), the plan constructed this way is also considered a possible solution.

In general, the plan generator uses sharing bits to explicitly mark sharing opportunities: whenever a partial plan is built using an equivalence class representative, the corresponding bit is set. In more colloquial words: the plan *offers* to share this operator. Note that it is sufficient to identify the selected representative, as all other operators in the equivalence class can be built by just using the representative and renaming the output. As sharing is only possible for whole subplans, the bit must only be set if the input is also sharable. Given, for example, three selections $\sigma_1$, $\sigma_2$ and $\sigma_3$, with $\sigma_1(R) \equiv_S \sigma_2(R)$. The two operator rules for $\sigma_1$ and $\sigma_2$ are in the same equivalence class, we assume that $\sigma_1$ was selected as representative. Now the plan $\sigma_1(R)$ is marked as "shares $\sigma_1$", as it can be used instead of $\sigma_2(R)$. The same is done for $\sigma_3(\sigma_1(R))$, as it can be used instead of $\sigma_3(\sigma_2(R))$. But for the plan $\sigma_1(\sigma_3(R))$ the *sharing* attribute is empty, as $\sigma_1$ cannot be shared (since $\sigma_3$ cannot be shared). The plans containing $\sigma_2$ do not set the *sharing* property, as $\sigma_1$ was selected as representative and, therefore, $\sigma_2$ is never shared.

Note that the sharing bits are only set when the whole plan can be shared, but the already existing sharing bits are still propagated even if it is no longer possible to share the whole plan. Otherwise a slightly more expensive plan might be pruned early even though parts of it could be reused for other parts of the query.

Explicitly marking the sharing opportunities serves two purposes. First, it is required to guarantee that the plan generator generates the optimal plan, as one plan only dominates another if it is cheaper and offers at least the same sharing opportunities. Second, sharing information is required by the cost model, as it has to identify the places where a DAG is formed (i.e. the input overlaps). This can now be done by checking for overlapping *sharing* properties. It is not sufficient to check if the normal logical properties overlap, as the plans pretend to perform different operations (which they do, logically), but share physical operators.

### 5.4 Search Space Exploration under Sharing

After describing the general approach for plan generation, we now present the specific steps required to handle shared plans. Unfortunately, the plan generator is not a single, concise algorithm but a set of algorithms that are slightly interwoven with the optimization rules. This is unavoidable, as the plan generator has to be as generic as possible and, therefore, does not understand the semantics of the operators. However, there is still a clear functional separation between the different modules: The *plan generator* itself maintains the partial plans, manages the memoization and organizes the search. The *optimization rules* describe the semantics of the operators and guide the search with their requirements. For the specific query, several optimization rules are *instantiated*, i.e. annotated with the

query specific information like selectivities, operator dependencies etc. Typically each operator in the original query will cause a rule instantiation. Note that we only present a simplified plan generator here to illustrate the approach, in particular the search space navigation. The detailed algorithms are discussed in [NM08].

Within the search space, the plan generator tries to find the cheapest plan satisfying all logical properties required for the final solution. Note that in the following discussion, we ignore performance optimization to make the conceptual structure clearer. In practice, the plan generator prunes plans against known partial solutions, uses heuristics like KBZ [KBZ86] to get upper bounds for costs etc. However, as these are standard techniques, we do not elaborate on them here.

The core of the plan generator itself is surprisingly small and only consists of a single function that finds the cheapest plans with a given set of logical properties. Conceptually this is similar to the top-down optimization strategy known from Volcano [Gra95, GM93]. The search phase is started by requesting the cheapest plan that provides the goal properties.

```
PLANGEN(goal)
 1  plans ← memoizationTable[goal]
 2  if plans already computed
 3    then return plans
 4  plans ← create a new, empty PlanSet
 5  shared ← goal rewritten using representatives
 6  if shared ∩ goal = ∅
 7    then plans ← PLANGEN(shared)
 8  for each r in instantiated rules
 9  do filter ← r.produces ∪ r.required
10    if filter ⊆ goal
11      then sub ← PLANGEN(goal \ r.produced)
12           plans ← plans ∪ {r(p)|p ∈ sub}
13  memoizationTable[goal] ← plans
14  return plans
```

What is happening here is that the plan generator is asked to produce plans with a given set of logical properties. First, it checks the memoization data structure (e.g. a hash table, logical properties→plan set) to see if this was already done before. If not, it creates a new set (initially empty) and stores it in the memoization structure. Then, it checks if the goal can be rewritten to use only representatives from equivalence classes as described in Section 5.3 (if the operators $o_1$ and $o_2$ are in the same equivalence class, the logical properties produced by $o_2$ can be replaced by those produced by $o_1$). If the rewrite is complete, i.e., the new goal is disjunct from the original goal ($shared \cap goal = \emptyset$), the current problem can be formulated as a new one using only share equivalent operators. This is an application of the DAG equivalence given in Section 4.5. Thus, the plan generator tries to solve the new problem and adds the results to the set of usable plans. Afterwards, it looks at all rule instances, checks if the corresponding filter is a subset of the current goal (i.e. the rule is relevant) and generates new plans using this rule. Note that the lines 8-12 are very simplified and assume unary operators. In practice, the optimizer delegates the search space navigation (here a simple $goal \setminus r.produced$) to the rules.

# 6 Evaluation

In the previous sections, we discussed several aspects of optimizing DAG-structured query graphs. However, we still have to show three claims: 1) creating DAG-structured query plans is actually beneficial for common queries, 2) sharing base scans as proposed by Cao et al. [CDCT08] is inferior to full DAG support, and 3) the overhead of generating DAG-structured plans is negligible. Therefore, we present several queries for which we create tree-structured and DAG-structured query plans. Both the compile time and the runtime of the resulting plans are compared to see if the overhead for DAGs is worthwhile. For the DAG plans, we created the full DAGs in a single phase optimizer using our DAG reasoning, and the scan sharing plans by a two phase optimizer based upon the tree plans as proposed in [CDCT08]. All experiments were executed on a 2.2 GHz Athlon64 system running Windows XP. The plans were executed using a runtime system that could execute DAGs natively [Neu05].

Each operator (join, group-by etc.) is given 1MB buffer space. This is somewhat unfair against the DAG-structured query plans, as they need fewer operators and could therefore allocate larger buffers. But dynamic buffer sizes would affect the cost model, and the space allocation should probably be a plan generator decision. As this is beyond the scope of this work, we just use a static buffer size here.

As a comparison with the state of the art in commercial database systems, we included results for DB2 9.2.1, which factorizes common subexpressions using materialization of intermediate results. As we were unable to measure the optimization time accurately enough, we only show the total execution time (which includes optimization) for DB2.

## 6.1 TPC-H

The TPC-H benchmark is a standard benchmark to evaluate relational database systems. It tests ad-hoc queries which result in relatively simple plans allowing for an illustrative comparison between tree- and DAG-structured plans. We used a scale factor 1 database (1GB).

Before looking at some exemplary queries, we would like to mention that queries without sharing opportunities are unaffected by DAG support: The plan generator produces exactly the same plans with and without DAG support, and their compile times are identical. Therefore, it is sufficient to look at queries, which potentially benefit from DAGs.

### Query 11

Query 11 is a typical query that benefits from DAG-structured query plans. It determines the most important subset of suppliers' stock in a given country (Germany in the reference query). The available stock is determined by joining `partsupp`, `supplier` and `nation`. As the top fraction is requested, this join is performed twice, once to get the total sum and once to compare each part with the sum. When constructing a DAG, this duplicate work can be avoided. The compile time and runtime characteristics are shown below:

| | tree | Scan Sharing | Full DAG | DB2 |
|---|---|---|---|---|
| compilation [ms] | 10.5 | 10.5 | 10.6 | - |
| execution [ms] | 4793 | 4256 | 2436 | 3291 |

While the compile time is slightly higher when considering DAGs (profiling showed this is due to the checks for share equivalence), the runtime is much smaller. The corresponding plans are shown in Fig. 6: In the tree version, the relations `partsupp`, `supplier` and `nation` are joined twice, once to get the total sum and once to get the sum for each part. In the DAG version, this work can be shared, which nearly halves the execution time. The scan sharing approach from [CDCT08] only slightly better than the tree plan as it still has to join twice. DB2 performs between trees and full DAGs, as it reuses intermediate results but materializes to support multiple reads.

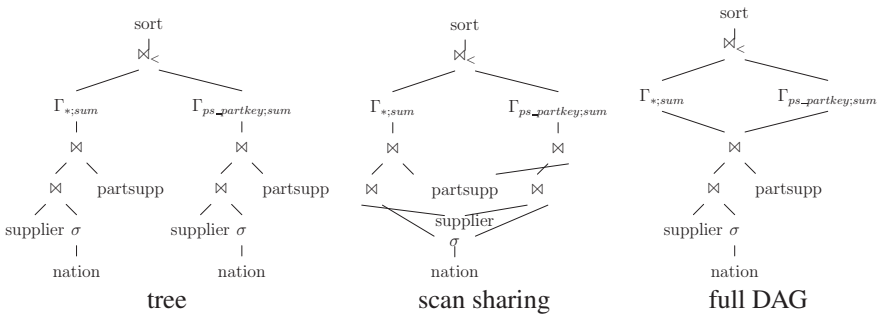



Figure 6: Execution plans for TPC-H Query 11

### Query 2

Query 2 selects the supplier with the minimal supply costs within a given region. Structurally, this query is similar to Query 11, as it performs a large join twice, once for the result and once to get the minimum. However, it is more complex, as the nested minimum query depends on the outer query. We assume that the rewrite step unnests the query (by grouping and using a join), but still the nested query lacks a relation used in the outer query, which prevents sharing the whole join. To allow for greater sharing, the relation (and the corresponding predicates) can be re-added by a magic set like transformation [MFPR90]. Here, we consider three alternatives: Normal tree construction, DAG construction and DAG construction with rules for magic set transformation enabled. We omit the sharing approach from [CDCT08] here, as it is nearly identical to the second alternative for this query (one additional (cheap) join). Compile time and runtime are shown below.

| | tree | DAG | DAG + magic set | DB2 |
|---|---|---|---|---|
| compilation [ms] | 9.3 | 9.2 | 9.7 | - |
| execution [ms] | 11933 | 7480 | 3535 | 8705 |

The compile times for tree and DAG are about the same (the DAG is slightly faster, as it can ignore some dominated alternatives), while the magic set variant is about 5% slower due to the increased search space. The runtime behavior of the alternatives is very different. Fig. 7 shows the plans. The tree variant simply calculates the outer query and the nested query independently and joins the result. The DAG variant tries to reuse some intermediate results (reducing the runtime by 37%). It still performs most of the joins in both parts, as the subqueries are not identical. It would be possible to share more by applying the join with *part* later, but this does not pay off due to selectivity of the join. When using the magic set transformation, large parts of the query become equivalent, which results in much greater sharing and also reduced aggregation effort. The consequence is a runtime reduction by 70% compared to the tree variant. DB2 performs better than the tree plan (again due to materializing intermediate results), but worse than both DAG plans.

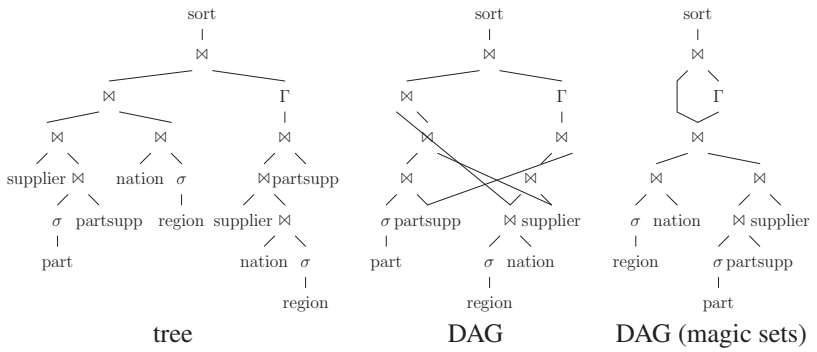



Figure 7: Execution plans for TPC-H Query 2

### 6.2 Conclusion

The experiments have shown that by considering sharing intermediate results, the compile time is mainly unaffected. It is somewhat affected when additionally magic sets are used (Query 2). Still, the compile time is negligible and clearly dominated by the runtime. The runtime effect of DAG support is non-negligible, as sharing can drastically reduce the runtime of many queries. Further, the real benefit comes from sharing intermediate results. Simply sharing scans as proposed in [CDCT08] improves runtimes a bit, but not nearly as much as full DAG support.

## 7 Conclusion

We presented the first plan generator which exploits the full potential of shared subplans by being able to explore the complete search space in a fashion that integrates reasoning about sharing and plan generation.

As we have seen, generating DAG-structured query evaluation plans is not easy. To solve the problem, we introduced the novel notion of *share equivalence* and saw that it is orthogonal to the standard notion of equivalence. Rules to reason about share equivalence made this hard to decide notion manageable for the plan generator.

In order to be able to build upon well-known algebraic equivalences, we separated regular equivalence and share equivalence. The former notion is applied only to logical plans which thus remain tree-structured. Reasoning about sharing is moved exclusively to the physical level. The central idea here was to annotate sharing possibilities explicitly and let the plan generator reason about them. To make this more efficient, plans are grouped into equivalence classes where those plans leading to the same sharing possibilities find themselves in the same class. Thereby, the reasoning step was reduced to a simple and efficient rewrite of the optimization goal. The resulting plan generator is the first one that guarantees two important features: (1) all sharing possibilities are detected and considered, and (2) the resulting plan is guaranteed to be optimal. Finally, experiments demonstrated that compared to generating tree-structured plans, the plan generation overhead is negligible, while the gains during query execution are tremendous.

There is plenty of room for future research. Let us mention just two important areas. The first is to develop and explore more optimization techniques that inherently require DAG support. With bypass plans, we have already seen an example of them. The second area where DAGs play a crucial role is stream processing.

**Acknowledgment:** We thank Simone Seeger for her help preparing the manuscript.

## References

[BBD+04] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Dilys Thomas. Operator scheduling in data stream systems. *VLDB J.*, 13(4):333–353, 2004.

[CDCT08] Y. Cao, G. Das, C.-Y. Chan, and K.-L. Tan. Optimizing Complex Queries with Multiple Relation Instances. In *SIGMOD*, pages 525–538, 2008.

[Cha98] Don Chamberlin. *A complete guide to DB2 universal database*. San Francisco, CA, USA, 1998.

[DSRS01] Nilesh N. Dalvi, Sumit K. Sanghai, Prasan Roy, and S. Sudarshan. Pipelining in Multi-Query Optimization. In *PODS*, 2001.

[GLJ01] César A. Galindo-Legaria and Milind Joshi. Orthogonal Optimization of Subqueries and Aggregation. In *SIGMOD*, 2001.

[GLSW93] Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, and Yun Wang. Query Optimization in the IBM DB2 Family. *IEEE Data Eng. Bull.*, 16(4):4–18, 1993.

[GM93] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE*, pages 209–218, 1993.

[GMUW99] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database System Implementation*. 1999.

[Gra90] Goetz Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. In *SIGMOD*, pages 102–111, 1990.

[Gra93] Goetz Graefe. Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[Gra94] Goetz Graefe. Volcano - An Extensible and Parallel Query Evaluation System. *IEEE Trans. Knowl. Data Eng.*, 6(1):120–135, 1994.

[Gra95] Goetz Graefe. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.

[HFLP89] Laura M. Haas, Johann Christoph Freytag, Guy M. Lohman, and Hamid Pirahesh. Extensible Query Processing in Starburst. In *SIGMOD*, pages 377–388, 1989.

[HSA05] Stavros Harizopoulos, Vladislav Shkapenyuk, and Anastassia Ailamaki. QPipe: A Simultaneously Pipelined Relational Query Engine. In *SIGMOD*, pages 383–394, 2005.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of Nonrecursive Queries. In *VLDB'86*, pages 128–137, 1986.

[Loh88] Guy M. Lohman. Grammar-like Functional Rules for Representing Query Optimization Alternatives. In *SIGMOD*, pages 18–27, 1988.

[Lor74] Raymond A. Lorie. XRM - An Extended (N-ary) Relational Memory. *IBM Research Report*, G320-2096, 1974.

[Mai83] David Maier. *The Theory of Relational Databases*. 1983.

[MFPR90] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is Relevant. In *SIGMOD*, pages 247–258, 1990.

[Neu05] Thomas Neumann. *Efficient Generation and Execution of DAG-Structured Query Graphs*. PhD thesis, Universität Mannheim, 2005.

[NM04] Thomas Neumann and Guido Moerkotte. An Efficient Framework for Order Optimization. In *ICDE*, pages 461–472, 2004.

[NM08] Thomas Neumann and Gudio Moerkotte. Single Phase Construction of Optimal DAG-structured QEPs. Research Report MPI-I-2008-5-002, Max-Planck-Institut für Informatik, 2008.

[RLL+01] Jun Rao, Bruce G. Lindsay, Guy M. Lohman, Hamid Pirahesh, and David E. Simmen. Using EELs, a Practical Approach to Outerjoin and Antijoin Reordering. In *ICDE*, pages 585–594, 2001.

[Roy98] Prasan Roy. Optimization of DAG-Structured Query Evaluation Plans. M.tech. thesis, Dept. of Computer Science and Engineering, IIT-Bombay, January 1998.

[RSSB00] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In *SIGMOD*, 2000.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management System. In *SIGMOD*, pages 23–34, 1979.

[Sel88] Timos K. Sellis. Multiple-Query Optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[SPMK95] Michael Steinbrunn, Klaus Peithner, Guido Moerkotte, and Alfons Kemper. Bypassing Joins in Disjunctive Queries. In *VLDB*, pages 228–238, 1995.

[SSM96] David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental Techniques for Order Optimization. In *SIGMOD*, pages 57–67, 1996.

[ZLFL07] Jingren Zhou, Per-Åke Larson, Johann Christoph Freytag, and Wolfgang Lehner. Efficient exploitation of similar subexpressions for query processing. In *SIGMOD*, pages 533–544, 2007.

# Efficient verification of B-tree integrity

Goetz Graefe
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303
Goetz.Graefe@HP.com

Ryan Stonecipher
Microsoft
1 Microsoft Way
Redmond, WA 98052
Ryan.Stonecipher@Microsoft.com

**Abstract**: The integrity of B-tree structures can become compromised for many reasons. Since these inconsistencies manifest themselves in unpredictable ways, all commercial database management systems include mechanisms to verify the integrity and trustworthiness of an individual index and of a set of related indexes, and all vendors recommend index verification as part of regular database maintenance. This paper introduces algorithms for B-tree validation, reviews the algorithms' strengths and weaknesses, and proposes a simple yet effective improvement for key verification across multiple B-tree levels. The performance is such that B-tree verification can become part of scans or backups. Our experimental comparisons include algorithm performance and scalability measured using a shipping product.

# 1 Introduction

Even with the most carefully implemented and tested database software, database corruptions can happen at any time. There are many reasons: even shipping software has defects for which vendors eventually offer "fix packs" or "service packs;" there are software defects in operating systems and their device drivers; there is complex hardware and software in today's storage systems, e.g., in network-attached storage; and there are environmental problems such as excessive vibration or overheating in a storage rack. Each of these system layers may have its own verification methods for its data and metadata, but the database management system running on top of them must provide the final verification mechanism that detects failures and corruptions. Therefore, all database vendors provide verification utilities and recommend their use as part of regular maintenance. The purpose of this paper is to lay foundations of reliable and efficient verification algorithms and to introduce a superior new algorithm. This algorithm requires little memory yet is so simple and fast that it even can be integrated into backup and restore operations.

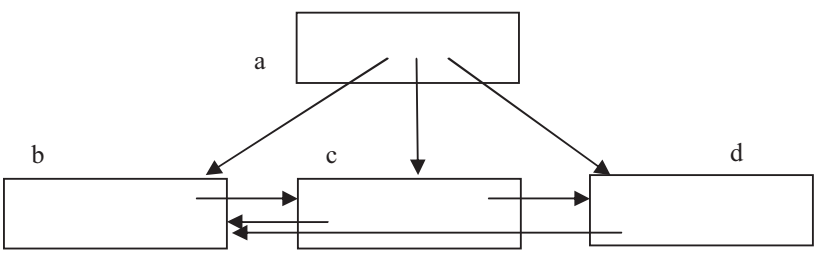



Figure 1. An incomplete leaf split.

For example, Figure 1 shows the result of an incomplete execution, an incomplete recovery, or an incomplete replication of splitting a leaf node. The cause might be a defect in the database software, e.g., in the buffer pool management, or in the storage management software, e.g., in snapshot or version management. When leaf node b was split and leaf node c was created, the backward pointer in successor node d incorrectly remained unchanged. A subsequent (descending) scan of the leaf level will produce a wrong query result, and subsequent split and merge operations will create further havoc.

Different B-tree inconsistencies result if all nodes but c are saved correctly in the database; in that case, a, b, and d point to a page full of garbage. If all nodes but b are saved correctly, the key range in b would not conform to the separator keys in a. If all nodes but a are saved correctly, a search for records in node c will not find them.

There are, of course, many failures and corruptions that may be encountered. A B-tree verification algorithm must be prepared to deal with and identify any of these cases. Our work focuses on the data structures defined and governed by the database management system, ignoring the effects and failure modes of the hardware and software underneath even if the software embedded in the so-called hardware is as large and complex as the database management system itself, e.g., to provide load management and distribution, snapshot backup, geo-replication, or redundancy with online recovery [PGK 88].

In many databases, the most fundamental data structure on disk is a variant of the well-known B-tree index [BM 72]. Therefore, verification of a B-tree's consistency is the fundamental operation we consider in this paper. As database indexes are usually in the form of B*-trees, we focus on those instead of the original B-tree design, and throughout we mean B*-trees when we mention B-trees. In addition, modern database management systems define, explicitly or implicitly, further consistency constraints, e.g., between multiple redundant indexes of a single table or between a materialized view and the base tables. Verification of those relationships is also considered here.

Allocation structures, metadata, and verification of their consistency are omitted here, because these differ too much between products to give such a discussion general value. For the purpose of code reuse alone, including code for consistency checking, B-tree indexes might be used to implement the allocation data structures. In this case, the problem of booting a consistency check arises: if the allocation information is kept in a B-tree, but that B-tree is not verified yet, where should database verification start? Within this paper, we ignore this problem and focus on verification of user tables and their B-tree indexes.

The variety of possible algorithms for B-tree verification is actually surprising. The main issue is verification of the complex web of pointers and keys, and of the many consistency constraints implied in a B-tree's structure. At first sight, nothing but a careful traversal of the tree structure seems safe and complete. If this method were the only possible method, dropping and re-creating indexes could be more efficient than verification.

We have found, however, that orders-of-magnitude faster algorithms can be employed. This can be achieved by dividing the problem into sub-problems, by dividing sub-problems into their contributing facts, and by aggregating such facts and verifying the result. Moreover, relational database systems have built-in mechanisms for implementing such algorithms, namely the query processor. We have found that both query optimization and query execution can contribute to the efficiency of B-tree verification.

Algorithms based on these insights have been shipping and used regularly on millions of production systems for years. Their introduction reduced the times tables had to be

read-only or offline and thus increased application availability. Recently, however, we realized that another substantial improvement is still possible. This improvement reduces the required effort to a level that permits B-tree verification as side effects of scans, e.g., during backup and restore operations where B-tree verification would serve to increase the confidence in backup copies and would thus increase their value.

Therefore, the contributions of this research are (i) the introduction, survey and comparison of multiple alternative approaches to verification of structural integrity of entire B-trees, (ii) a description of a fast verification algorithm invented for SQL Server, (iii) the description of actual implementation and usage of multiple algorithms in a commercial product, (iv) identification of opportunities for further improvements of these algorithms, (v) the evaluation of the performance and scalability of alternative algorithms in the context of a shipping product, and (vi) a simpler and faster method for verifying large multi-level B-trees. Automatic repair is beyond the scope of this paper.

After a review of related work, the two main sections of the paper survey alternative algorithmic approaches to B-tree verification, including details on opportunities for improvement, and report on our performance evaluation. We then offer some recommendations and our conclusions from this work.

# 2 Related work

Several researchers have analyzed frequency and causes of disk errors. For example, Bairavasundaram et al. [BGS 08] found "more than 400,000 instances of checksum mismatches over the 41-month period," affecting almost 1% of the monitored near-line disks. Even for disk arrays with redundancy, Hafner et al. [HDB 08] observe trends that are "are increasing the likelihood of undetected disk errors", which "can cause silent data corruption that may go completely undetected (until a system or application malfunction) or may be detected by software in the storage I/O stack." Our goal is to detect corruptions that manifest themselves as inconsistent B-tree indexes, e.g., those illustrated in Figure 1.

Any paper on B-trees, their structures and their algorithms, owes a debt to early work that defined and popularized B-trees [BM 72, C 79]. Their defining characteristics are uniform root-to-leaf distance, multiple keys and child pointers per interior node, and (in many implementations) sibling pointers among nodes on the same level [GR 93]. Subsequent work has made many important improvements, e.g., compression [BU 77, PP 03].

Unfortunately, verification of physical integrity of databases has not received much attention in database research. Logical verification, both initial verification of new explicitly declared consistency constraints and incremental maintenance of previously declared constraints, has been researched in the past [RSS 96, S 75]. The last one of the techniques described below is related to these prior ones, i.e., referential integrity and foreign key constraints are similar to the relationship between a clustered index and its non-clustered indexes. The other techniques described in the following sections, however, are not related to logical database verification because those prior publications did not consider verification of the tree structure and its many pointers, assuming instead that access methods are unconditionally reliable.

Mohan described the danger of partial writes due to performance optimizations in the SCSI standard [M 95]. His focus was on problem prevention using appropriate page

modification, page verification after each read operation, logging, log analysis, recovery logic, etc. The complexity of these techniques, together with the need for ongoing improvements in these performance-critical modules, reinforces our belief that complete, reliable, and efficient verification of B-tree structures is a required defensive measure.

Küspert investigated index verification techniques in the 1980s [K 85], with a focus on local verification as a side effect of ordinary B-tree accesses during query and update processing. For example, page format and contents can be verified before interpreting a database page recently fetched from disk. Page verification may include its relationship to its parent page. Parent and child must belong to successive levels of the same B-tree, and the key values in the child must belong into the key interval defined by the parent. The parent page usually also contains information about a node's neighbors that can be compared to the sibling pointers in a child node during a root-to-leaf B-tree search.

Our techniques are compatible with and complementary to these prior verification techniques. In fact, we present an extension to Küspert's techniques. Our overall goal is different, however, as our techniques permit complete assurance for an entire B-tree or even all B-tree indexes belonging to a table, view, database, or backup.

Prior techniques used in Siemens software reduced verification of B-tree consistency to set comparisons, e.g., the set of B-tree nodes, the set of parent-child pointers, and the sets of left and right sibling pointers [W 81, summarized in K 85]. Expensive set operations were replaced by checksums. Set comparisons are also the conceptual basis of our methods based on aggregation and bitmaps. Our algorithms differ from the earlier ones in two ways, however. First, our algorithms permit gathering much less data in the second pass needed after the first pass has found that a problem exists. Second, our algorithms verify not only the pointer structure but also the ordering relationships of keys among sibling nodes and among ancestor and descendent nodes. Key verification cannot be added to the Siemens algorithms because checksums support only equality comparisons whereas key comparisons also require "less than" comparisons.

Some of the discussion below proposes changes to the traditional page layout for B-trees. Optimization of page layouts, in particular with respect to CPU caches, has been researched in multiple efforts during the last decade. However, these efforts have been focused on CPU caches rather than on the B-tree structure. Prior research has suggested the same change in page layout as is proposed below, but failed to appreciate its importance for B-tree verification despite mentioning the subject [G 04].

SQL Server is used as reference implementation in this paper. Any other product would serve just as well, probably with similar issues, problems, insights, and results. Like other database products, SQL Server stores variable-length records in B-trees with fixed-length nodes or pages. It uses pages of 8 KB, with special techniques for records exceeding this size. An indirection vector within each page indicates the location of records using byte offsets. This indirection vector grows from the high end of the page such that the records may grow from the low end.

Another design decision in SQL Server B-tree pages is that all records in an interior node have the same layout including both a key and a pointer, i.e., the number of keys in an interior node equals the number of pointers. When an interior node is split, entire records are moved from the old node to the new node, such that the new right node contains not only separator keys but also the low boundary key. While this design is common [GR 93], it is not ubiquitous. In SQL Server, it is exploited in a way described later.

Finally, each table may have one clustered index and multiple non-clustered B-tree indexes. The clustered index serves as the primary, non-redundant data store, whereas the non-clustered indexes reference rows in the clustered index physically (using page number and slot number) or logically (using a search key in the clustered index) – SQL Server switched from the former to the latter design choice in release 7.

SQL Server does not employ a technique called $B^{link}$-trees [LY 81]. For high concurrency while splitting nodes, these trees permit side pointers to new nodes that are not yet referenced in the parent node, with repair at the earliest opportunity. Search is directed to the node at hand or its right neighbor using a separator key associated with the side pointer. This is the same key value eventually posted as separator key in the parent node. Verification of $B^{link}$-trees is not covered explicitly in this paper, but the algorithms can be adapted with moderate effort and complexity. It is interesting to note, however, that both a low boundary key within each node and a high boundary key within each node are already common or have already been proposed elsewhere. They will be exploited in some of the verification algorithms to be discussed.

# 3 Verification algorithms

In the algorithms described here, we assume that there are no concurrent transactions updating the indexes and tables being checked. This assumption is true if the verification operation holds a table-level shared lock that covers the metadata, all indexes, etc., or if the verification operation is isolated from concurrent activity by other means, e.g., by copy-on-write semantics implemented within the database or a lower software layer.

We further assume that complete verification is required. All following algorithms are able to verify all constraints in a B-tree index such that successful completion guarantees complete consistency. A database vendor is free to choose among these algorithms based on the desired performance, scalability, engineering effort, etc.

B-tree verification can be divided into multiple aspects. First, each individual page must be consistent, and the first section below describes current and possible techniques that apply to each individual B-tree node. Second, the network of interrelated B-tree nodes must be verified, including child pointers and neighbor pointers as well as key ranges and separator keys. Third, if multiple B-trees store a relational table, these must be verified against each other. For example, a table's clustered and non-clustered indexes must contain the same count of valid records in order to represent the same set of logical rows. Finally, the relationship among tables must be verified, e.g., explicitly declared foreign key constraints, as well as the relationship between tables and materialized and indexed views. The following sections review appropriate algorithms for these problems.

Verification algorithms can produce various forms of output, from a binary decision whether or not a B-tree is currently consistent all the way to a specific identification of each individual problem. Between those extremes fall algorithms that describe the type or approximate location of existing problems. If corruptions are fairly frequent even for small data volumes, specific information is worthwhile even if it is the most expensive to obtain. On the other hand, if corruptions are very rare and a simple binary decision is least expensive to obtain, that type of algorithm has the greatest value, assuming, of course, that identification of specific problems is possible if a corruption exists. The in-

termediate algorithms trade off performance against output information, such that the output information is less specific but finding the actual corruption, should one be found to exist, is less expensive than the naïve algorithm.

In the following sections, we point out the verification algorithms' most salient aspects, their advantages and disadvantages, their usage of the algorithm in SQL Server, and opportunities for improvements. Discussions of advantages and disadvantages as well as of future opportunities apply to any database management system relying on B-tree indexes. SQL Server is used to provide examples of each algorithm's usage and history rather than convey a limitation to a specific product or release.

## *3.1 In-page verification*

Verification of information within a single B-tree node is fairly straightforward. It is described here in some detail for the sake of completeness. Specific systems and their data structures design require appropriate modifications.

It should start with a physical test to protect against partial or "torn" write or read operations [M 95], i.e., cases in which a B-tree node (e.g., 8 KB) requires multiple disk sectors (e.g., of 512 B) and only some of them are written or read due to a problem in electrical power, vibration, etc. Using a checksum operation of all words in the page or of only a select few ones (say every 512$^{th}$ byte) gives reasonable assurance that the in-memory image read from disk and the one prior to the last write are indeed equal.

**Opportunities** are plentiful to go beyond the basic algorithm described above. First, torn-page detection can be made sufficiently fast that it is worthwhile to integrate it into all read operations in the database management system. The main expense of torn-page detection is the number of misses in the CPU cache, but even those may be worthwhile if the I/O hardware is feared not to be very reliable. Even further verification steps could be integrated into the I/O operations and might be worthwhile in some systems.

In addition to verification, the same database utility could add general maintenance tasks within each page. If so, database or index verification improves not only confidence in the system's correctness but also performance of subsequent operations. Maintenance tasks can be deeply integrated in the verification task or they can be scheduled to follow the verification tasks, focusing on opportunities identified by the verification utility.

## *3.2 Index navigation*

Having explored in-page verification, we now review alternative algorithmic approaches to verification of a B-tree's structure. For each algorithmic approach, we first focus on verification of a single B-tree and then consider matching multiple B-trees, e.g., a clustered index and its non-clustered indexes. Verification of a B-tree's structure covers the network of pointers among B-tree nodes as well as the ordering of key values among sibling nodes and among parent and child nodes. In database systems using a different variant of B-trees, the following algorithms require adaptation. For example, we assume that leaf nodes as well as interior nodes point to their immediate neighbors by means of page identifiers, with no pointers at the left and right edges of the B-tree. Verification of clustered and non-clustered indexes requires precise one-to-one matches of their records.

The first approach is naïve navigation of the index's structure, in particular of the child pointers and the neighbor pointers. In an ordered B-tree index, a breadth-first traversal is possible but probably not ideal. A depth-first traversal sweeps from the lowest to the highest search key, matching forward and backward pointers as well as key ranges at all B-tree levels. If the B-tree structure is exploited for deep read-ahead for a fast index-order scan, the resulting traversal is really a hybrid of breadth-first and depth-first scans. Of course, a parent node must be verified before its contents may be employed for read-ahead or any other traversal.

The progression through the B-tree's key range is important not only for consistency but also to prevent infinite loops, e.g., among leaf pages if the linkage information is corrupted in an unfortunate way. Massive sequences of duplicate keys in non-unique indexes could foil this safety, but both clustered and non-clustered indexes actually contain only unique entries, if necessary by use of artificial "uniquifier" fields in clustered indexes or by inclusion of the clustered index's search key in non-clustered indexes and their sort order. Such uniqueness is required, for example, for accurate deletion of B-tree entries.

**Performance** of index navigation will be linear with the number of pages allocated for the B-tree, with very modest demands on the buffer pool. Only single-page read operations can be employed if the "next leaf" pointer is used to scan through the leaf pages, whereas appropriate use of the parent nodes enables large, multi-page read operations with the attendant efficiency improvement. This efficiency improvement can be realized only if the B-tree's layout within the database is not fragmented due to bad algorithms during index creation or due to many insertions and page splits without subsequent B-tree defragmentation. In addition to the large read operations, parent nodes also enable multiple asynchronous read operations, which are helpful in a striped storage organization.

**Multiple indexes** for a table, say a clustered index and a few non-clustered indexes, must first be verified for internal consistency and then entries in each non-clustered index must be matched against the clustered index. Obviously, the number of records must be the same in each index; in addition, these records must actually match one-to-one. Moreover, an index's tree structure must be verified before the verification algorithm should attempt a key search in the index.

**Compared to alternative algorithms**, the main advantage of this algorithm is its simplicity and code reuse. Using the same code that ordinary database operations use for index access ensures that the code is tested thoroughly. Given that the main purpose of index verification is certainty that the database is correct, and given that index verification defends against many errors that include software defects in the database management system, the choice of this algorithm is a tradeoff strictly for correctness and dependability. Other advantages include very moderate demands on the buffer pool as well as notification of the error very quickly after the B-tree node containing the error is read.

On the other hand, this algorithm suffers from poor I/O performance including many repeated read operations for each page; thus, its overall performance and scalability are poor unless the entire table fits into the available buffer pool. In other words, the algorithm is still attractive for in-memory databases including caches on mid-tier or client machines, even if a small fraction of the data might have spilled to disk.

**SQL Server** used to depend on a variant of this algorithm for index verification. The original design target in the mid 1980s had been 16-bit machines, so small memory requirements were very important. The algorithm is still in use today, but only after the

current algorithm for matching non-clustered indexes against the clustered index has found the existence of an error but not its precise location, as described later.

**Opportunities** exist to improve the above basic approach should that be desired. For example, multiple indexes can be verified independently and thus concurrently. Depending on relative CPU and I/O bandwidth, this can speed up the tree verification phase by a small factor, assuming that the buffer hit rate is not materially affected by this concurrency. During the matching phase, read-ahead in the non-clustered index provides a small performance advantage. In the clustered index, B-tree searches can employ asynchronous prefetch by pursuing multiple searches concurrently. On a machine with more disk arms than execution threads, this can improve performance of the match phase also by a small factor. Finally, the matching operation between clustered and non-clustered indexes is rather similar to join operations. The simplest algorithm employs essentially index nested loops join; alternative join algorithms include merge join and hash join.

## *3.3 Aggregation of facts*

An entirely different approach is needed in order to support terabyte databases. The following method gathers the same facts and verifies the same consistency constraints but separates the verification from the read operations. In other words, as data pages are read and information is extracted, it is not verified immediately. Instead, facts are extracted and streamed into a matching algorithm, e.g., "a leaf node on disk page 5 points to a successor leaf node on disk page 92." When a matching fact is found, e.g., "a leaf node on disk page 92 points to a predecessor leaf node on disk page 5," verification for these two facts is successful. At the end of the entire matching operation, the B-tree is consistent if and only if all facts have been matched.

While this approach might seem awkward at first, it offers several advantages. The most significant advantage is that B-tree pages are read only once rather than many times as in the index navigation method above. The I/O reduction may exceed two orders of magnitude. Moreover, B-tree pages may be read in any order and disk-order scans are acceptable. Depending on the fragmentation status of the B-tree, large read operations may contribute another order of magnitude to the speed. The usual optimizations apply just as in ordinary query execution, including asynchronous read-ahead, striping, etc.

The crucial component of this approach is in the selection and representation of facts extracted from B-tree pages and matched in the verification step. For the chain of neighbors, the fact "Page x follows page y" is extracted from both pages x and y. For the parent-child relationship, "Parent x points to child y for key range [a, b)" is extracted from parents and children. These matches also verify the level of the two pages (leaf pages are level 0) and permit the appropriate flexibility in matching separator keys in the parent and actual keys in the child. For the root page, one special fact is generated to provide an artificial parent; alternatively, the appropriate catalog entry and its pointer to the B-tree's root might provide this fact.

For example, consider the B-tree in Figure 2 with root page a and two leaves b and c. Shaded areas represent records and their keys; two keys are different (equal) if their shading is different (equal). Eight facts will be extracted from the B-tree nodes in Figure 2:

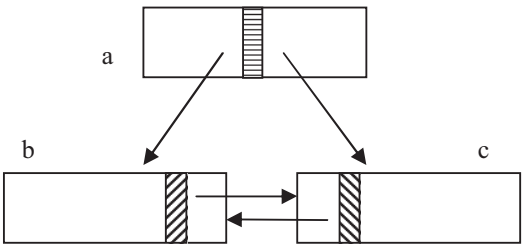



Figure 2. Root and two leaves.

1. From page b, the fact that b is a leaf page, together with its actual key range
2. From page b and its successor pointer, the fact that c follows b
3. From page c, the fact that c is a leaf page, together with is actual key range
4. From page c and its predecessor pointer, the fact that c follows b – this matches with fact 2 above
5. From page a, the fact that b is a leaf page, together with its permissible key range, which is open at the low end – this matches with fact 1 above
6. From page a, the fact that c is a leaf page, together with its permissible key range, which is open at the high end – this matches with fact 3 above
7. From page a, the fact that a is a level-1 node, together with its actual key range (a singleton value)
8. An artificial fact to match fact 7 – this fact could also be derived from the catalogs

With no sibling pointers along the left and right edge of the B-tree, no facts are generated there. The key ranges include information about open and closed ranges, as appropriate. Facts verifying sibling pointers could be augmented with key information, enabling the aggregation to verify that indeed all keys in the predecessor are lower than those in the successor. However, this relationship is already verified transitively via the separator key in the parent. In other words, no additional information or confidence is gained by comparing the two leaf records (shaded diagonally in Figure 2) with each other in addition to comparing each of them with the separator key in the root (shaded horizontally).

**Cousin nodes** in a B-tree with multiple levels are neighboring nodes with no shared parent but instead a shared grandparent or even higher ancestor. Cousin nodes gives rise to what we call the "cousin problem," illustrated in Figure 3. Verification of keys and pointers among cousin nodes does not have an immediate or obvious solution. Nonetheless, two alternative solutions with different performance characteristics are discussed below, one employed today and a novel one that is both simpler and faster.

The essence of this problem as shown in Figure 3 is that the key separating leaves d and e is not in a shared parent node but in a higher ancestor node, in this case in root a. The potential problem is that there is no easy way to verify that all keys in page d are indeed smaller than the separator key in page a and that all keys in page e are indeed larger than the separator key in page a. Correct key relationships between neighbors (b-c and d-e) and between parents and children (a-b, a-c, b-d, c-e) do not guarantee correct key relationships across skipped levels (a-d, a-e). Two alternative solutions for the cousin problem will be discussed shortly.

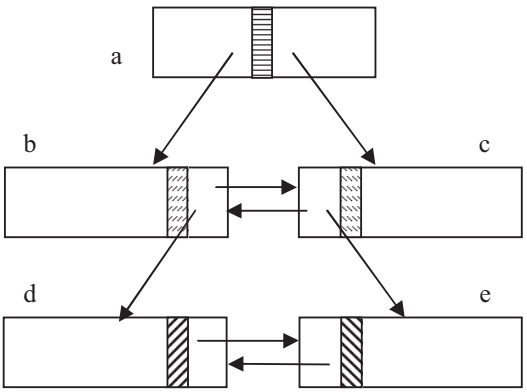



Figure 3. Cousin nodes.

**Performance** of verification by fact aggregation is proportional to the number of facts derived from a B-tree and thus to the number of pointers or relationships in the B-tree. Each node has a parent, and each node (or only each leaf) has a sibling, with exceptions for the root and for nodes along the left and right edge of the B-tree. Thus, the number of facts is about four times the number of nodes or pages in the B-tree.

Extracting these facts from data pages represents a substantial data reduction. For example, if each fact requires 20-40 bytes of space, 4 facts per B-tree node require 80-160 bytes. Thus, for data pages of 8 KB, the data required for verification of page linkages in a B-tree amounts to 1-2% of the data space.

The data volume of the facts also depends on the method employed to group the individual facts derived from individual pages. Perhaps the most obvious method groups facts by type and the pair of page numbers. For example, in Figure 2, facts 2 and 4 are of the sibling type and may be grouped on the pair (b, c). Similarly, facts 1 and 5 are of the parent-child type and may be grouped on the pair (a, b). This method permits very fine partitioning of all facts as it produces multiple distinct fact records from each page.

An alternative method defines each fact as primarily about one page, not about a pair of pages, and groups facts by this primary page only. For example, in a parent-child relationship, the child may be defined as the primary page, such that facts 2 and 4 above are both considered facts about page c. Similarly, the right sibling in a sibling relationship may be defined as the primary page. In Figure 2, facts 3 and 6 would be considered facts about page c. This technique saves memory by creating fewer facts records. For example, when inspecting page c, facts 3 and 4 above generate only a single aggregation record rather than two. The matching logic must require two matching facts for such a combined fact. Given that most pages in a B-tree are both another node's child and another node's right sibling, this technique immediately reduces the data volume for the aggregation operation, e.g., the size of the hash table for hash aggregation.

For the matching step, all algorithms for "group by" and "distinct" queries can be applied, including sort- and hash-based algorithms [G 93], as the essence of matching is to group and combine all detail facts about each page. The complexity of the entire procedure is *O (N log N)*, i.e., the same as sorting the fact data.

**Compared to alternative algorithms**, this approach are that it scales reasonably well yet produces precise error messages. Other advantages include sharing code with query execution, in particular code for disk-order scans, sort- and hash-based aggregation, pipelining, parallelism, and memory management.

**SQL Server,** starting with release 7, verifies the structural integrity of B-trees using a version of this approach with dynamic generation of SQL syntax and using language extensions reserved for such internal queries.

The product also employs some optimizations in the algorithm's implementation, and the information laid out in the explanation of Figure 3 is aggregated using the fewest possible individual fact records. For example, information about both the relationships c-e and d-e in Figure 3 is considered information about node e and aggregated as such.

In combination with the design choice to keep the same number of keys and pointers in each interior node, this optimization also permits solving the cousin problem. Recall that, in a split of an interior node, the new right node retains the low key; in Figure 3, node c contains a copy of the separator key in node a. The cousin problem is solved by aggregating both the low key of node c and the high key of node d into the information about node e, i.e., by ensuring the correct order between nodes c and d. Verification of "second cousins" (also third, etc.) in multi-level B-trees is possible because separators keys are copied during node splits at any level in the B-tree.

If multiple SQL Server indexes are verified in a single pass, their allocation information is aggregated in a single disk-order scan. The facts retained from the allocation information include which page belongs to which B-tree, and verification of a page's information about the B-tree it belongs to is part of the matching process.

**Opportunities** for improvement are many. For example, the sibling information can be augmented with information about space utilization within pages. If two neighboring pages are found to be nearly empty, an opportunity for merging and thus space reclamation exists. In the opposite case, two pages that are nearly full can be split into three pages with more standard space utilization, and two neighboring pages with a large difference in space utilization may benefit from some movement and load balancing.

Facts can be eliminated as soon as all matching facts are found. In a disk-order scan, success in early aggregation indicates a high quality of clustering, whereas poor success in early aggregation indicates a need for defragmentation for faster index-order scans such as range queries. The opportunity is to focus defragmentation on specifically those pages where it can achieve the largest difference for scan performance.

**Fence keys** are a technique that can help with the cousin problem illustrated in Figure 3, even if they were originally motivated by write-optimized B-trees [G 04]. The essential difference to traditional B-tree designs is that page splits not only post a separator key to the parent page but also retain copies of this separator key as high and low "fence keys" in the two post-split sibling pages. Note that separators and thus fence keys can be very short due to prefix and suffix truncation [BU 77]. These fence keys take the role of sibling pointers, replacing the traditional page identifiers with search keys.

Figure 4 illustrates the concept of fence keys, and how they replace the sibling pointers of Figure 3. As before, areas with equal shading indicate equal key values. The focus of the discussion is on the one key value with five copies shaded horizontally. Pages d and e are cousins because they belong to the same B-tree level and share a fence key value but not a parent. In order to navigate from page d to its successor, a search from the

root page is required. However, such navigation steps will rarely be required because the fence keys are lockable resources; thus, key range locking never needs to navigate to a sibling leaf in search of a lockable key value. Large scans, on the other hand, need to employ parent and grandparent nodes in order to achieve deep, many-page read-ahead.

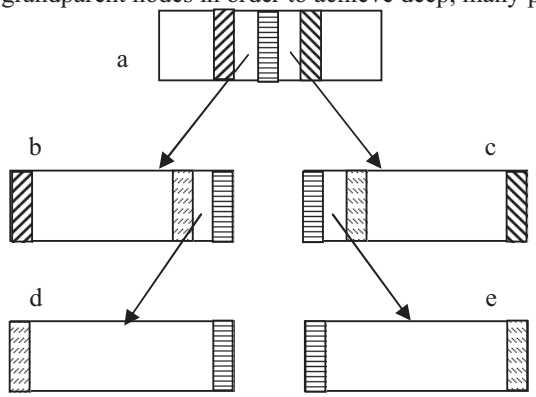



Figure 4. Fence keys.

The important benefit here is that verification is simplified and the cousin problem can readily be solved, including "second cousins", "third cousins", etc. in B-trees with additional levels. In Figure 4, the following four pairs of facts can be derived about the key marked by horizontal shading, each pair derived independently from two pages.

1. From page a, the fact that b is a level-1 page, and its high fence key
2. From page a, the fact that c is a level-1 page, and its low fence key
3. From page b, the fact that b is a level-1 page, and its high fence key – this matches with fact 1 above
4. From page b, the fact that d is a leaf page, and its high fence key
5. From page c, the fact that c is a level-1 page, and its low fence key – this matches with fact 2 above
6. From page c, the fact that e is a leaf page, and its low fence key
7. From page d, the fact that d is a leaf page, and its high fence key – this matches with fact 4 above
8. From page e, the fact that e is a leaf page, and its low fence key – this matches with fact 6 above

The separator key from root a is replicated along the seam of neighboring nodes all the way to the leaf level. Equality and consistency are checked along the entire seam and, by transitivity, across the seam. Thus, fence keys also solve the problem of second and third cousins etc. in B-trees with additional levels.

Beyond solving the cousin problem, i.e., enabling verification of neighbors that do not share a parent, the list above requires only equality comparisons between page identifiers, keys, and level information derived from different pages. In contrast, the verification in Figure 3 requires "less than" comparisons, as does the solution employed in SQL Server.

Complete reliance on equality comparisons enables the next and most efficient algorithmic approach to B-tree verification, discussed shortly.

Finally, the list above indicates that only a single record format and aggregation operation is required to capture the facts required for B-tree verification. This record format includes page identifier, B-tree level, a key value, and an indication whether the key value is a low fence key or a high fence key. Thus, fence keys simplify B-tree verification not only on a conceptual level but also on this detailed implementation level.

Fence keys also extend the local online verification techniques described by Küspert [K 85]. In traditional systems, neighbor pointers can be verified during a root-to-leaf navigation only for siblings but not for cousins, because the verification of a leaf's cousin pointer would require an I/O operation to fetch the cousin's parent node (also its grandparent node for a second cousin, etc.). Thus, Küspert's technique cannot verify all correctness constraints in a B-tree, no matter how many search operations perform verification. Fence keys, on the other hand, are equal along entire B-tree seams, from leaf level to the ancestor node where the key value serves as separator key. A fence key value can be exploited for online verification at each level in a B-tree, and an ordinary root-to-leaf B-tree descent during query and update processing can verify not only siblings with a shared parent but also cousins, second cousins, etc. Two search operations for keys in neighboring leaves verify all B-tree constraints, even for cousin leaves, and search operations touching all leaf nodes verify all correctness constraints in the entire B-tree.

## *3.4 Bit vector filtering*

In addition to these advantages of fence keys over traditional neighbor pointers, fence keys and the complete reliance on equality comparisons during B-tree verification enable a further simplification that reduces the complexity of the entire operation to a single scan. The complexity of this new method has no logarithmic component. Moreover, memory consumption is very moderate and can be fixed independently of the data size.

The basic idea of this approach is quite simple, and readily applies to the pairs of facts derived from Figure 4. Instead of precisely aggregating facts, this algorithmic approach employs a bitmap to verify that facts derived from the on-disk data indeed match up correctly. For each B-tree node and each child pointer in the on-disk data structures, the combination of index identifier, page identifier, B-tree level, low and high fence keys is hashed to a bit position and the bit value currently in that position is reversed. At the end, the entire bitmap should be back in its original state. If it is not, the scanned on-disk data contains an error or corruption. The precise type and location of that error are not known, however. In that sense, it is the first algorithm in this survey that truly only verifies the correctness rather than pinpoints the error or corruption.

There are multiple assumptions underlying this approach. First, on-disk corruptions must be rare; otherwise, this approach is not promising. Second, double-errors also must be rare, and the probability of errors hidden due to hash collisions of double errors must be negligible. Third, it must be more important to speed up those verification runs that find nothing amiss than to ensure that corruptions are pinpointed in the first attempt. Fourth, facts must match in even numbers, typically pairs. Fifth, hashing key values to bit positions supports only equality predicates, not range predicates.

If an on-disk corruption is found to exist, a second pass must find it before it can be repaired. This pass needs to read the same data again, but it only needs to investigate those facts that map to bit positions in the bitmap indicating a failed match. This investigation may employ either of the prior algorithms, i.e., either index navigation starting from each fact that maps to a non-matching bit position or an aggregation of all facts that match to such bit positions.

**Compared to other algorithms**, the advantage of using bitmaps instead of aggregation is higher performance with less memory, because neither sorting nor a hash table is required. Moreover, the entire process completes in a single scan of the data pages. The main disadvantages are twofold. First, it only works for equality predicates, not range predicates. Thus, it can be applied to B-trees with fence keys but not to traditional B-trees that require "less than" comparisons during verification. Second, the method does not produce precise error information. If an error is found to exist, a second pass over the data must find the precise error and its location. Minor disadvantages are that facts have to match up in even numbers and that there is a theoretical possibility, although with very low probability, of errors hidden due to hash collisions.

**SQL Server** employs bitmaps for matching non-clustered indexes against clustered indexes. Bit vector filtering is faster with much less memory than aggregation for the same problem. A typical size for the bit vector filter is 32 KB or ¼ M bits. If an error is found, index navigation is used to isolate the fault. Bit vector filtering is not used for verification of B-tree structures, because SQL Server does not employ fence keys in B-tree nodes and leaves. It is not employed for verification of sibling pointers or child pointers because those facts are verified together with the appropriate key relationships.

**Opportunities** exist for further improvement. For example, a bitmap might not be necessary. Instead, a single integer checksum might suffice. On the other hand, the improvement in performance and scalability due to a checksum rather than a bitmap might not be worth the increased probability of failure to detect a corruption and the increased search effort if an error is found to exist.

Multiple bitmaps permit isolating the type of error, e.g., parent-child relationships versus sibling relationships. Separate bitmaps also permit isolating the location of the corruption, e.g., a bitmap for each B-tree.

The most important opportunities, however, might be in the algorithm's usage rather than in the algorithm itself. Because of its single-scan behavior and its low cost, the algorithm can be integrated into backup and restore operations, database mirroring, and similar database utilities that scan or copy entire tables or even entire databases. Verification of back-up data, both while creating a backup and while restoring it, increase users' confidence in the backup and thus the value of the backup data. Today's standard method is to run a verification step prior to a backup. Integration of verification and backup into a single scan cuts the entire operation's elapsed time in half.

## *3.5 Query evaluation*

The final algorithmic approach relies even more on the database system's query processing engine than the prior ones. The basic idea in this approach is to formulate a query to find violations and to report the query output as database corruptions in need of repair.

This approach is also employed in many data definition statements, e.g., when a new integrity constraint is explicitly defined by a database administrator.

This approach to database verification is most appropriate when verification of the on-disk data structures requires traditional query operations, e.g., when comparing a materialized and indexed view with its base tables, when deriving a computed column using arithmetic and functions, and when extracting index keys from user-defined types.

**Compared to the other algorithmic approaches** for index verification, this algorithmic approach to database verification can evaluate complex expressions. It reuses existing code including query optimization and parallel query execution. Its main disadvantage is its dependency on a large volume of complex code. In addition, unless the query processing implementation supports optimization and execution of multiple queries together, multiple scans of the same data might be needed. Finally, some query optimizers suffer from somewhat unpredictable plan choices and thus unpredictable performance, which may be considered particularly undesirable for utilities that take entire tables or even an entire database offline.

**SQL Server** relies on this approach for "not null" constraints and for "check" constraints, exploiting indexes if possible. It also employs this approach for comparing materialized views with query results, and for indexes on user-defined types. The final verification step uses query logic, not bitmaps.

**Opportunities** for improvement remain, of course. For example, the same code should be employed during verification of data definition statements and during verification on on-disk database contents, even if these two functions have traditionally been implemented by different teams within a large database development team. A second opportunity is that verification of an entire database and all its structures is a complete, well-structured workload that can be planned as a whole, including temporary materialized and indexed views that can be shared for multiple verification steps.

# 4 Performance evaluation

In order to assess performance and scalability of each algorithm, we ran a number of tests using the algorithms implemented in SQL Server 2005. Figure 6 shows the performance of various index verification tests relative to the scan performance. The data are the "orders" and "line items" tables in a TPC-H database with 10 GB of data plus indexes. The test hardware is a workstation with a 3 GHz dual-core Intel CPU, 2 GB RAM, and a single SATA disk drive. Each test starts with a cold I/O buffer. The tables in this test, orders and line items, each had a clustered index on a date column plus a non-clustered index on the primary key and on each foreign key. Each test includes a variant with parallelism disable and enabled. Each test is shown as a set of 4 bars.

The 1st set of bars in Figure 6 (marked "clustered index scan") shows the performance of "select count (*)" queries, which measure primarily the scan performance of the I/O hardware. Using query optimization hints, we forced a disk-order scan of the entire clustered index. By definition, each bar shows 100%, because we use this query to show the relative performance of more complex operations. Proportional to the data volumes, the actual execution times for operations against the "line item" table are about four times longer than those against the "orders" table.

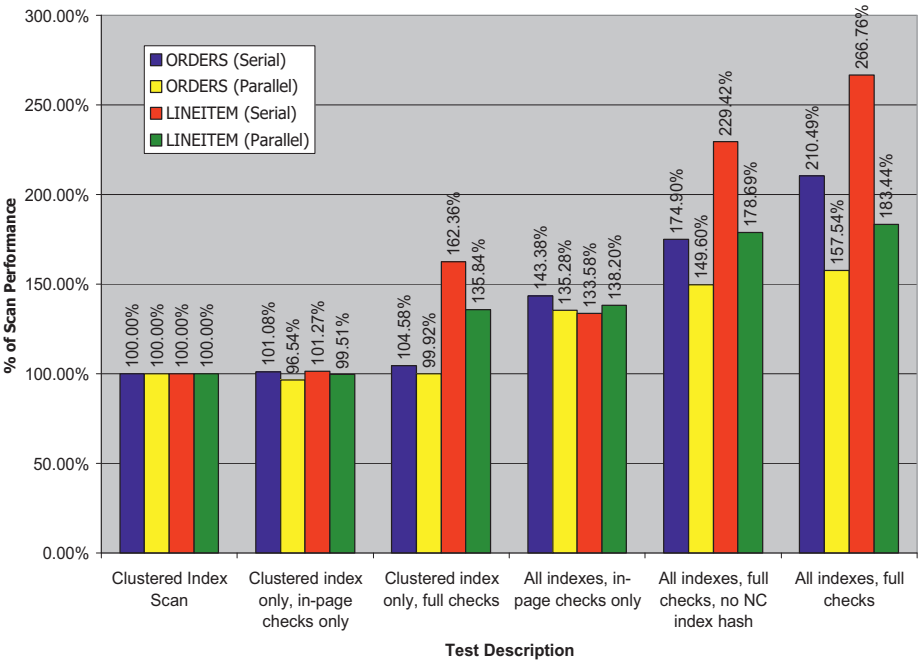



Figure 6. Algorithm performance.

The 2nd set of bars in Figure 6 (marked "clustered index only, in-page checks only") illustrates the overhead of in-page consistency verification. If B-tree verification is limited to in-page checks, the CPU can readily keep up with the I/O hardware. Thus, the total execution time for in-page-only verification is practically equal to that of the "select count (*)" query in all 4 cases. Thus, the in-page checks outlined in Section 3.1 can be achieved with acceptable effort. This argues for performing such checks not only during offline verification but also online during normal query and update processing, as already proposed by Küspert [K 85] and Mohan [M 95].

The 3rd set of bars in Figure 6 (marked "clustered index only, full checks") illustrates the cost of full structural verification of a B-tree. For verification of all aspects of the clustered index B-tree fact aggregation as described in Section 3.3 is employed and adds to the execution times. Due to efficient fact extraction and aggregation, this operation can keep up with a disk-order scan, at least for the "orders" table.

The performance for the "orders" table is almost equal to the "select count (*)" query, whereas the performance for the "line items" table is not. The reason is the difference in the table sizes. While the facts extracted from the "orders" table can be sorted in the server's allocated memory, the "line items" table is four times larger and sorting the extracted facts requires I/O to temporary run files. Without sufficient memory, even moderate sizes of tables and indexes incur the penalty of I/O to temporary files.

As this example illustrates, the performance and scalability of sort operations dominates the complexity, performance, and scalability of B-tree verification by fact aggregation. Parallelism and multiple CPU cores speed up the verification task, because a single CPU core cannot perform in-page verification, fact extraction, and sorting with the same bandwidth as the disk drive. However, parallelism adds its own overhead, too, and using both CPUs fails to cut elapsed times in half.

The 4th set of bars in Figure 6 (marked "all indexes, in-page checks only") illustrates the additional data size due to non-clustered indexes. For in-page verification of both clustered and non-clustered indexes, execution times rise beyond those of the "select count (*)" query and the clustered index in-page verification by a fairly uniform amount. In the experimental database, the size of the non-clustered indexes relative to the clustered index is about 40% for the "orders" table and about 35% for the "line items" table.

The 5th set of bars in Figure 6 (marked "all indexes, full checks, no NC index hash") illustrates the cost of B-tree verification for multiple indexes in a single operation. All a table's B-trees are checked completely but indexes are not matched against one another. Thus, this experiment shows the same task as the 3rd set of bars but for an increased data volume. The performance difference between the 4th and 5th sets of bars is quite similar to the performance difference between the 2nd and 3rd sets of bars. For example, the difference between 162% and 229% is 41% (229/162=1.41), appropriate for the data volume.

The results discussed so far enable a few conclusions. First, verification of a B-tree structure using fact aggregation is many times faster than a traditional method based on index navigation. Second, due to its foundations in query processing and efficient sorting, verification using fact aggregation scales very well. Third, verification of a B-tree's structure is still the most expensive aspect of B-tree verification.

The 6th set of bars in Figure 6 (marked "all indexes, full checks") illustrates the cost of bitmap processing. As readily apparent in a comparison to the 5th set of bars, matching records in clustered indexes with records in non-clustered indexes using bitmaps adds only about 12-20% to the execution times; much less if two CPU cores are employed. This is a low cost for the added value; the best alternative a join for each non-clustered index. It is important to realize that the bitmap operation requires only a small amount of memory, and that this amount is virtually independent of the data size.

This low additional CPU load due to bitmap processing (in this case, for cross-index verification) also points to further possible gains. Both CPU load and overall verification performance would improve if cross-page verification within each B-tree would use bitmaps instead of sorting and aggregation of fact records, i.e., would use the new B-tree verification algorithm proposed in this paper. Instead of adding 35-40% to the cost of the data scan, cross-page verification would add merely about 1-2%. It would add less than the 12-20% observed above for bitmap processing because there are merely 2 facts per B-tree node, not tens or hundreds of keys per leaf. Moreover, verification efforts for larger databases and indexes would increase linearly rather than with the *O (N log N)* complexity of sorting, and it would avoid entirely any I/O to temporary run files.

To summarize the performance observations, B-tree verification can be performed at I/O speeds, including complete verification of all links and relationships within each B-tree index and of the relationships of each non-clustered index and the clustered index. The introduction of fact extraction and aggregation has been a tremendous improvement over the prior verification algorithm based on index navigation. The cost of bitmap proc-

essing (as observed for cross-index verification) indicates that reliance on bitmaps also for cross-page verification could further improve performance and scalability. It could enable streaming verification not only during scans but also during backup and restore operations, during database mirroring, and any other database utilities that essentially perform copy operations of databases or tables.

# 5 Summary and conclusions

In summary, consistency verification within and between B-tree indexes is as important as solid concurrency control and dependable backup. Production users invoke consistency verification weekly or even daily in order to guard against data loss due to problems in their hardware or software, from SAN management software and networking to device drivers and the database management system itself. Moreover, verification of on-disk data structures is invoked even more frequently during development and testing of database software [WY 95], where high performance and good scalability directly contribute to shortening the testing effort and thus the development schedule.

Intra-page verification is fairly straightforward, although further opportunities for innovation have been identified. For inter-page verification, there is a surprising variety of algorithmic approaches. For pragmatic reasons that each seemed decisive at their time, the developers of SQL Server have, over the course of more than 20 years, taken different approaches to different aspects of the overall problem. Other database products have gone through similar histories. In this paper, we have reviewed these approaches, compared their strengths and weaknesses, and evaluated their performance inasmuch as possible with a product that cannot readily be modified for such investigations.

Based on these improvements, database verification proceeds at nearly the speed of scans. Bandwidth above a gigabyte per minute can be observed on ordinary workstations using fact aggregation within B-trees and bit vector filtering across indexes. A verification task that takes minutes today used to take hours or even days using index navigation.

Among the new insights of this research, the most important is the advantage of fence keys for efficient B-tree verification. In addition to their benefits explored in earlier work [G 04], they permit complete and correct verification of all relationships among B-tree nodes and their keys. In particular, as illustrated in Figure 4, they easily solve the cousin problem illustrated in Figure 3, including second cousins, third cousins, etc. Moreover, due to the avoidance of all “less than” comparisons in cross-page checks, fence keys permit replacing the current algorithm based on sorting and aggregation with an algorithm based entirely on bit vector filtering. The resulting algorithm reduces both processing effort and memory requirements for efficient index verification in very large databases. Thus, this improved algorithm promises to keep up with the highly tuned I/O mechanisms used in backup and restore, a capability database owners demand in order to avoid a separate verification step for their backup media.

Many further extensions and improvements of the algorithms described here are possible. Obvious extensions include generalizations for partitioned and shared-nothing architectures, automatic repair of corrupted on-disk data structures, coverage of data duplication and replication including database replication and mirroring, and coverage of other storage formats in addition to traditional B-tree indexes. Such other formats range from

unordered heap structures and storage structures for large binary objects to bitmap indexes and column stores. Further possibilities include verification of complex values including XML objects and user-defined types, online verification during active transaction processing, and incremental verification exploiting short periods of low user activity.

# 6 Acknowledgements

This report would not have been possible without the contributions by the inventors of Microsoft's patents on B-tree and database verification. Mike Zwilling provided helpful suggestions on the presentation of the material. Ahmed Ezzat and Gottfried Vossen suggested excellent improvements in the presentation of our work.

# 7 References

[BD 83] Dina Bitton, David J. DeWitt: Duplicate Record Elimination in Large Data Files. ACM Trans. Database Syst. 8(2): 255-265 (1983).

[BGS 08] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau: An analysis of data corruption in the storage stack. FAST 2008: 223-238.

[BM 72] Rudolf Bayer, Edward M. McCreight: Organization and Maintenance of Large Ordered Indices. Acta Inf. 1: 173-189 (1972).

[BU 77] Rudolf Bayer, Karl Unterauer: Prefix B-Trees. ACM Trans. Database Syst. 2(1): 11-26 (1977).

[C 79] Douglas Comer: The Ubiquitous B-Tree. ACM Comput. Surv. 11(2): 121-137 (1979).

[G 93] Goetz Graefe: Query Evaluation Techniques for Large Databases. ACM Comput. Surv. 25(2): 73-170 (1993).

[G 04] Goetz Graefe: Write-Optimized B-Trees. VLDB Conf. 2004: 672-683.

[GR 93] Jim Gray, Andreas Reuter: Transaction Processing: Concepts and Techniques. Morgan Kaufmann 1993.

[HDB 08] James L. Hafner, Veera Deenadhayalan, Wendy Belluomini, Krishnakumar Rao: Undetected disk errors in RAID arrays. IBM Journal of Research and Development 52(4-5): 413-426 (2008).

[BM 70] Rudolf Bayer, Edward M. McCreight: Organization and Maintenance of Large Ordered Indexes. SIGFIDET Workshop 1970: 107-141.

[K 85] Klaus Küspert: Fehlererkennung und Fehlerbehandlung in Speicherungsstrukturen von Datenbanksystemen. Informatik Fachberichte 99. Springer 1985.

[LY 81] Philip L. Lehman, S. Bing Yao: Efficient locking for concurrent operations on B-trees. ACM Trans. Database Syst. 6(4): 650-670 (1981).

[M 95] C. Mohan: Disk Read-Write Optimizations and Data Integrity in Transaction Systems Using Write-Ahead Logging. ICDE 1995: 324-331.

[MH 94] C. Mohan, Donald J. Haderle: Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking. EDBT Conf. 1994: 131-144.

[MHL 92] C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Trans. Database Syst. 17(1): 94-162 (1992).

[PGK 88] David A. Patterson, Garth A. Gibson, Randy H. Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID). ACM SIGMOD Conf. 1988: 109-116.

[PP 03] Meikel Pöss, Dmitry Potapov: Data Compression in Oracle. VLDB Conf. 2003: 937-947.

[RSS 96] Kenneth A. Ross, Divesh Srivastava, S. Sudarshan: Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. SIGMOD Conf. 1996: 447-458.

[S 75] Michael Stonebraker: Implementation of Integrity Constraints and Views by Query Modification. ACM SIGMOD Conf. 1975: 65-78.

[W 81] Christian Weber: Ein Verfahren zur schnellen Konsistenzprüfung von Datenbanken. Angewandte Informatik 23(11): 497-501 (1981).

[WY 95] D. Wildfogel, Ramana Yerneni: Efficient Testing of High Performance Transaction Processing Systems. VLDB 1997: 595-598.

# A Bayesian Approach to Estimating the Selectivity of Conjunctive Predicates

Max Heimel, Volker Markl, Keshava Murthy
IBM Germany, TU Berlin, IBM San Jose
mheimel@de.ibm.com, marklv@cs.tu-berlin.de, rkeshav@us.ibm.com

**Abstract:** Cost-based optimizers in relational databases make use of data statistics to estimate intermediate result cardinalities. Those cardinalities are needed to estimate access plan costs in order to choose the cheapest plan for executing a query. Since statistics are usually collected on single attributes only, the optimizer can not directly estimate result cardinalities of conjunctive predicates over multiple attributes. To avoid having to fall back to assuming statistical independence, modern relational database systems offer the possibility to additionally collect joint statistics over multiple attributes. These statistics allow a direct cardinality estimate for conjunctive predicates.

A widely used approach is collecting the number of distinct value combinations as a joint statistic. This can be used for a uniformity based estimate, which assumes each value combination to occur equally often. Although this estimate is likely an improvement, it is still inaccurate, since "real world" data is unlikely to be uniform.

This paper proposes a new approach of estimating the result cardinality of conjunctive predicates over multiple attributes of a relation. The proposed method combines knowledge from single-column histograms using a conditional probability based "uniform correlation" approach. Initial evaluation shows that this method yields better results for estimating predicates on highly correlated attributes than the classic uniformity based approach.

## 1 Introduction

Modern relational database systems make use of cost-based optimizers to generate the most efficient access plan for a given SQL query. A cost-based optimizer generates a set of possible plans for a given query and estimates the execution cost for each of those plans. The plan with the lowest estimated cost is chosen for execution.

In order to estimate plan costs, the optimizer has to know the cardinalities of intermediate operations contained in the plan. The cost of a relational operation depends on how large the input relations are, therefore these cardinalities are necessary to give an accurate cost estimate. Cost-based optimizers usually use data statistics - e.g. data histograms - to estimate those cardinalities.

Data statistics are usually collected for single attributes only. This allows the optimizer to estimate how many tuples of a relation fulfill a condition for a single attribute. Since the available statistics normally do not contain information about dependency patterns between attributes of a relation, the optimizer has to assume statistical independence between filtered attributes when multiple conjunctive attribute conditions have to be considered. In

the case of a filter that queries multiple attributes that share some kind of dependency, the independence assumption can lead to drastically underestimated result cardinalities. Especially in the case of complex SQL queries, those underestimated cardinalities will propagate through the cost estimation - effectively resulting in drastically underestimated plan costs. This error might bias the optimizer into choosing a non-optimal plan for execution.

In order to avoid incorrect independence assumptions during query optimization, modern cost-based optimizers make use of multi-dimensional statistics. Since those statistics are joint over multiple attributes, they inherently include information about potential dependency patterns. Since multi-dimensional histograms are expensive to build and maintain, most database systems use simpler methods. This usually involves collecting the number of distinct combined values of a set of attributes and assuming each of those distinct combinations is equally likely to occur.

Although this simple uniformity assumption leads to better estimations, it also introduces new errors. In reality, the distinct value combinations will almost never occur equally likely. Since joint data statistics are currently not combined with available single-attribute distribution knowledge, the optimizer is basically "giving away" available information on how the combined attributes are distributed. We tackle this problem by proposing a novel approach to achieve better estimates by including available single-predicate knowledge into the cardinality estimate.

## 2 Related Work

The concept of cost-based optimizers in relational database systems was introduced in system R by IBM. [SAC+79] describes the optimizer of system R and introduces concepts such as selectivity estimation and the independence assumption for multiple attributes. [PSC84] further investigates the selectivity estimation methods presented by system R. Although system R is already 30 years old, the introduced concepts are still valid today. A more modern look at query optimization can be found in [Cha98], which summarizes the major research topics in this field during the last 30 years.

While those three papers give a more high-level overview about the concepts of relational query optimization, there are also several papers dealing with actual methods of performing estimations. [Ioa03] gives an overview over previous and current work in the field of selectivity estimation using single-attribute data histograms. For constructing multi-dimensional histograms, [MPS99] compares multiple algorithms for partitioning a two-dimensional histogram into buckets and shows that finding the optimal partitioning is essentially NP-hard. Due to this, practical multi-dimensional histogram techniques are using either heuristics or query feedback to perform the partitioning. Examples of practical multi-dimensional histograms include [TGIK02] and [SHM+06]. An overview comparing different types of multi-dimensional histograms can be found in [PI97]. [GTK01] presents a slightly different approach for the multi-dimensional estimation, which uses probabilistic models (e.g. Bayesian Networks) to compute a selectivity estimate.

[MHK+07] shows how to improve estimation quality by using the maximum entropy principle to consistently combine estimates from available multi-dimensional statistics. In contrast to this, our approach introduces a novel and improved way of computing a direct cardinality estimate from one statistics set. These improved estimates could then e.g. be used as input for a maximum entropy based estimator, which should effectively improve cardinality estimates even in cases where no fitting set of statistics is available.

## 3 Problem & Terminology

A conjunctive filter $\theta$ consisting of $n$ single predicates is defined as:

$$\theta = \bigwedge_{i=1}^{n} \theta_i \tag{1}$$

For this paper, we assume that each single predicate $\theta_i$ is an equality predicate of the form $a_i = v_i$, which can be evaluated as:

$$\theta_i(t) = \begin{cases} true, & \text{if } t(a_i) = v_i \\ false, & else \end{cases} \tag{2}$$

Hereby $t$ is a tuple of relation $R$, $a_i$ denotes an attribute name of relation $R$ and $v_i$ is a constant value. The expression $t(a_i)$ denotes the value of attribute $a_i$ in tuple $t$. We will assume that all attributes are from the same relation $R$ and that no attribute is queried twice by $\theta$. This constraint is needed, since a conjunct of two different filters on the same attribute is unsatisfiable and would lead to incorrect estimations.

When no multi-dimensional statistics are available, the optimizer will estimate the cardinality of a conjunctive filter $\theta$ by assuming statistical independence between the single attributes. Using this, the cardinality of $\theta$ is estimated as:

$$|\sigma_\theta(R)| \approx \frac{1}{|R|^{n-1}} \prod_{i=1}^{n} |\sigma_{\theta_i}(R)| \tag{3}$$

Hereby $\sigma_\theta$ denotes the relational selection, which returns the set of tuples from relation $R$ that fulfill filter $\theta$.

Let us consider a relation describing cars as tuples with attributes *make* and *model*. From data histograms, the optimizer knows that $500$ cars are manufactured by "*Opel*" and that $100$ cars are of model "*Astra*". Assuming $10.000$ tuples within the database, the independence assumption tells us there are $5$ cars of model "*Astra*" produced by "*Opel*". This is obviously wrong, since only "*Opel*" produces the "*Astra*": the strong correlation between attributes *make* and *model* leads to an underestimated result cardinality.

One way to avoid underestimated cardinalites due to incorrect independence assumptions is by applying a uniformity based estimation. Hereby, the optimizer assumes that all existing value combinations occur equally likely:

$$|\sigma_\theta(R)| \approx |R| \frac{1}{\vartheta_\theta} \tag{4}$$

In this equation $\vartheta_\theta$ denotes the number of distinct value combinations in the attribute set queried by filter $\theta$. Using the uniformity based approach, let us reconsider the car example. If we assume there are $125$ different value combinations, a uniformity approach would estimate the cardinality for each combination as $80$ tuples. This estimate is obviously better than the independence based estimate of $5$ tuples.

Although it generally leads to better estimates - especially in case of incorrect independence assumptions - the uniformity approach introduces a new estimation error source. If, for example, we choose to find all cars of model "*F430*" that were produced by "*Ferrari*", the uniformity approach would still result in an estimate of $80$ tuples. This is presumably incorrect, since cars by Ferrari are likely to occur much less frequently than cars by Opel. The problem of the uniformity approach is that each combination is treated equally - although they might occur with different frequencies. By including single predicate knowledge from data statistics into the estimate we should be able to improve this. For example: if we know that *Opel* occurs much more often than *Ferrari* within the relation, we should be able to use this knowledge to adjust estimates accordingly.

## 4 Solution

The relational selection operator can be expressed in form of a probabilistic experiment: We take a random tuple out of relation $R$ and check whether it fulfills filter $\theta$. The probability that the random tuple fulfills the filter condition is:

$$P\left(\theta\left(t\right)\right) = \frac{|\sigma_\theta(R)|}{|R|} \tag{5}$$

Through rearranging equation 5, we are able to express the cardinality of filter $\theta$ as:

$$|\sigma_\theta\left(R\right)| = P\left(\theta\left(t\right)\right)|R| \tag{6}$$

We are thus able to give a cardinality estimate through estimating the filter probability. In the context of relational operations, the "probability" of a filter is called *selectivity*. In short: the selectivity tells us the fraction of tuples in a relation, that fulfil the filter.

Using the probability based approach, we are able to incorporate single-attribute knowledge into the selectivity estimate. We can do this by rewriting the joint probabilty through using *conditional probabilities*. The conditional probability $P(A|B)$ denotes the probability that event $A$ will happen under the condition that event $B$ has already occured. The joint probability is then the product of $P(A|B)$ and $P(B)$.

If we translate the definition of conditional probabilites to estimating the selectivity for a conjunctive pair of predicates $\theta_1$ and $\theta_2$ we get:

$$\begin{aligned} P\left(\theta_1\left(t\right) \wedge \theta_2\left(t\right)\right) &= P\left(\theta_2\left(t\right)|\theta_1\left(t\right)\right) P(\theta_1\left(t\right)) \\ &= P\left(\theta_1\left(t\right)|\theta_2\left(t\right)\right) P(\theta_2\left(t\right)) \end{aligned} \tag{7}$$

From now on, we will refer to the conditional probabilites in equation 7 as *conditional selectivities*. Since the two single predicate selectivities $P(\theta_1\left(t\right))$ and $P(\theta_2\left(t\right))$ can be

retrieved from data statistics we are able to estimate the joint selectivity through estimating these conditional selectivities. A very simple way of doing this, is to assume a *uniform correlation* between the two queried attributes. This means we assume the conditional selectivites to be constant over the complete data range:

$$\begin{aligned} P\left(\theta_1\left(t\right) \wedge \theta_2\left(t\right)\right) &\approx \gamma_{\theta_1} P(\theta_1\left(t\right)) \\ &\approx \gamma_{\theta_2} P(\theta_2\left(t\right)) \end{aligned} \tag{8}$$

Equation 7 shows two alternatives for estimating the joint selectivity: one for each part of the filter. The approximation introduced in 8 results in estimation errors of different size for the two alternatives. Since we can't really predict which of the two ways results in the smaller error, we use the average of both alternatives as the final estimate:

$$P\left(\theta_1\left(t\right) \wedge \theta_2\left(t\right)\right) \approx \frac{\gamma_{\theta_1} P(\theta_1\left(t\right)) + \gamma_{\theta_2} P(\theta_2\left(t\right))}{2} \tag{9}$$

The conditional selectivity constants $\gamma_{\theta_1}$ and $\gamma_{\theta_2}$ tell us how many values of the other attribute are - in general - paired with one attribute value. We are able to uniformly estimate those constants as:

$$\gamma_{\theta_i} = \frac{\vartheta_{\theta_i}}{\vartheta_\theta} \tag{10}$$

Hereby $\vartheta_{\theta_i}$ denotes the number of distinct values in the attribute queried by predicate $\theta_i$. By combining equations 6, 9 and 10 we get the final conditional probability based cardinality estimation as:

$$\left|\sigma_\theta\left(R\right)\right| \approx \frac{|R|}{2}\left(\frac{\vartheta_{\theta_1}}{\vartheta_\theta} P\left(\theta_1\left(t\right)\right) + \frac{\vartheta_{\theta_2}}{\vartheta_\theta} P\left(\theta_2\left(t\right)\right)\right) \tag{11}$$

Using the same approach as for predicate pairs, we can generalize formula 11 to conjuncts of $n$ predicates. The resulting estimation formula is:

$$\left|\sigma_{\bigwedge_{i=1}^{n}\theta_i}\left(R\right)\right| \approx \frac{|R|}{n}\sum_{i=1}^{n}\frac{\vartheta_{\theta_i}}{\vartheta_\theta} P\left(\theta_i(t)\right) \tag{12}$$

Let us reconsider the car example from chapter 3 using the conditional selectivity based approach. We will assume there are in total 115 distinct models and 25 distinct makes in the database, this allows us to estimate the two conditional selectivity constants for *make* and *model* as:

$$\gamma_{make} = \frac{25}{125} = 0.2 \qquad \gamma_{model} = \frac{115}{125} = 0.92$$

If we assume there are 15 cars of make *Ferrari* and two cars of model *F430*, we are able to use equation 11 for the cardinality estimation. Equation 11 gives us 96 *Astra* cars from *Opel* and 2.42 *F430* cars from *Ferrari*. Those estimates are nearly perfect (we would expect 100 and 2) and show the potential of the conditional selectivity method compared to the uniformity approach.

## 5 Evaluation

For evaluation, we want to identify the expected general behaviour of the conditional selectivity based estimator compared to the two classic ones (independence and uniformity). The quality of an estimator is dependent on how correct the underlying assumption is for a given relation. For the classic estimators we can give rather accurate predictions:

- The independence assumption based estimator will obviously result in very bad estimations for distributions with high correlation between the attributes.
- In case of the uniformity approach, the estimation error is mainly influenced by the variance of the frequencies within the distribution. A highly varianced frequency distribution should thus result in high errors when using uniformity based estimation.

Evaluating the conditional selectivity method is a bit more complicated, since the used assumption of "uniform correlation" is not as tangible as the other assumptions. However, we can give some general thoughts on the expected behaviour. We expect the conditional selectivity method to work best in case of highly correlated distributions. In these cases, the attributes are very uniformly correlated - in the sense that each value of an attribute has almost exactly one partner in the other attribute(s). For weakly correlated distributions we expect the conditional selectivity method to behave nearly like the classic independence assumption based estimator, since the conditional selectivity estimator approximates the independence estimator in those cases[1].

Figure 1 shows how the estimation error of the three possible estimators (independence, uniformity, conditional selectivity) develops with regard to the correlation factor. For creating the figure, we implemented the conditional selectivity approach into the query optimizer of IBM Informix Dynamic Server and measured the total error for simple select statements on a relation consisting of two attributes. The relation was filled with artificial numerical data with specified variance and correlation, having a skewed frequency distribution to simulate "real-world" data. For figure 1 we created multiple data sets with a

[1] In lowly correlated distributions, the single attributes are nearly statistically independent. This means the number of distinct attribute value combinations approaches the product of the number of distinct attribute values:

$$\vartheta_\theta \approx \prod_j \vartheta_{\theta_j} \tag{13}$$

If we insert equation 13 into the conditional selectivity estimation formula 12 we get:

$$\left|\sigma_{\bigwedge_{i=1}^{n}\theta_i}(R)\right| \approx \frac{|R|}{n}\sum_{i=1}^{n} P(\theta_i(t)) \prod_{j\neq i}^{n} \frac{1}{\vartheta_{\theta_j}} \tag{14}$$

We can now apply a simple uniformity based approximation:

$$\left|\sigma_{\bigwedge_{i=1}^{n}\theta_i}(R)\right| \approx \frac{|R|}{n}\sum_{i=1}^{n} P(\theta_i(t)) \prod_{j\neq i}^{n} P(\theta_j(t)) \approx |R| \prod_{i=1}^{n} P(\theta_i(t)) \tag{15}$$

Comparing the right-hand side of equation 15 to the definition of independence assumption based estimation in 3 shows that they are equivalent.

constant variance and increasing correlation factors[2].

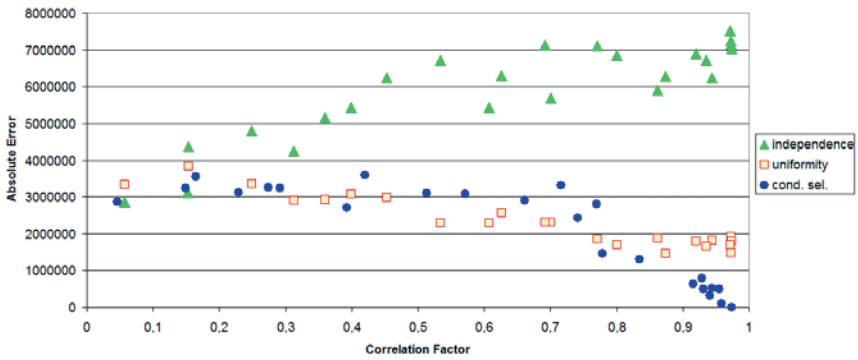



Figure 1: Theoretical comparison of estimators

There are three notable conclusions from figure 1:

1. The independence assumption leads to very high errors in case of high (and medium) correlation factors. The other two estimators lead to drastically better results!

2. The uniformity based estimation leads to a rather constant estimation error, which is not surprising since all data sets we used had roughly the same frequency variance.

3. The conditional selectivity based estimation seems to create roughly the same estimation errors as the uniformity based estimation. However - as we expected - the error decreases drastically for very highly correlated attributes.

## 6 Experiments

In order to confirm the statements from the evaluation chapter, this section presents experiments conducted using the conditional selectivity method. We compare the cardinality estimation quality of the conditional selectivity and the uniformity approach, using the implementation in IBM Informix Dynamic Server. The experiments were performed on attribute pairs of existing "real-world" customer data. Each attribute pair was queried multiple times using simple conjunctive select queries. To compare the estimation quality, each workload was run twice: once using the classic uniformity estimator and once using the conditional selectivity estimator. Table 1 shows statistical properties of the used groups and whether the estimation improved or deterioated: values larger than one represent an improvment by using the conditional selectivity estimator.

[2]The used correlation factor is the $\phi^2$ factor from [IMH+04], which ranges between 0.0 - meaning independence - and 1.0 - meaning complete dependency.

| Nr. | #rows | #distinct | variance | $\phi^2$ | improvement |
|---|---|---|---|---|---|
| 1 | 10001 | 193 | 264.019 | 0.992588 | - (inf) |
| 2 | 14260 | 39 | 297.967 | 0.894439 | 1.753 |
| 3 | 12096 | 280 | 105.311 | 0.631532 | 1.506 |
| 4 | 12096 | 40 | 669.102 | 0.536271 | 0.828 |
| 5 | 10001 | 691 | 58.3671 | 0.201792 | 1.288 |
| 6 | 8872 | 1832 | 3.97732 | 0.00521 | 2.165 |

Table 1: Evaluated attribute pairs

To further illustrate the results, figure 2 shows scatter plots presenting the change in estimation errors for all six groups. Each dot of the plot represents one query, the x-coordinate being the estimation error using the uniformity approach, the y-coordinate the error using the conditional selectivity approach. Therefore each dot between the two lines represents a query for which the conditional selectivity approach resulted in an improved estimate.

Looking at the plot for group one, it shows all[3] points lying on the x-axis. This means the conditional selectivity approach resulted in - as expected - perfect estimates for the group with the highest correlation. The plots for groups two and three show most points in between the two lines: For these groups, the conditional selectivity approach results in a reasonable improvement. The same is true for groups five and six. The only group for which the classic uniformity approach worked (slightly) better was group four, which is mid-correlated and has a rather high variance.

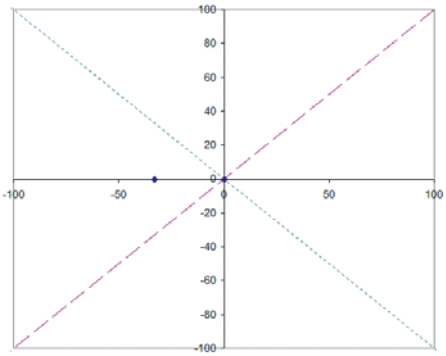
The experimental results are basically consistent with the considerations from the previous chapter: The conditional selectivity based approach seems to result in better estimates for lowly and especially for highly correlated groups. For mid-correlated groups, the two estimators seem to be equally good, with a slight advantage for the uniformity approach.

## 7 Conclusion & Future Work

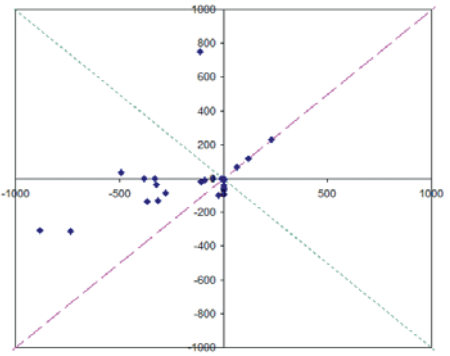
Summarizing the results, we can say that the conditional selectivity approach offers nearly perfect estimations for highly correlated attributes - without the need for the expensive construction and maintenance of a multidimensional histogram. Although the estimation quality decreases for lower correlated attributes, the method still seems to at least match the uniformity approach in most cases. Thus, the conditional selectivity based approach provides a clear advantage for highly correlated attributes, while it seemingly doesn't have a serious disadvantage on mildly correlated ones: The method looks very promising and should be studied further in the future.

There are two interesting topics future studies could concentrate on: the first topic is finding heuristics for determining cases in which the uniformity approach works better than the conditional selectivity approach. These heuristics could be used by an "intelligent op-

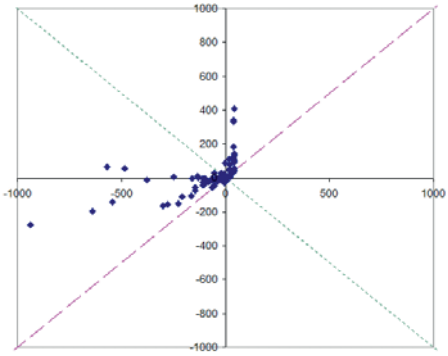
[3] Since many queries led to the same estimation errors, it appears there are only two points in the plot.



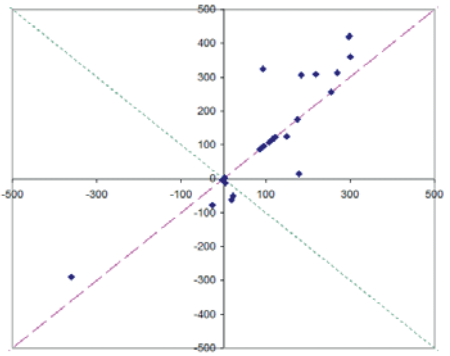

(a) group 1



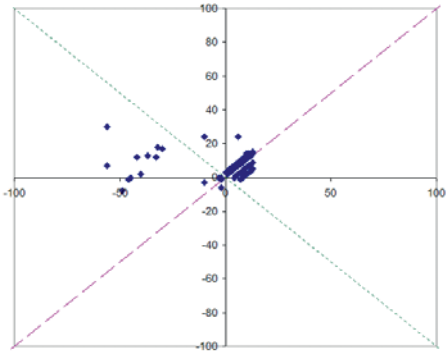

(b) group 2



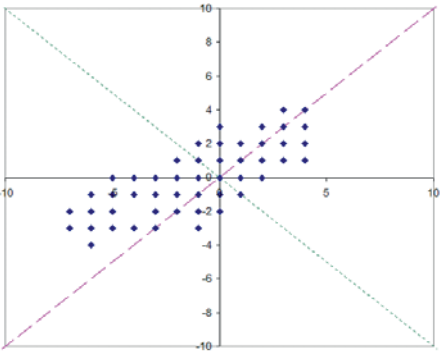

(c) group 3




(d) group 4




(e) group 5




(f) group 6

Figure 2: Comparing uniformity and conditional selectivity based estimation

timizer" to automatically determine the best possible estimator for a given relation. The second topic focuses on optimizing the conditional selectivity factors. There is probably a better way of choosing those factors than the proposed approach, which should lead to a better estimation quality for the conditional selectivity approach.

## References


[Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43, New York, NY, USA, 1998. ACM.

[GTK01] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. *SIGMOD Rec.*, 30(2):461–472, 2001.

[IMH+04] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, New York, NY, USA, 2004. ACM.

[Ioa03] Yannis Ioannidis. The history of histograms (abridged). In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 19–30. VLDB Endowment, 2003.

[MHK+07] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *The VLDB Journal*, 16(1):55–76, 2007.

[MPS99] S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 236–256, London, UK, 1999. Springer-Verlag.

[PI97] Viswanath Poosala and Yannis E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 486–495, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[PSC84] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 256–276, New York, NY, USA, 1984. ACM.

[SAC+79] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, New York, NY, USA, 1979. ACM.

[SHM+06] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. ISOMER: Consistent Histogram Construction Using Query Feedback. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 39, Washington, DC, USA, 2006. IEEE Computer Society.

[TGIK02] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. Dynamic multidimensional histograms. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 428–439, New York, NY, USA, 2002. ACM.

# Index-supported Similarity Join on Graphics Processors

Christian Böhm
University of Munich
boehm@dbs.ifi.lmu.de

Robert Noll
University of Munich
nollr@cip.ifi.lmu.de

Claudia Plant
Technische Universität München
plant@lrz.tum.de

Andrew Zherdin
Technische Universität München
andrew.zherdin@lrz.tum.de

**Abstract:** The similarity join is an important building block for similarity search and data mining algorithms. In this paper, we propose an algorithm for similarity join on Graphics Processing Units (GPUs). As major advantages GPUs provide extremely high parallelism combined with a high bandwidth in data transfer to main memory. To exploit these advantages for similarity join, we propose an index structure designed for the specific environment of GPU. Experiments demonstrate massive performance gains of our method over conventional similarity join on CPU and significant further speed-up by index support.

## 1 Introduction

In recent years, *Graphic Processing Units* (GPUs) have evolved from simple chips controlling the display device into powerful coprocessors supporting the CPU in various ways. Graphics applications such as realistic 3D games are computationally demanding and require a large number of complex algebraic operations for each update of the display image. Therefore, today's graphics hardware contains a large number (up to some hundreds) of simple but programmable processors which are optimized to cope with this high workload of vector, matrix and other computations in a highly parallel way.

In terms of peak performance, the highly parallel graphics hardware has outperformed state-of the art multi-core CPUs by a large margin. Therefore, there is a great effort in many research communities such as life sciences [LSVMW07, MV08] or data mining [CSK08] to use the computational capabilities of GPUs even for purposes which are not graphics-related at all. The corresponding research area is called General Processing-Graphical Processing Unit (GP-GPU). Vendors of graphics hardware have anticipated this trend and developed libraries, precompilers and application programming interfaces for this kind of programming. Most prominently, NVIDIA's technology *Compute Unified Device Architecture* (CUDA) offers a C++ programming interface in which both the *host program* to be executed on CPU as well as the so-called *kernel functions* to be executed in a massively parallel fashion on GPU are assembled in a single program [cud07]. An analogous technique is also offered by ATI under the brand names Close-to-Metal, Stream SDK, and Brook-GP.

The *similarity join* is a basic operation of a database system designed for similarity search and data mining on feature vectors. In such applications, we are given a large set $D$ of objects which are associated with a vector from a multidimensional space, the feature space. The similarity join determines pairs of objects which are similar to each other. The most widespread form is the $\epsilon$-join which determines those pairs from $D \times D$ which have a Euclidean distance of no more than a user-defined radius $\epsilon$:

**Definition 1 (Similarity Join)** *Let $D \subseteq \mathbb{R}^d$ be a set of feature vectors of a $d$-dimensional vector space and $\epsilon \in \mathbb{R}^+$ be a threshold. Then the similarity join is the following set of pairs:*

$$\text{SimJoin}(D, \epsilon) = \{(p, q) \in (D \times D) \quad | \quad \text{dist}(p, q) \leq \epsilon\},$$

where $\text{dist}(p, q) = \sqrt{(p-q)^{\mathbf{T}} \cdot (p-q)}$ is the Euclidean distance between $p$ and $q$.

If $p$ and $q$ are elements of the same set, the join is a similarity self-join. Most algorithms including the method proposed in this paper can also be generalized to the more general case of non-self-joins in a straightforward way. Algorithms for a similarity join with nearest neighbor predicates have also been proposed. The similarity join is a powerful building block for similarity search and data mining. It has been shown that important data mining methods such as clustering and classification can be based on the similarity join. Using a similarity join instead of single similarity queries can accelerate data mining algorithms by a high factor [BBBK00].

In this paper, we propose two algorithms for similarity join on the GPU. The first one is a parallelization of the Nested Loop Join (NLJ) with particular focus on the special demands of the graphics hardware. The second one is an indexed join algorithm which operates on an index structure which is particularly suited for the GPU. We demonstrate the superiority of our approach over the corresponding sequential join methods on CPU. The remainder of this paper is organized as follows: Section 2 quickly reviews the related work in similarity join processing and in GP-GPU processing in general. Section 3 explains the graphics hardware and the corresponding programming model. Section 4 and Section 5 are dedicated to the non-indexed and indexed NLJ on graphics hardware, respectively. Section 6 contains an extensive experimental evaluation of our technique, and Section 7 concludes this paper.

## 2 Related Work

For given multidimensional point data sets $A$ and $B$ similarity join operations can be classified depending on whether a multidimensional indexing structure exists on both data sets $A$ and $B$ [BKS93] or on neither [BBKK01, DS01]. There is a wide range of data structures that have been adopted for similarity join processing, quad-trees [DS01, KS00], R-trees [BKS93], $\epsilon$-kbd trees [SSA97], hash functions [GIM99], and space-filling curves [DS01] to name a few. Two recent methods are predicated on the application of grids to multidimensional point data sets, namely the *Epsilon Grid Order* (EGO) [BBKK01] and the *Generic External Space Sweep* (GESS) method [DS01]. EGO is based on a particular

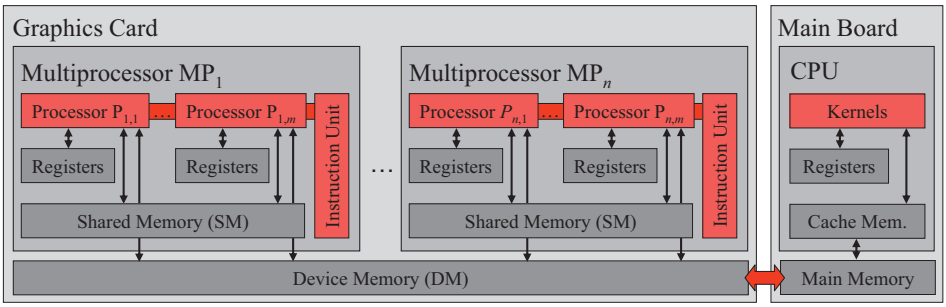



Figure 1: Architecture of a GPU.

sort order of the data points, which is obtained by laying an equidistant grid with cell length $\epsilon$ over the data space and comparing the grid cells lexicographically. By an external sorting algorithm and a particular scheduling strategy during the join phase, EGO avoids the problem of holding large portions of the data sets simultaneously in main memory. GESS constructs a hypercube with side length $\epsilon$ around each multidimensional input point. The similarity join problem is thus transformed to a spatial intersection [KS00] problem.

Some recent approaches, e.g. [GGKM06] and [GLW+04] demonstrate that important building blocks for query processing in databases, e.g. sorting can be significantly speed up by the use of GPUs. In [HYF+08] algorithms for relational join on GPU are presented. Probably the most related approach to ours is [LSS08], which is to the best of our knowledge the only paper on similarity join on GPU. The authors propose an algorithm based on the concept of space filling curves, e.g. the $z$-order, for pruning of the search space. The $z$-order of a set of objects can be determined very efficiently on GPU by highly parallelized sorting.

## 3 Architecture of the GPU

Graphic Processing Units (GPUs) of the newest generation are powerful coprocessors, not only designed for games and other graphic-intensive applications, but also for general-purpose computing (in this case, we call them GPGPUs). From the hardware perspective, a GPU consists of a number of multiprocessors, each of which consists of a set of simple processors which operate in a SIMD fashion, i.e. all processors of one multiprocessor execute in a synchronized way the same arithmetic or logic operation at the same time, potentially operating on different data. For instance, NVIDIA's GPU of the G80 series (e.g. the graphics card Geforce 8800 GTX) has 16 multiprocessors, each consisting of 8 processors, summarizing to a total amount of 128 processors inside one GPU. The architecture of the GPU is visualized in Figure 1: Apart from some memory units with special purpose in the context of graphics processing (e.g. texture memory), we have three important types of memory.

The *shared memory* (SM) is a memory unit with fast access (at the speed of register access, i.e. no delay). SM is shared among all processors of a multiprocessor. It can be used for local variables but also to exchange information between threads on different processors of the same multiprocessor. It cannot be used for information which is shared among threads on different multiprocessors. SM is fast but very limited in capacity (16 KBytes per multiprocessor). The second kind of memory is the so-called *device memory* (DM), which is the actual video RAM of the graphic card (also used for frame buffers etc.). DM is physically located on the graphics card (but not inside the GPU), is significantly larger than SM (typically up to some hundreds of MBytes), but also significantly slower. In particular, memory accesses to DM cause a typical latency delay of 400-600 clock cycles (G80), but the bandwidth for transferring data between DM and GPU (86 GB/s on G80) is higher than that of CPU and main memory (about 10 GB/s on current CPUs). DM can be used to share information between threads on different multiprocessors. If some threads schedule memory accesses from contiguous addresses, these accesses can be coalesced, i.e. taken together to improve the access speed. A typical cooperation pattern for DM and SM is to copy the required information from DM to SM simultaneously from different threads (if possible, considering coalesced accesses), then to let each thread compute the result on SM, and finally, to copy the result back to DM. The third kind of memory considered here is the *main memory* which is not part of the graphics card. The GPU has no access to the address space of the CPU. The CPU can only write to or read from DM using specialized API functions. The bottleneck of these operations is, of course, larger than DM accesses of the GPU.

The basis of the programming model of GPUs are threads. Threads are lightweight processes which are easy to create and to synchronize. In contrast to CPU processes, a context switch between different threads does not cause any considerable overhead either. In typical applications, thousands or even millions of threads are created, for instance one thread per pixel in gaming applications. The CUDA programming library [cud07] contains API functions to create a large number of threads on the GPU, each of which executes a function called *kernel function*. The kernel functions (which are executed in parallel on the GPU) as well as the host program (which is executed sequentially on the CPU) are defined in an extended C++ syntax. The kernel functions are restricted with respect to functionality (e.g. no recursion).

On GPUs the threads do not even have an individual instruction pointer. An instruction pointer is rather shared by several threads. For this purpose, threads are grouped into so-called *warps* (typically 32 threads per warp). One warp is processed simultaneously on the 8 processors of a single multiprocessor (SIMD) using 4-fold pipelining (totalling in 32 threads executed synchronously). To allow 4-fold pipelining, each processor contains also four arithmetic logic units (ALUs). If not all threads in a warp follow the same execution path, the different execution paths are executed in a serialized way.

Multiple warps are grouped into *thread groups* (TG). It is recommended [cud07] to use multiples of 64 threads per TG. The different warps in a TG (as well as different warps of different TGs) are executed independently. The threads in one thread group use the same shared memory and may thus communicate via the SM. The threads in one thread group can be synchronized (let all threads wait until all warps of the same group have

```
algorithm sequentialNLJ(data set D)
  for each q ∈ D do          // outer loop
    for each p ∈ D do        // inner loop: search all points p which are similar to q
      if dist(p, q) ≤ ε then
        report (p, q) as a result pair or do some further processing on (p, q)
end
```

Figure 2: Sequential Algorithm for the Nested Loop Join.

reached that point). The latency delay of the DM can be hidden by scheduling other warps of the same or a different thread group whenever one warp waits for an access to DM. To allow switching between warps of different thread groups on a multiprocessor, it is recommended [cud07] that each thread uses only a small fraction of the shared memory and registers of the multiprocessor.

## 4 Similarity Join Without Index Support

The baseline technique to process any join operation with an arbitrary join predicate is the nested loop join (NLJ) which performs two nested loops, each enumerating all points of the data set. For each pair of points, the distance is calculated and compared to $\epsilon$. The pseudocode of the sequential version of NLJ is given in Figure 2.

It is easily possible to parallelize the NLJ, e.g. by creating an individual thread for each iteration of the outer loop. The kernel function then contains the inner loop, the distance calculation and the comparison. During the complete run of the kernel function, the current point of the outer loop is constant, and we call this point the *query point* $q$ of the thread, because the thread operates like a similarity query, in which all database points with a distance of no more than $\epsilon$ from $q$ are searched. The query point $q$ is always held in a register of the processor.

Our GPU allows a truly parallel execution of a number $p$ of incarnations of the outer loop, where $p$ is the total number of ALUs of all multiprocessors (i.e. the warp size 32 times the number of multiprocessors). Moreover, all the different warps are processed in a quasi-parallel fashion, which allows to operate on one warp of threads (which is ready-to-run) while another warp is blocked due to the latency delay of a DM access of one of its threads.

The threads are grouped into thread groups, which share the SM. In our case, the SM is particularly used to physically store for each thread group the current point $p$ of the inner loop. Therefore, a kernel function first copies the current point $p$ from the DM into the SM, and then determines the distance of $p$ to the query point $q$. The threads of the same warp are running perfectly simultaneously, i.e. if these threads are copying the same point from DM to SM, this needs to be done only once (but all threads of the warp have to wait until this relatively costly copy operation is performed). However, a thread group may (and should) consist of multiple warps. To ensure that the copy operation is only performed once per thread group, it is necessary to synchronize the threads of the thread group before and after the copy operation using the API function synchronize(). This API function blocks all threads in the same TG until all other threads (of other warps) have

```
algorithm GPUsimpleNLJ(data set D)      // host program executed on CPU
  deviceMem float D'[][] := D[][];      // allocate memory in DM for the data set D
  #threads := |D|;      // number of points in D
  #threadsPerGroup := 64;
  startThreads (simpleNLJKernel, #threads, #threadsPerGroup);      // one thread per point
  waitForThreadsToFinish();
end.

kernel simpleNLJKernel (int threadid)
  register float q[] := D'[threadid][] ;      // copy the point from DM into the register
                                              // and use it as query point q
                                              // index is determined by the thread-id
  for i := 0 ... n − 1 do      // this used to be the inner loop in Figure 2
    synchronizeThreadGroup();
    shared float p[] := D'[i][];      // copy the current point p from DM to SM
    synchronizeThreadGroup();      // Now all threads of the thread group can work with p
    if dist(p, q) ≤ ε then
      report (p, q) as a result pair using synchronized writing
      or do some further processing on (p, q) directly in kernel
end.
```

Figure 3: Parallel Algorithm for the Nested Loop Join on the GPU.

reached the same point of execution. The pseudocode for this algorithm is presented in Figure 3.

If the data set does not fit into DM, a simple partitioning strategy can be applied. It must be ensured that the potential join partners of an object are within the same partition as the object itself. Therefore, overlapping partitions of size $2 \cdot \epsilon$ can be created.

## 5 An Index Structure to Support the Similarity Join on GPU

The performance of the NLJ can be greatly improved if an index structure is available. On sequential processing architectures, the indexed NLJ leaves the outer loop unchanged. The inner loop is replaced by an index-based search retrieving candidates that may be join partners of the current object of the outer loop. The effort of finding these candidates and refining them is often orders of magnitude smaller compared to the non-indexed NLJ.

When parallelizing the indexed NLJ for the GPU, we follow the same paradigm as in the last section, to create an individual thread for each point of the outer loop. It is beneficial to the performance, if points having a small distance to each other are collected in the same warp and thread group, because for those points, similar paths in the index structure are relevant. We will see later how this grouping of similar points is easily achieved.

Our index structure needs to be traversed in parallel for many search objects using the kernel function. Since kernel functions do not allow any recursion, and as they need to have small storage overhead by local variables etc., the index structure must be very simple as well. To achieve a good compromise between simplicity and selectivity of the index, we propose a data partitioning method with a constant number of directory levels. The first level partitions according to the first dimension of the data space, the second level according to the second dimension, and so on. Therefore, before performing the join,

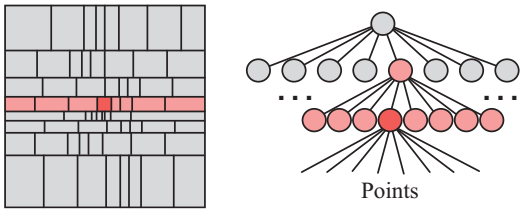



Figure 4: Index Structure for GPU.

some transformation technique should be applied which guarantees a high selectivity in the first dimensions (e.g. Principle Component Analysis, Fast Fourier Transform, Discrete Wavelet Transform, etc.). Figure 4 shows a simple, 2-dimensional example of a 2-level directory (plus the root node which is considered as level-0), similar to [KHT89, LEL97]. The fanout of each node is 8. In our experiments in Section 6, we used a three-level directory with fanout 16. Our algorithm now changes only a little bit. Before starting the actual join processing, our simple index structure must be constructed in a bottom-up way by fractionated sorting of the data: First, the data set is sorted according to the first dimension, and partitioned into the specified number of quantile partitions. Then, each of the partitions is sorted individually according to the second dimension, and so on. The boundaries are stored using simple arrays which can be easily accessed in the subsequent kernel functions. In principle, this index construction can already be done on the GPU, because efficient sorting methods for GPU have been proposed [GGKM06]. Since bottom up index construction is typically not very costly compared to the join algorithm, our method performs this preprocessing step on CPU. After index construction, we have not only a directory in which the points are organized in a way that facilitates search. Moreover, the points are now clustered in the array, i.e. points which have neighboring addresses are also likely to be close together in the data space (at least when projecting on the first few dimensions). Both effects are exploited by our join algorithm displayed in Figure 5.

Instead of performing an outer loop like in a sequential indexed NLJ, our algorithm now generates a large number of threads: One thread for each iteration of the outer loop (i.e. for each query point $q$). Since the points in the array are clustered, the corresponding query points are close to each other, and the join partners of all query points in a thread group are likely to reside in the same branches of the index as well. Our kernel method now iterates over three loops, each loop for one index level, and determines for each partition if the point is inside the partition or, at least no more distant to its boundary than $\epsilon$. The corresponding subnode is accessed if the corresponding partition is able to contain join partners of the current point of the thread. When considering the warps which operate in a fully synchronized way, a node is accessed, whenever at least one of the query points of the warps is close enough to (or inside) the corresponding partition.

For both methods, indexed and non-indexed nested loop join on GPU, we need to address the question how the resulting pairs are processed. The most general technique is to report the results to the CPU where they can be further processed and permanently stored if

```
algorithm GPUindexedJoin(data set D)
  deviceMem index idx := makeIndexAndSortData(D); // changes ordering of datapoints
  int #threads := |D|, #threadsPerGroup := 64;
  for i = 1 ... (#threads/#threadsPerGroup) do
    deviceMem float blockbounds[i][] := calcBlockBounds(D, blockindex);
  deviceMem float D'[][] := D[][];
  startThreads (indexedJoinKernel, #threads, #threadsPerGroup); // one thread per datapoint
  waitForThreadsToFinish ();
end.

algorithm indexedJoinKernel (int threadid, int blockid)
  register float q[] := D'[threadid][] ;      // copy the point from DM into the register
  shared float myblockbounds[] := blockbounds[blockid][];
  for x_i := 0 ... indexsize.x do
    if IndexPageIntersectsBoundsDim1(idx,myblockbounds,x_i) then
      for y_i := 0 ... indexsize.y do
        if IndexPageIntersectsBoundsDim2(idx,myblockbounds,x_i, y_i) then
          for z_i := 0 ... indexsize.z do
            if IndexPageIntersectsBoundsDim3(idx,myblockbounds,x_i, y_i, z_i) then
              for w := 0 ... IndexPageSize do
                synchronizeThreadGroup();
                shared float p[] :=GetPointFromIndexPage(idx,D', x_i, y_i, z_i, w);
                synchronizeThreadGroup();
                if dist(p, q) ≤ ε then
                  report (p, q) as a result pair using synchronized writing
end.
```

Figure 5: Algorithm for Similarity Join on GPU with Index Support.

required. This is easily possible by a buffer in DM which can be copied to the CPU after the termination of all kernel threads. The result pairs must be written into this buffer in a synchronized way to avoid that two threads write simultaneously to the same buffer area. The CUDA API provides atomic operations (such as atomic increment of a buffer pointer) to guarantee this kind of synchronized writing. Buffer overflows are also handled by our similarity join methods. If the buffer is full, all threads terminate and the work is resumed after the buffer is emptied by the CPU.

## 6 Experimental Evaluation

To evaluate the performance of similarity join on GPU, we compare four different variants for performing similarity join: (1) nested loop join on CPU, (2) NLJ on CPU with index support, (3) NLJ on GPU and (4) NLJ on GPU with index support. All variants have been implemented in C++. The index structure described in Section 5 was used in variant (2) and (4). The experiments have been performed on a workstation with Intel Core 2 Duo CPU E4500 2.2 GHz and 2 GB RAM which is supplied with a NVIDIA 8500GT GPU (2 multiprocessors each consisting of 8 processors) with 512MB VRAM.

We generated three 8-dimensional synthetic data sets of various sizes (up to 10 million points) with different data distributions: Data set DS1 contains uniformly distributed data. DS2 consists of 5 Gaussian clusters which are randomly distributed in feature space. Similar to DS2, DS3 is also composed of 5 Gaussian clusters, but the clusters are correlated. The threshold $\epsilon$ was selected to obtain a join result where each point was combined with one or two join partners on average.

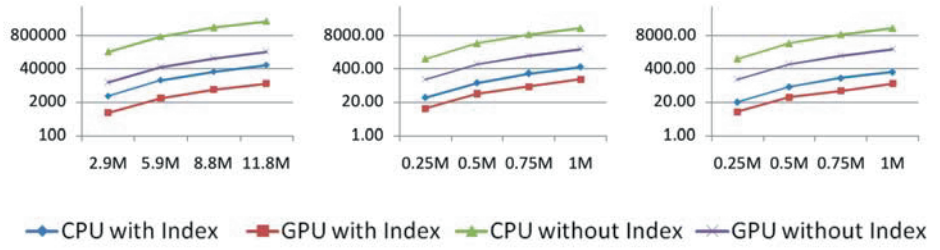



Figure 6: Experimental Results: Runtime in seconds on different data volumes ranging from 250,000 (0.25 M) to 11.8 (11.8 M) million points.

Figure 6 displays the runtime in seconds of all four variants for similarity join in logarithmic scale. The time needed for data transfer from CPU to GPU and back as well as the (negligible) index construction time has been included. NLJ on GPU with index support performs best in all experiments. Note that, due to massive parallelization, NLJ on GPU without index support outperforms CPU without index by a large factor (15.9 on 180 MByte of uniform data). The GPU algorithm with index support outperforms the corresponding CPU algorithm (with index) by a factor of 4.6. The overall improvement of the indexed GPU algorithm over the non-indexed CPU version is 246. This result demonstrates the potential of boosting performance of database operations with designing specialized index structures and algorithms for GPU. Further experiments (varying dimensionality, join selectivity ($\epsilon$) and fanout and number of levels of the index) were conducted but must be left out due to space limitations. The speedup of the GPU versions over the corresponding CPU versions was always in the same order of magnitude like in the experiments shown here.

## 7 Conclusions

In this paper, we have addressed the question, how similarity join algorithms can be efficiently executed on Graphics Processing Units (GPU). We have proposed an index structure which is particularly suited to be held in the Device Memory of the GPU and which can be accessed by kernel functions of the GPU in an efficient way. Since graphics processors correspond to a highly parallel architecture consisting of up to some hundreds of single programmable processors, we have also shown how similarity join algorithms can be parallelized. We have proposed two efficient similarity join algorithms, one based on the nested loop join and the other based on the indexed nested loop join. Both algorithms are particularly dedicated for GPU processing and outperform the sequential algorithms by a large factor.

## References


[BBBK00] Christian Böhm, Bernhard Braunmüller, Markus M. Breunig, and Hans-Peter Kriegel. High Performance Clustering Based on the Similarity Join. In *CIKM*, pages 298–305, 2000.

[BBKK01] Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data. In *SIGMOD*, pages 379–388, 2001.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *SIGMOD*, pages 237–246, 1993.

[CSK08] Bryan Catanzaro, Narayan Sundaram, and Kurt Keutzer. Fast Support Vector Machine Training and Classification on Graphics Processors. In *ICML 2008*, pages 104–111, 2008.

[cud07] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[DS01] Jens-Peter Dittrich and Bernhard Seeger. GESS: a scalable similarity-join algorithm for mining large data sets in high dimensional spaces. In *KDD*, pages 47–56, 2001.

[GGKM06] Naga K. Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *SIGMOD*, pages 325–336, 2006.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *VLDB*, pages 518–529, 1999.

[GLW+04] Naga K. Govindaraju, Brandon Lloyd, Wei Wang, Ming C. Lin, and Dinesh Manocha. Fast Computation of Database Operations using Graphics Processors. In *SIGMOD*, pages 215–226, 2004.

[HYF+08] Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational joins on graphics processors. In *SIGMOD*, pages 511–524, 2008.

[KHT89] Masaru Kitsuregawa, Lilian Harada, and Mikio Takagi. Join Strategies on KD-Tree Indexed Relations. In *ICDE*, pages 85–93, 1989.

[KS00] Nick Koudas and Kenneth C. Sevcik. High Dimensional Similarity Joins: Algorithms and Performance Evaluation. *IEEE Trans. Knowl. Data Eng.*, 12(1):3–18, 2000.

[LEL97] Scott T. Leutenegger, J. M. Edgington, and Mario A. Lopez. STR: A Simple and Efficient Algorithm for R-Tree Packing. In *ICDE*, pages 497–506, 1997.

[LSS08] Michael D. Lieberman, Jagan Sankaranarayanan, and Hanan Samet. A Fast Similarity Join Algorithm Using Graphics Processing Units. In *ICDE*, pages 1111–1120, 2008.

[LSVMW07] Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. Molecular Dynamics Simulations on Commodity GPUs with CUDA. In *HiPC*, pages 185–196, 2007.

[MV08] Svetlin Manavski and Giorgio Valle. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, 9(Suppl 2), 2008.

[SSA97] Kyuseok Shim, Ramakrishnan Srikant, and Rakesh Agrawal. High-Dimensional Similarity Joins. In *ICDE*, pages 301–311, 1997.

# Systemübergreifende Kostennormalisierung für Integrationsprozesse

Matthias Böhm[1], Dirk Habich[2], Wolfgang Lehner[2], und Uwe Wloka[1]

[1] Hochschule für Technik und Wirtschaft Dresden (FH), Fachgebiet Datenbanksysteme
mboehm@informatik.htw-dresden.de
wloka@informatik.htw-dresden.de
[2] Technische Universität Dresden, Lehrstuhl Datenbanken
dirk.habich@tu-dresden.de
wolfgang.lehner@tu-dresden.de

**Zusammenfassung** Auf Grundlage der Vielzahl proprietärer Integrationssysteme ist zunehmend die Entwicklung von Ansätzen zur modellbasierten Generierung von Integrationsprozessen zu beobachten. Eine derartige Generierung bietet weiterhin die Möglichkeit der Auswahl des optimalen Integrationssystems, welche ein hohes Optimierungspotenzial in sich birgt. Die Grundlage für eine solche Entscheidung ist jedoch eine integrationssystemübergreifende Kostennormalisierung, um die Vergleichbarkeit von Verarbeitungsstatistiken zu ermöglichen. Basierend auf einem plattformunabhängigen Kostenmodell und der systemübergreifenden Kostennormalisierung kann eine kostenbasierte Optimalitätsentscheidung hinsichtlich des effizientesten Integrationssystems getroffen werden. Folglich können hierbei veränderliche Workload-Charakteristika in die Betrachtung einbezogen werden. In diesem Papier zeigen wir zunächst die Probleme der systemübergreifenden Kostenmodellierung und Kostennormalisierung auf, welche bislang noch nicht betrachtet wurden. Darauf aufbauend diskutieren wir die plattforminvariante Kostenmodellierung und die Erhebung von Verarbeitungsstatistiken. Weiterhin führen wir drei Algorithmen zur Kostennormalisierung ein, mit denen die Vergleichbarkeit von Verarbeitungsstatistiken sichergestellt wird. Als Konsequenz kann das vorgestellte Konzept in beliebigen Ansätzen der Generierung von Integrationsprozessen zum Einsatz kommen.

## 1 Einleitung

Integrationsprozesse – im Sinne von workflowbasierten Integrationsaufgaben – gewinnen zunehmend an Bedeutung, da ein kontinuierlicher Wandel des Datenmanagements von der Verwaltung zentral vorgehaltener Daten hin zur Verwaltung von hochgradig verteilten und heterogenen Datenbeständen und Anwendungen erfolgt. In diesem Kontext ist die Effizienz kompletter IT-Infrastrukturen von den eingesetzten Integrationsplattformen abhängig. Beispiele für derartige Plattformen sind EAI-Server (Enterprise Application Integration), ETL-Werkzeuge (Extraction Transformation Loading), föderierte DBMS, WfMS (Workflow Management Systems) und WSMS (Web Service Management Systems). Um die Probleme des hohen Entwicklungsaufwands und der geringen Portabilität zu reduzieren, wurden Ansätze [BWHL08,DHW+08,AN08] zur modellbasierten Generierung von Integrationsprozessen entwickelt. Derartige Generierungsansätze bieten die Möglichkeit der Auswahl des optimalen Integrationssystems.

Um sich verändernde Workload-Charakteristika in ausreichendem Maße einzubeziehen, sollte diese Optimalitätsentscheidung kostenbasiert (auf Basis von Verarbeitungsstatistiken) erfolgen. Dies eröffnet jedoch das Problem des Kostenvergleichs heterogener Integrationsplattformen, im Sinne der plattformunabhängigen Kostenmodellierung und der entsprechenden Kostennormalisierung konkreter Statistiken in das plattformunabhängige Modell. Mit dem Ziel der Herstellung einer systemübergreifenden Vergleichbarkeit sind die Aspekte Parallelität, Ressourcenverwendung, unterschiedliche Hardware und Verarbeitungsmodelle, Semantik der erhobenen Statistiken sowie fehlende und inkonsistente Statistiken im Rahmen der Normalisierung zu betrachten.

Um das Problem der systemübergreifenden Kostennormalisierung und einen möglichen Lösungsansatz aufzuzeigen, leisten wir in diesem Papier folgende Beiträge:

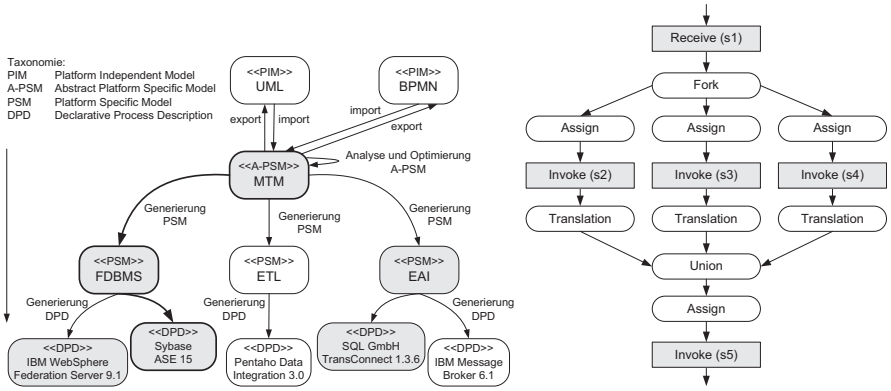
- In Abschnitt 2 erläutern wir die wesentliche Problemstellung der systemübergreifenden Kostennormalisierung für heterogene Integrationssysteme. Dabei präsentieren wir auch unsere generellen Annahmen zur Auswahlmöglichkeit von Integrationssystemen und beschreiben unsere Vision des *Invisible Deployments*.
- Aufbauend auf der Problembeschreibung diskutieren wir die plattformunabhängige Modellbildung in Abschnitt 3. Hier stellen wir sowohl ein plattforminvariantes Kostenmodell als auch Anforderungen an die Erhebung von Statistiken dar.
- In Abschnitt 4 führen wir dann drei Algorithmen zur Kostennormalisierung ein, welche plattformspezifische Verarbeitungsstatistiken in das plattforminvariante Kostenmodell überführen und damit die Vergleichbarkeit herstellen.
- Danach präsentieren wir die experimentelle Evaluierung in Abschnitt 5.

Abschließend zeigen wir mögliche Anwendungsgebiete im Rahmen der Analyse verwandter Arbeiten in Abschnitt 6 und fassen das Papier in Abschnitt 7 zusammen.

## 2 Vision und Problembeschreibung

In diesem Abschnitt sollen die Problematik der plattformübergreifenden Kostenmodellierung und die entsprechende Kostennormalisierung näher diskutiert werden. Deshalb wollen wir zunächst kurz auf unser GCIP Framework (Generation of Complex Integration Processes) [BWHL08] (als ein repräsentativer Ansatz für die modellgetriebene Generierung von Integrationsprozessen) eingehen. Das Grundkonzept der Generierung bei diesem Ansatz ist in Abbildung 1(a) dargestellt. In diesem Rahmen können Integrationsprozesse auf plattformunabhängiger Ebene (PIM[3]) mit UML-Aktivitätsdiagrammen oder BPMN-Prozessbeschreibungen modelliert werden. Diese Modelle werden anschließend in eine zentrale Repräsentation (das sogenannte Message Transformation Model (MTM) [BBW+07]) importiert. Ausgehend von dieser Repräsentation erlaubt das Framework die Generierung von Integrationsaufgaben für unterschiedliche Integrationssystemtypen (PSM[4]). Momentan werden die Typen FDBMS, ETL und EAI unterstützt. Ausgehend von den plattformspezifischen Repräsentationen werden dann Prozessbeschreibungen für unterschiedliche konkrete Integrationssysteme generiert. Das folgende Beispiel veranschaulicht einen typischen Integrationsprozess.

[3] Platform-Independent Model (PIM): Eine plattformunabhängige Prozessbeschreibung
[4] Platform-Specific Model (PSM): Eine systemtypspezifische Prozessbeschreibung




(a) Generierungskonzept (b) Beispiel Integrationsprozess

Abbildung 1: Grundkonzept GCIP Framework

**Beispiel 1** Integrationsprozess: *Abbildung 1(b) zeigt ein Beispiel für die zentrale Repräsentation (MTM). Dieser Integrationsprozess beschreibt den Empfang einer Nachricht von System $s1$ sowie drei nebenläufige Teilprozesse, in denen die externen Systeme $s2$, $s3$ und $s4$ angefragt werden und ein Schema-Mapping der Resultate in ein einheitliches Schema vorgenommen wird. Abschließend erfolgt die Vereinigung der Ergebnismengen und die Übertragung an ein System $s5$. Dieser Integrationsprozess kann auf eine Reihe von Integrationssystemtypen abgebildet werden. Beispielsweise kann ein Trigger für FDBMS oder ein WSBPEL-Prozess für EAI-Server generiert werden.*

## 2.1 Grundlegende Annahmen

Die Schlussfolgerungen der Generierung von Integrationsprozessen lassen sich zu zwei wesentlichen Annahmen konkretisieren:

**Annahme 1** Generierbarkeit: *Integrationsprozesse können plattformunabhängig modelliert werden. Darauf aufbauend ist es möglich, proprietäre Beschreibungen von Integrationsprozessen für konkrete Integrationssysteme zu generieren.*

**Annahme 2** Auswahlmöglichkeit: *In einer typischen IT-Infrastruktur existieren mehrere Integrationssysteme mit überlappenden Funktionalitäten. Folglich besteht die Möglichkeit der Auswahl des optimalen Integrationssystems.*

Aufbauend auf der Annahme der Generierbarkeit von Prozessbeschreibungen konkreter Integrationssysteme (durch verwandte Arbeiten [BWHL08,DHW+08,AN08] getestet und nachgewiesen), ist folglich die Voraussetzung einer Wahlmöglichkeit zwischen alternativen Integrationssystemen gegeben. Weiterhin ist auch die Praxisrelevanz dieser Wahlmöglichkeit dadurch belegt, dass typische IT-Infrastrukturen eine Menge an unterschiedlichen Integrationssystemen mit überlappenden Funktionalitäten (wie beispielsweise spezielle Operatoren, unterstützte externe Systeme, Möglichkeiten auf externe Ereignisse zu reagieren oder transaktionale Eigenschaften) beinhalten [Sto02]. Folglich kann ein gegebener Integrationsprozess mit Hilfe unterschiedlicher Integrationssysteme realisiert werden, ohne das externe Verhalten zu beeinflussen. Somit kann die Auswahlentscheidung

auf Grundlage von nichtfunktionalen Eigenschaften, wie beispielsweise Effizienz, Skalierbarkeit und Ressourcenverbrauch – und damit kostenbasiert – getroffen werden.

### 2.2 Vision des *Invisible Deployment*

Auf den beschriebenen Annahmen beruht unsere langfristige Vision des *Invisible Deployment* von Integrationsprozessen. Die Kernidee dieser Vision ist es die Ebene der Integrationssysteme und -technologien systematisch zu abstrahieren und letztendlich transparent zu gestalten (zu virtualisieren). Neben Konfigurationsmanagement und einigen anderen Herausforderungen, besteht hier die Notwendigkeit der Auswahl des optimalen Integrationssystems. Diese Auswahl soll in einem zweistufigen Verfahren erfolgen. In einem ersten Schritt (*Schritt 1: Auswahl Kandidatenmenge*) ist zunächst eine Kandidatenmenge von Integrationssystemen zu bestimmen, welche aus funktionaler Sicht zur Realisierung eines gegebenen Integrationsprozesses eingesetzt werden können. Hierzu sind die funktionalen Anforderungen aus dem Integrationsprozess abzuleiten und mit den Merkmalsvektoren [BF07] existierender Integrationssysteme zu vergleichen. Auf Grundlage der Kandidatenmenge kann in einem zweiten Schritt (*Schritt 2: Auswahl Optimales Integrationssystem*) eine kostenbasierte Optimalitätsentscheidung – ausschließlich basierend auf nichtfunktionalen Eigenschaften – getroffen werden. Einen Teilschritt zum Erreichen dieser Vision könnte ein *Advisor* zur Unterstützung von Auswahlentscheidungen darstellen. Dabei werden normalisierte Kosten und das definierte plattforminvariante Kostenmodell dazu verwendet dem Nutzer eine *What-If*-Schnittstelle – im Sinne von Plan- und Kostendiagrammen [RH05] der unterschiedlichen Systeme für veränderliche Workload-Charakteristika – bereit zu stellen.

### 2.3 Problemstellungen der Kostenmodellierung und Kostennormalisierung

Zur Realisierung des Invisible Deployment ist die kostenbasierte Auswahl von Integrationssystemen eine zwingende Voraussetzung. Derartige kostenbasierte Entscheidungen erfordern im Wesentlichen zwei weitere Grundvoraussetzungen. Dies ist zum einen ein plattforminvariantes Kostenmodell, welches für Entscheidungen über Integrationssysteme hinweg zum Einsatz kommen kann. Zum anderen ist die Kostennormalisierung (Kostentransformation und Kostenvervollständigung) von eminenter Bedeutung. Im Rahmen dieser Normalisierung sind folglich Verarbeitungsstatistiken der einzelnen Systeme zu erheben und entsprechend zu transformieren und zu vervollständigen.

Als Konsequenz stellen also die Kostenmodellierung und Kostennormalisierung ein grundlegendes Problem in diesem Kontext dar. Dabei ist das Kostenmodell ausschließlich im Rahmen von Integrationsprozessen einsetzbar, während die Problematik der systemübergreifenden Kostennormalisierung auch in anderen Domänen Anwendung finden kann. Da im Hinblick auf diese Grundvoraussetzungen jedoch keinerlei Ergebnisse existieren, ist hier grundlegende Forschungsarbeit zu leisten. Gerade bei der Kostennormalisierung offenbaren sich hier folgende, grundlegende Problemstellungen:

- *Parallelität:* Im Rahmen der Normalisierung ist die Parallelität sowohl von Prozessinstanzen als auch Operatorinstanzen zu beachten. Darin ist die generelle Arbeitslast der gesamten Plattform einzubeziehen.

- *Ressourcenverwendung:* Neben der Parallelität ist ebenfalls die Ressourcenverwendung der einzelnen Instanzen und Operatoren zu analysieren. Hierbei ist die effektive Ressourcenverwendung $R_e(o_i)$ als auch die maximal sinnvolle Ressourcenverwendung $R_o(o_i)$ eines Operators $o_i$ zu beachten.
- *Unterschiedliche Hardware:* Ein wesentliches Problem ist die unterschiedliche Hardware, auf der die jeweiligen Integrationssysteme ausgeführt werden. Im Rahmen der Normalisierung muss dieser Unterschied ausgeglichen werden.
- *Unterschiedliche Verarbeitungsmodelle:* Die Spezifika des jeweiligen Verarbeitungsmodells müssen in die Normalisierung einbezogen werden. Beispiele hierfür stellen die synchrone und asynchrone Verarbeitung als auch die Unterscheidung in das instanzbasierte und das vektorisierte (pipes and filter) Verarbeitungsmodell dar.
- *Semantik erhobener Statistiken:* Die Semantik der erhobenen Statistiken muss vereinheitlicht werden. Dies bedeutet, dass die Verarbeitungszeit die gleichen Aufgaben (Operatoren, Kardinalitäten) umfasst.
- *Fehlende Statistiken:* auf Grund der unterschiedlichen Verarbeitungsmodelle und primären Anwendungsgebiete der einzelnen Integrationssysteme müssen fehlende Statistiken (Statistiken, welche nicht erhoben werden können) auf Grundlage anderer Statistiken interpoliert oder geschätzt werden.
- *Inkonsistente Statistiken:* Auf Grund der Erhebung von sehr feingranularen Statistiken kann die Inkonsistenz zwischen Statistiken nicht ausgeschlossen werden und ist folglich im Rahmen der Normalisierung zu beseitigen.
- *Statistischer Fehler:* Im Rahmen von mehrmaligen Schätzungen von systemübergreifenden Kosten sollte der statistische Fehler (Differenz zwischen Erwartungswert und erhobenen Statistiken) reduziert und adaptiv minimiert werden.

Die aufgezeigten Probleme werden (mit Einschränkungen bei unterschiedlichen Verarbeitungsmodellen und der Minimierung des statistischen Fehlers) im Rahmen dieses Papiers mit einem plattforminvarianten Kostenmodell und geeigneten Algorithmen zur Kostennormalisierung adressiert.

**Annahme 3** *Der im Rahmen dieser Arbeit vorgestellte Ansatz zur Kostennormalisierung ermöglicht die Vergleichbarkeit der Kosten (im Sinne der Verarbeitungseffizienz) von Integrationsprozessen über Integrationssysteme und Plattformen hinweg.*

An dieser Stelle sei weiterhin auf das Hauptproblem bei der Kostennormalisierung verwiesen. So ist die Kostennormalisierung per Definition unidirektional. Dies hat Auswirkungen auf unsere Vision des *Invisible Deployment*. Es kann ausschließlich eine Normalisierung der Verarbeitungsstatistiken konkreter Integrationssysteme in eine plattforminvariante (damit normalisierte) Form erfolgen, da eine denormalisierte Repräsentation auf exakt eine normalisierte Form abgebildet wird. Hingegen ist eine Denormalisierung abstrakter Kosten in Verarbeitungsstatistiken spezifischer Integrationssysteme nicht sinnvoll, da eine Normalform in eine unendliche Menge denormalisierter Formen transformiert werden kann.

## 3 Plattforminvariante Modellbildung

Im Rahmen dieses Abschnitts präsentieren wir ein plattformunabhängiges Kostenmodell für Integrationsprozesse. Weiterhin erläutern wir das Verfahren der Erhebung von Statistiken sowie deren Transformation und Vervollständigung.

Tabelle 1: Abstrakte und normalisierte Kosten der Operatoren des MTM [BBW+07]

| **Operatorname** | **Abstrakte Kosten** $C(o_l)$ | **Normalisierte Kosten** $NC(o_l)$ |
|---|---|---|
| Receive | $\|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{out}\|}$ |
| Invoke | $\|ds_{in}\| + \|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$ |
| Reply | $\|ds_{in}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|}$ |
| Switch | - | $\sum_{i=1}^{n} P(pa_i)\left(\sum_{j=1}^{i} t_e(exppa_j) + \sum_{k=1}^{m_i} t_e(o_{i,k})\right)$ |
| Fork | - | $\max_{i=1}^{n}\left(\sum_{j=1}^{m_i} t_e\left(o_{i,j}\right) + i \cdot C\left(thread\right)\right)$ |
| Iteration | - | $m \cdot \sum_{i=1}^{n} t_e\left(o_i\right)$ |
| Delay | - | $t_e(o_l)$ |
| Signal | - | 0 |
| Assign | $\|ds_{in}\| + \|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$ |
| Translation | $\|ds_{in}\| + \|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$ |
| Selection | $\sigma = \|ds_{in}\|$ OR $\sigma_{sort} = \frac{\|ds_{in}\|}{2}$ | $\frac{t_e(o_l)}{\|ds_{in}\|}$ OR $\frac{t_e(o_l)}{\frac{\|ds_{in}\|}{2}}$ |
| Projection | $\|ds_{in}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|}$ |
| Join | $\bowtie_{NL} = \|ds_{in1}\| + \|ds_{in1}\| \cdot \|ds_{in2}\|$ | $\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in1}\|\cdot\|ds_{in2}\|}$ |
| | $\bowtie_{SM} = \|ds_{in1}\| + \|ds_{in2}\|$ | $\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in2}\|}$ |
| Setoperation | $\cup_{all} = \|ds_{in1}\| + \|ds_{in2}\|$ | $\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in2}\|}$ |
| | $\cup, \cap, - = \|ds_{in1}\| + \|ds_{in1}\| \cdot \|ds_{in2}\|$ | $\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in1}\|\cdot\|ds_{in2}\|}$ |
| Split | $\|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{out}\|}$ |
| Orderby | $\|ds_{in}\| \cdot \log(\|ds_{in}\|)$ | $\frac{t_e(o_l)}{\|ds_{in}\|\cdot\log(\|ds_{in}\|)}$ |
| Groupby | $\|ds_{in}\| + \frac{\|ds_{in}\|^2}{2}$ | $\frac{t_e(o_l)}{\|ds_{in}\|+\frac{\|ds_{in}\|^2}{2}}$ |
| Window | $\frac{\|ds_{in}\|^2}{2}$ | $\frac{t_e(o_l)}{\frac{\|ds_{in}\|^2}{2}}$ |
| Validate | $\|ds_{in}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|}$ |
| Savepoint | $\|ds_{in}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|}$ |
| Action | $\|ds_{in}\| + \|ds_{out}\|$ | $\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$ |

### 3.1 Kostenmodell

Das plattforminvariante Kostenmodell definiert Kosten für die einzelnen Operatoren des zentralen Message Transformation Models (MTM) [BBW+07]. Das Hauptproblem bei der Definition dieses Kostenmodells ist der systemübergreifende Charakter des Anwendungsgebietes. So genügt es auf der einen Seite eben *nicht*, lediglich abstrakte Kosten (beispielsweise auf Ebene von Kardinalitäten, wie im Datenbankbereich) miteinander zu vergleichen, da man hier nicht nur alternative Pläne für ein und dasselbe System miteinander vergleicht. Vielmehr müssen hier die normalisierten Verarbeitungszeiten der einzelnen Systeme einbezogen werden, um eine absolute Vergleichbarkeit zu schaffen. Auf der anderen Seite genügt aber auch der Vergleich der normalisierten Verarbeitungszeiten *nicht*, da dies kein Indiz über die Arbeitslast (im Sinne von Kardinalitäten) ist.

Folglich definieren wir ein zweistufiges Kostenmodell für die einzelnen Operatoren des MTM. Dieses enthält abstrakte Kosten $C(o_l)$ und normalisierte Kosten $NC(o_l)$ (siehe Ta-

belle 1). Dabei basieren die abstrakten Kosten auf Kardinalitäten, während die normalisierten Kosten mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$ aus den normalisierten (effektiven) Verarbeitungszeiten $t_e(o_l)$ und den abstrakten Kosten $C(o_l)$ berechnet werden. Damit kann dem systemübergreifenden Charakter Rechnung getragen werden. Im Minimum werden Kardinalitäten und Verarbeitungszeiten mit der Granularität einzelner Operatoren benötigt (Anforderungen an die Erhebung). Daraus können weitere Statistiken (wie zum Beispiel die relativen Häufigkeiten der Nutzung alternativer Prozesspfade) abgeleitet werden.

Im Detail sind die Operatoren des MTM in interaktions-, kontrollfluss- und datenflussorientierte Operatoren zu unterscheiden. Die abstrakten Kosten der Gruppe der interaktionsorientierten Operatoren sind aus der Semantik dieser abgeleitet. So kann `Receive` lediglich Daten empfangen und `Reply` lediglich Daten zurückgeben, während ein `Invoke` sowohl Input- als auch Output- Daten aufweisen kann. Die normalisierten Kosten ergeben sich hier mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$. Im Gegensatz dazu weisen die kontrollflussorientierten Operatoren keine abstrakten Kosten auf, da hier keine Datenoperationen vorgenommen werden. Die normalisierten Kosten ergeben sich hier wiederum aus der spezifischen Semantik der einzelnen Operatoren, wobei `Switch`, `Fork` und `Iteration` komplexe Operatoren darstellen und somit ebenfalls Kosten der enthaltenen Operatoren $o_i$ einschließen. Für die Menge an datenflussorientierten Operatoren adaptierten wir das in [Mak07] dargestellte Kostenmodell eines relationalen Anfrageoptimierers und erweiterten es hinsichtlich der Operatoren des MTM zu dem abstrakten Kostenmodell. Die normalisierten Kosten ergeben sich mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$.

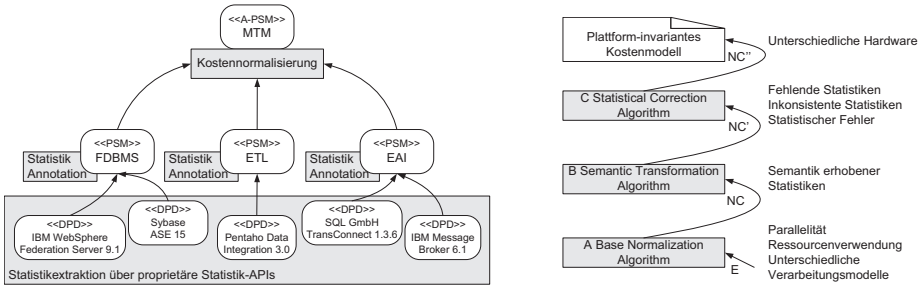
Um die Notwendigkeit von normalisierten Kosten für die systemübergreifende Kostenmodellierung zu veranschaulichen, nutzen wir das folgende illustrative Beispiel.

**Beispiel 2** Systemübergreifender Kostenvergleich: *Nehmen wir an, ein Integrationsprozess $P$ (einfache Sequenz von Operatoren: `Receive` $o_1$, `Translation` $o_2$ und `Invoke` $o_3$) wird sowohl von System $s1$ als auch von System $s2$ ausgeführt.*

Tabelle 2: Beispiel-Kostenvergleich auf der Basis normalisierter Kosten

| **Plan / System** | **Statistik (Aggregat)** | $o_1$ | $o_2$ | $o_3$ | $\sum$ |
|---|---|---|---|---|---|
| $P_{s1}$ | $t_e(o_l)$ | 10 ms | 150 ms | 70 ms | 230 ms |
| | $\|ds_{in}\|$ | - | 1573 | 1345 | |
| | $\|ds_{out}\|$ | 1573 | 1345 | 0 | |
| | $C(o_l)$ | 1573 | 2918 | 1345 | |
| | $NC(o_l)$ | 0.00636 ms | 0.05141 ms | 0.05204 ms | 0.10981 ms |
| $P_{s2}$ | $t_e(o_l)$ | 10 ms | 140 ms | 61 ms | 211 ms |
| | $\|ds_{in}\|$ | - | 1105 | 1017 | |
| | $\|ds_{out}\|$ | 1105 | 1017 | 0 | |
| | $C(o_l)$ | 1105 | 2122 | 1017 | |
| | $NC(o_l)$ | 0.00905 ms | 0.06598 ms | 0.05998 ms | 0.13501 ms |

*Tabelle 2 stellt beispielhaft Statistiken für diesen Fall dar. Sofern wir ausschließlich die Kosten $C(o_l)$ respektive die Verarbeitungszeiten $t_e(o_l)$ betrachten würden, kämen wir zu der Fehlannahme, dass $s2$ das optimale System darstellen würde. Indem wir jedoch die gewichteten Kosten $NC(o_l)$ mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$ in die Betrachtung einschließen (und damit die relative Verarbeitungszeit pro Tupel berechnen), können wir die eigentliche systemübergreifende Optimalitätsentscheidung zu Gunsten von $s1$ fällen.*




(a) Statistikextraktion (b) Kostennormalisierung

Abbildung 2: Überblick des Ablaufs der Statistikextraktion und Kostennormalisierung

Mit dieser Vorgehensweise kann also dem Problem der *unterschiedlichen Hardware* auf modellinhärente Weise begegnet werden. Weiterhin wird die Auswahlmöglichkeit über Systemgrenzen hinweg (bei disjunkten Prozessinstanzen) erst ermöglicht.

### 3.2 Erhebung von Verarbeitungsstatistiken

In diesem Abschnitt möchten wir, aufbauend auf dem eingeführten Kostenmodell, einen Überblick über die Erhebung von Verarbeitungsstatistiken und deren Normalisierung geben. In der Abbildung 2(a) wird die systemübergreifende Statistikextraktion und die Kostennormalisierung im Detail aufgezeigt.

Die Statistikerhebung erfolgt dabei in vier Schritten. Zunächst sind die Monitoring-Funktionalitäten der einzelnen Integrationssysteme zu nutzen, um Statistiken aufzunehmen. Diese können in einem zweiten Schritt über die jeweiligen proprietären Statistik-APIs extrahiert werden. Wir modellieren die erhobenen Basisstatistiken als eine Menge $E$ von Statistiktupeln $e_i$ mit $E = (e_1, e_2, \ldots, e_{n(m+1)})$. Dabei ergibt sich die Anzahl an Statistiktupeln $N$ mit $N = |E| = n(m+1)$ aus der Menge der verarbeiteten Prozessinstanzen $n$ und der Anzahl an Operatoren eines Prozesstyps $m$ (plus 1, da Statistik des Gesamtprozesses enthalten ist). Weiterhin modellieren wir ein Statistiktupel $e$ mit $e = (id, t0, t1, ds1, ds2, ds3)$, wobei $id \in \mathbb{N}$ einen Identifikator des Operators, $t0 \in \mathbb{R}$ die absolute Startzeit und $t1 \in \mathbb{R}$ die absolute Endzeit angibt. Darüber hinaus repräsentieren $ds1 \in \mathbb{N}$ und $ds2 \in \mathbb{N}$ die Kardinalitäten von Input-Datenmengen, während $ds3 \in \mathbb{N}$ die Kardinalität der Output-Datenmenge angibt. Dabei sind $id$, $t0$ und $t1$ obligatorisch für jeden Operator, während $ds1$, $ds2$ und $ds3$ in Abhängigkeit der Semantik der einzelnen Operatoren anzugeben sind. Auf systemtypspezifischer Ebene werden diese Basisstatistiken, in einem dritten Schritt, entsprechend den Systemtypen und Systemen mit Informationen annotiert. Bevor die Statistiken in dem vierten Schritt in das plattforminvariante Modell überführt werden können, muss die eigentliche Kostennormalisierung erfolgen.

Die Kostennormalisierung selbst (Abbildung 2(b)) erfolgt durch drei Algorithmen. Im ersten Algorithmus werden die Basisstatistiken normalisiert und aggregiert. Mit diesem Algorithmus können die drei Probleme *Parallelität*, *Ressourcenverwendung* und (mit Einschränkungen) *Unterschiedliche Verarbeitungsmodelle* behoben werden. Im Rahmen des zweiten Algorithmus werden die operatorspezifischen Statistiken von den plattformspezifischen Modellen in das MTM überführt. Dabei erfolgt die inverse Zuordnung zur Generie-

rung von Integrationsprozessen. Damit kann das Problem *Unterschiedliche Semantik der erhobenen Statistiken* beseitigt werden. Der dritte Algorithmus bereinigt die plattforminvarianten Statistiken, indem fehlende Werte geschätzt und korrigiert werden. Dies beseitigt die Probleme *Fehlende Statistiken*, *Inkonsistente Statistiken* und *Statistischer Fehler*.

# 4 Kostennormalisierung

Wie bereits angedeutet, umfasst die Kostennormalisierung drei aufeinander aufbauende Algorithmen, welche in diesem Abschnitt im Detail vorgestellt werden.

## 4.1 Parallelität, Ressourcenverwendung und Verarbeitungsmodelle

Der erste Algorithmus adressiert die Beseitigung von Parallelität, unterschiedliche Ressourcenverwendung und die Nutzung verschiedener Verarbeitungsmodelle. Das Grundkonzept soll zunächst an einem Beispiel veranschaulicht werden.

**Beispiel 3** Grundnormalisierung: *Die Parallelität von Prozessinstanzen, und damit auch von Operatorinstanzen (Operatoren einer Prozessinstanz), beeinflusst die effektive Ressourcenverwendung und somit letztendlich auch die Verarbeitungszeit. Die Kernidee ist die Einteilung der Zeitskala in Abschnitte (bestimmt durch den Beginn und das Ende von Instanzen) und die Normalisierung der Abschnittszeit entsprechend den parallel genutzten Ressourcen (und entsprechend der Anzahl paralleler Instanzen $num$). Die Abbildung 3 zeigt ein Beispiel dieser Grundnormalisierung.*

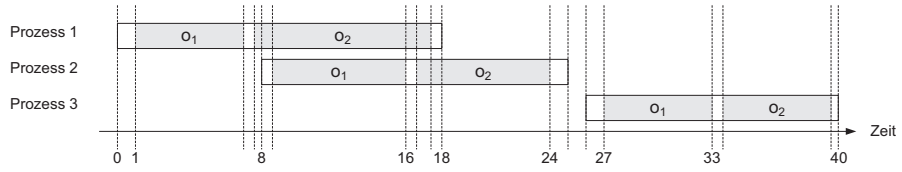



Abbildung 3: Beispiel einer Grundnormalisierung

*Entsprechend der Normalisierungsvorschrift* $t_e(o_l) = \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$ *mit* $Re(o) = \min(\frac{1}{num}, Ro(o, e.id))$ *für die einzelnen Abschnitte, vergleicht Tabelle 3 die erhobenen Statistiken mit den normalisierten Statistiken. Hierbei nehmen wir* $R_o(P_i) = 0.1$, $R_o(o_1) = 0.6$ *und* $R_o(o_2) = 0.85$ *als maximale Ressourcenverwendung an (kann festgelegt oder aus den erhobenen Statistiken berechnet werden). Damit ergeben sich die normalisierten Kosten der einzelnen Abschnitte. Beispielsweise errechnen sich die Kosten des Operators* $o_2$ *der Prozessinstanz* $P2$ *mit* $t_e(P_2o_2) = \frac{0.5}{0.85} \cdot 1.0 + \frac{0.5}{0.85} \cdot 0.5 + \frac{0.85}{0.85} \cdot 6.0 = 6.88$. *Im Kostenvergleich der parallel ausgeführten Prozessinstanzen* $P_1$ *und* $P_2$ *mit der Prozessinstanz* $P_3$ *(ohne Parallelität) ist der Einfluss der Kostennormalisierung zu erkennen.*

Tabelle 3: Beispiel-Kostenvergleich auf der Basis gewichteter Kosten

| | $t_e(o_1)$ | **Normalisiert** | $t_e(o_2)$ | **Normalisiert** | $t_e(P_i)$ | **Normalisiert** |
|---|---|---|---|---|---|---|
| $P_1$ | 6 ms | 6 ms | 10 ms | 6.09 ms | 18 ms | 14.09 ms |
| $P_2$ | 7.5 ms | 6.25 ms | 7.5 ms | 6.88 ms | 17 ms | 15.13 ms |
| $P_3$ | 6 ms | 6 ms | 6 ms | 6 ms | 14 ms | 14 ms |
| AVG | 6.5 ms | 6.08 ms | 7.73 ms | 6.32 ms | 16.33 ms | 14.41 ms |

Der Algorithmus 1 zeigt eine schematische Realisierung dieses Konzeptes. Im Wesentlichen besteht dieser Algorithmus aus den drei Teilen Sortierung, Normalisierung und Aggregation. Die Sortierung erfolgt zum Zwecke einer verbesserten Auswertbarkeit der Minima während der Normalisierung. Innerhalb des zweiten Schritts wird über die Menge an Statistiken iteriert und jede Prozessinstanz in ihre logischen Abschnitte untergliedert (kann durch $t0$ und $t1$ von Prozessinstanzen und Operatorinstanzen bestimmt sein). Die normalisierte Abschnittszeit ergibt sich mit $ne.t_e = \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$ aus der gemessenen Abschnittszeit gewichtet mit dem effektiven und maximalen Ressourcenverbrauch. Dabei

**Algorithm 1** Base Normalization

**Require:** base statistic $E$, process plan $P$

1: $NE \leftarrow \oslash, NC \leftarrow \oslash$
2: *// P*art 1: sort base statistics
3: $E \leftarrow$ sort $E[t0_i]$ asc
4: *// P*art 2: normalize base statistics
5: **for** $i = 1$ to $|E|$ **do** *// foreach instance*
6: $ne \leftarrow$ create $ne(e.id, 0, ed.s1, e.ds2, e.ds3)$
7: $t0_{part} \leftarrow e.t0$
8: $t1_{part} \leftarrow 0$
9: **while** `TRUE` **do** *// foreach instance part*
10: *// determine the next part border*
11: **if** $t1_{part} > 0$ **then**
12: $t0_{part} \leftarrow t1_{part}$
13: **end if**
14: $t1_{part} \leftarrow \min(\min(\pi_{t0}(\sigma_{t0>t0_{part} \wedge t0 \leq e.t1} E)), \min(\pi_{t1}(\sigma_{t1>t0_{part} \wedge t1 \leq e.t1} E)))$
15: **if** $t1_{part}$ is `NULL` **then** *// no next part border*
16: **break**
17: **end if**
18: *// compute the normalized part time*
19: $num \leftarrow count\,(\texttt{distinct}\ \ e \in [t0_{part}, t1_{part}]\ E)$
20: $R_e(o) \leftarrow \min(\frac{1}{num}, R_o(o, e.id))$
21: $ne.t_e \leftarrow ne.t_e + \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$
22: **end while**
23: $NE \leftarrow NE \cup ne$
24: **end for**
25: *// P*art 3: aggregate statistics
26: **for** $i = 1$ to $|NE|$ **do** *// foreach statistic tuple*
27: **if** $id \in NC$ **then**
28: $nc \leftarrow \sigma_{id=ne.id} NC$
29: **else**
30: $NC \leftarrow NC \cup nc$ where $nc \leftarrow$ create $nc(ne.id, 0, 0, 0, 0, 0)$
31: **end if**
32: $n \leftarrow nc.n, \quad nc.n \leftarrow n + 1, \quad nc.t_e \leftarrow \frac{n \cdot nc.t_e + ne.t_e}{n+1}$
33: $\forall k\ with\ 1 \leq k \leq 3 : nc.ds_k \leftarrow \frac{n \cdot nc.ds_k + ne.ds_k}{n+1}$
34: **end for**
35: **return** $NC$

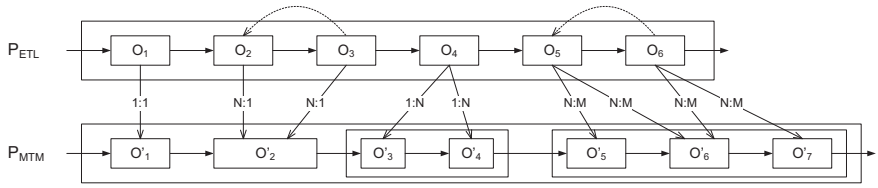



Abbildung 4: Beispiel-Transformation der Semantik von Operatoren

wird der effektive Ressourcenverbrauch mit $R_e(o) = \min(\frac{1}{num}, R_o(o, e.id))$ errechnet. Die Anzahl paralleler Abschnitte $num$ wird hierbei als Menge disjunkter Instanzen genutzt. Somit wird beispielsweise eine Operatorinstanz $o_1$, jedoch nicht deren umschließende Prozessinstanz $p_x$ betrachtet. Auf Grund des instanzbasierten Charakters nimmt eine Prozessinstanz bei ihrer Normalisierung stets die maximale Ressourcenverwendung $R_o(o)$ des aktuell ausgeführten Operators an. Im Rahmen der Aggregation wird das arithmetische Mittel über alle Instanzen mit gleicher $id$ (eindeutig pro Prozesstyp) berechnet.

**Theorem 1.** *Der Algorithmus* Base Normalization *zur Beseitigung des Einflusses von Parallelität, Ressourcenverwendung und unterschiedlichen Verarbeitungsmodellen weist eine quadratische Komplexität von $O(n^2m^2)$ auf.*

*Proof.* Zum Beweis von Theorem 1 sind die Einzelkomplexitäten der drei Algorithmusteile zu beweisen. Nehmen wir an, dass die Anzahl der Statistiktupel $N$ mit $N = n \cdot (m+1)$ bestimmt ist, wobei $n$ die Anzahl der Prozessinstanzen und $m$ die Anzahl der Operatoren angibt (plus 1, da für die gesamte Prozessinstanz). Teil 1 des Algorithmus sortiert die erhobenen Basisstatistiken. Es ist bekannt, dass die Sortierung von $E$ mit $O(N \log(N))$ möglich ist. Der zweite Teil des Algorithmus normalisiert die Basisstatistiken. Dies wird auf Ebene der Instanzabschnitte realisiert. Dabei kann die $i$-te Instanz im schlimmsten Fall $2i-1$ Abschnitte aufweisen (im Fall, dass jede $i$-te Instanz die $i+1$-te Instanz vollständig überdeckt). Insgesamt sind somit also maximal

$$\sum_{i=1}^{n(m+1)} (2i-1) = (2 \sum_{i=1}^{n(m+1)} i) - n(m+1) = n^2m^2 + 2n^2m + n^2$$

Abschnitte zu evaluieren. Auf Grund der sortierten Reihenfolge der Statistiken, nehmen wir konstante Komplexität mit $O(1)$ für die Berechnung der folgenden Abschnittsgrenzen an. Somit weist der zweite Teil des Algorithmus eine quadratische Komplexität mit $O(n^2m^2) = O(n^2m^2 + 2n^2m + n^2) \cdot O(1)$ auf. Der dritte Teil des Algorithmus realisiert die Aggregation der Statistiken ($n \cdot (m+1)$) mit linearer Komplexität von $O(n \cdot m)$). Zusammenfassend weist der Algorithmus entsprechend des Theorems eine quadratische Komplexität mit $O(n^2m^2) = O(nm \log(nm)) + O(n^2m^2) + O(nm)$ auf.

### 4.2 Transformation der Semantik von Operatoren

Aufbauend auf dem Ergebnis der Grundnormalisierung ist im zweiten Schritt die Normalisierung der Semantik von Operatorstatistiken – invers zur Generierung – zu realisieren. Das folgende Beispiel veranschaulicht dieses Problem.

**Beispiel 4** Semantiktransformation: *In Abbildung 4 ist (auf schematische Weise) das Problem der Überführung von Statistiken aufgezeigt. Hier wurden Statistiken für einen ETL-Job erhoben und müssen nun in die plattforminvariante Form überführt werden. Dieses*

*Problem adressiert unterschiedliche Operatoren beider Modellebenen. Sofern eine einfache `1:1` Abbildung besteht, können die Statistiken direkt übernommen werden. Auch eine `N:1`-Abbildung ist unproblematisch, da hier lediglich die Einzelstatistiken der N-Seite zu aggregieren sind. Bei `1:N`- oder `N:M`-Abbildungen ist jedoch der komplexe Operator `Process` zur Repräsentation der Statistiken zu nutzen. Im Beispiel würden die Statistiken des Operators $o_4$ also zwar übernommen, jedoch könnten keine Rückschlüsse auf die Einzeloperatoren $o'_3$ und $o'_4$ gezogen werden.*

Die Realisierung der Semantiktransformation erfolgt mit Hilfe des Algorithmus *Semantik Transformation* (Algorithmus 2). Zunächst werden die Operatorsequenzen des plattformspezifischen Prozessplans $P$ und des plattformunabhängigen Prozessplans $P'$ ermittelt. Anschließend arbeitet der Algorithmus linear über alle Operatoren $o_i \in P$. Sofern $o_i$ ein komplexer Operatortyp ist, wird der Algorithmus rekursiv aufgerufen. Weiterhin wird

**Algorithm 2** Semantic Transformation

**Require:** normalized stats $NC$, node $P$, node $P'$ *// a process plan is a specialized node*
1: $NC' \leftarrow \emptyset, \quad o \leftarrow o \in P, \quad o' \leftarrow o' \in P'$
2: **for** $i = 1$ to $|o|$ **do** *// foreach operator in o*
3: **if** $o_i \in$ `Process`, `Switch`, `Fork`, `Iteration` **then**
4: $NC' \leftarrow NC' \cup$ execute *Semantic Transformation*$(NC, o_i, P')$
5: **end if**
6: $struct \leftarrow$ execute *evalMappingRule*$(o_i, o')$
7: **if** $struct \in$ `1:1` **then**
8: *// copy operator statistics*
9: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(\pi_{struct.id,n,t_e,ds1,ds2,ds3}(\sigma_{id=id(o_i)}NC))$
10: **else if** $struct \in$ `1:N` **then**
11: *// copy operator statistics to parent node*
12: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(\pi_{struct.id,n,t_e,ds1,ds2,ds3}(\sigma_{id=id(o_i)}NC))$
13: **for** $j = 1$ to $|o' \in struct|$ **do** *// foreach operator $o'_j \in struct$*
14: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(id(o'_j), -1, -1, -1, -1, -1)$
15: **end for**
16: **else if** $struct \in$ `N:1` **then**
17: *// sum up and copy statistics*
18: $t_e \leftarrow \pi_{\sum_{j=1 \wedge o_j \in struct}^{i}}(\sigma_{id=id(o'_j)}NC)$
19: $n, ds1, ds2, ds3 \leftarrow$ operator specific copy of statistics
20: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(struct.id, n, t_e, ds1, ds2, ds3)$
21: **else if** $struct \in$ `N:M` **then**
22: *// sum up statistics and copy to parent node*
23: $t_e \leftarrow \sum_{j=1 \wedge o_j \in struct}^{i} \pi_{t_e}(\sigma_{id=id(o_j)}NC)$
24: $n, ds1, ds2, ds3 \leftarrow$ operator specific copy of statistics
25: **for** $j = 1$ to $|struct|$ **do** *// foreach operator in struct*
26: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(id(o_j), -1, -1, -1, -1, -1)$
27: **end for**
28: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(struct.id, n, t_e, ds1, ds2, ds3)$
29: **end if**
30: **end for**
31: **return** $NC'$

im Wesentlichen die inverse Abbildungsregel zwischen dem Operator $o_i$ und der Operatormenge $o'$ bestimmt. Die resultierende Datenstruktur wird nun hinsichtlich der vier konzeptuellen Abbildungsmöglichkeiten (`1:1`, `1:N`, `N:1` und `N:M`) unterschieden. Im Falle der `1:1`-Abbildung erfolgt eine einfache Kopie der Statistiken. Schwieriger stellt sich die `1:N`-Abbildung dar. Hier werden die Statistiken für den kompletten Teilgraphen (welcher mehrere Operatoren enthält) abgebildet, während für die Einzeloperatoren fehlende Statistiken vermerkt werden, so dass diese im nachfolgenden Algorithmus zu schätzen sind. Bei der `N:1` Abbildung kann hingegen einfach die Summe über die Verarbeitungszeiten gebildet und die Start- und Endkardinalitäten können schlicht übernommen werden. Die `N:M`-Abbildung ist eine Kombination aus `1:N`- und `N:1`-Abbildungen. So werden die Statistiken zunächst (ähnlich zu `N:1`) zusammengefasst und anschließend (ähnlich zu `1:N`) auf einen Teilgraphen abgebildet.

**Theorem 2.** *Der Algorithmus* Semantic Transformation *zur Beseitigung des Einflusses der Semantik plattformspezifischer Operatoren weist eine quadratische Komplexität von* $O(m^2 + m')$ *auf.*

*Proof.* Generell arbeitet der Algorithmus linear über alle Operatoren des plattformspezifischen Prozessplans. Im schlimmsten Fall (N:1 oder N:M) müssen alle Statistiken des plattformspezifischen Prozesses aggregiert werden. Hierfür ergeben sich

$$\sum_{i=1}^{m}(m-i)+m' = m(m+1) - \frac{(m+1)^2 - m + 1}{2} + m' = \frac{m^2 - m}{2} + m'$$

notwendige Operatorevaluierungen. Folglich weist der Algorithmus eine quadratische Komplexität mit $O(m^2 + m') = O(\frac{m^2-m}{2} + m')$ auf.

### 4.3 Statistische Korrektur

Anschließend erfolgt die statistische Korrektur im Sinne des Erkennens von Inkonsistenzen zwischen Statistiken sowie deren Vervollständigung. Dabei können Inkonsistenzen ausschließlich durch die Integrationssysteme verursacht worden sein, während fehlende Statistiken durch die Integrationssysteme und durch die zuvor vorgenommene Transformation der Semantik (`1:N`- und `N:M`-Abbildungen) begründet sein können.

**Beispiel 5** Statistikkorrektur: *Nehmen wir einen Prozess $P$ an, welcher eine Sequenz von vier Operatoren $o_1$ bis $o_4$ umfasst. Für $o_1$ und $o_4$ liegen jeweils Statistikvektoren $NC(o_i)$ vor, worin Input-Datenmengen $|ds_{in}|$, Output-Datenmengen $|ds_{out}|$ sowie die effektive Verarbeitungszeit $t_e(o_i)$ enthalten sind. Für die Operatoren $o_2$ und $o_3$ existieren keine Einzelstatistiken sondern lediglich Aggregate über den Teilprozess sp ($o_2$, $o_3$). Um die fehlenden Statistiken zu berechnen, können wir zunächst $|ds_{in}(o_2)| = |ds_{in}(sp)|$ sowie $|ds_{out}(o_3)| = |ds_{out}(sp)|$ setzen. Nun besteht das Problem der Schätzung der Kardinalitäten der Zwischenresultate. Hierzu wenden wir mit $|ds_{out}(o_2)| = |ds_{in}(sp)| -$*

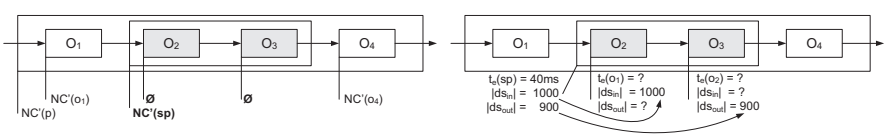



Abbildung 5: Beispielkorrektur der normalisierten Statistiken

(2) $\frac{|ds_{in}(sp)|-|ds_{out}(sp)|}{2}$ *und* $|ds_{in}(o_3)| = |ds_{in}(sp)| - (3-1)\frac{|ds_{in}(sp)|-|ds_{out}(sp)|}{2}$ *die lineare Extrapolation an und kommen zum Ergebnis* $|ds_{out}(o_2)| = |ds_{in}(o_3)| = 950$*. Nun können wir mit Hilfe der abstrakten Kosten* $C(o_i)$ *und der effektiven Verarbeitungszeit* $t_e(sp)$ *die fehlenden Verarbeitungszeiten mit* $t_e(o_i) = t_e(sp) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$ *berechnen. Sofern wir von den Operatoren* `Translation` *und* `Invoke` *ausgehen, ergibt sich hier:* $t_e(o_2) = 40ms \cdot \frac{1950}{3800} = 20.53ms$ *und* $t_e(o_3) = 40ms \cdot \frac{1850}{3800} = 19.47ms$*.*

Der Algorithmus 3 (*Statistical Correction*) zeigt diese statistische Korrektur. Zunächst wird versucht, Inkonsistenzen zwischen Statistiken aufzudecken. Hierzu wird für alle Operatoren die Bedingung $|ds_{out}(o_i)| \neq |ds_{in}(o_{i+1})|$ überprüft. Wir können hier Gleichheit fordern, da wir zur Aggregation das arithmetische Mittel und nicht die exponentielle

**Algorithm 3** Statistical Correction

**Require:** operator sequence $o$, total costs $NC(p)$, operator costs $NC(o)$
execute *assignCosts*$(o, NC(p), NC(o))$ *//assign costs to nodes with* $id(o) = nc.id$
$flag \leftarrow$ **false**, $sum \leftarrow 0$
// check for inconsistencies between statistics
**for** $i = 1$ to $|o| - 1$ **do** *// foreach operator in o*
  **if** $|ds_{out}(o_i)| \neq |ds_{in}(o_{i+1})|$ **then**
    $flag \leftarrow$ **true**; **break**
  **end if**
  $sum \leftarrow sum + t_e(o_i)$
**end for**
$sum \leftarrow sum + t_e(o_m)$
**if** $|ds_{in}(P)| \neq |ds_{in}(o_1)|$ `OR` $|ds_{out}(P)| \neq |ds_{out}(o_m)|$ `OR` $sum > t_e(P)$ **then**
  $flag \leftarrow$ **true**
**end if**
**if** $flag$ **then**
  $\forall i$ : clear statistics $NC(o_i) \leftarrow -1$
**end if**
// check for missing statistics
**if** $flag$ `OR` $\forall i : NC(o_i) = \oslash$ **then**
  // recompute missing statistics (linear extrapolation)
  **for** $i = 1$ to $|o|$ **do** *// foreach operator in o*
    $ds_{in}(o_i) \leftarrow ds_{in}(P) - (i-1)\frac{ds_{in}(P)-ds_{out}(P)}{|o|}$
    $ds_{out}(o_i) \leftarrow ds_{in}(P) - (i)\frac{ds_{in}(P)-ds_{out}(P)}{|o|}$
    $C(o_i) \leftarrow C(ds_{in}(o_i), ds_{out}(o_i))$
  **end for**
  **for** $i = 1$ to $|o|$ **do** *// foreach operator in o*
    $t_e(o_i) \leftarrow t_e(P) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$
    **if** $o_i \in$ `Process`, `Switch`, `Fork`, `Iteration` **then**
      execute *Statistical Correction* for $o_i$
    **end if**
  **end for**
**end if**
**return** $NC'$

Glättung verwenden. Sofern in diesem Schritt Ungleichheit festgestellt wurde, die Summe der Verarbeitungszeiten nicht mit dem hierarchisch übergeordneten Knoten übereinstimmt oder die Input- und Output-Kardinalitäten des Prozesses nicht mit dem ersten respektive letzten Operator übereinstimmen, werden die Statistiken entsprechend gelöscht. Im zweiten Hauptschritt des Algorithmus werden die fehlenden Statistiken mit Hilfe linearer Extrapolation geschätzt. Hierzu sind in einem ersten Schleifendurchlauf die Input- und Output-Kardinalitäten $ds_{in}(o_i)$ und $ds_{out}(o_i)$ in Abhängigkeit der Statistiken des übergeordneten Knotens zu schätzen, sowie die abstrakten Kosten $C(o_i)$ (siehe Tabelle 1) zu summieren. In einem zweiten Schleifendurchlauf (die Summe wird hier benötigt) wird dann letztendlich die Verarbeitungszeit $t_e(o_i)$ mit $t_e(o_i) = t_e(P) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$ geschätzt. Sofern ein komplexer Operatortyp vorliegt, wird auch hier rekursiv der Algorithmus für Teilgraphen ausgeführt. Dies erfolgt aber bewusst nachdem die Statistiken für diesen Knoten geschätzt wurden. Somit kann die Schätzung des Teilgraphs auf Schätzungen des Elternknotens beruhen.

**Theorem 3.** *Der Algorithmus* Statstical Correction *zur Korrektur und Vervollständigung von Statistiken weist eine lineare Komplexität von $O(m)$ auf.*

*Proof.* Im schlimmsten Falle (fehlende Statistiken für den letzten Operator) werden vier Schleifendurchläufe über die Menge an Operatoren zur statistischen Berichtigung benötigt. Somit ergibt sich eine lineare Komplexität von $O(m) = O(4m)$.

## 5 Experimentelle Evaluierung

Aufbauend auf den theoretischen und konzeptionellen Ergebnissen dieser Arbeit wurde die praktische Anwendbarkeit des Ansatzes an Hand von Experimenten evaluiert. Hierzu realisierten wir die Algorithmen im Rahmen unserer `WFPE` (*Workflow Process Engine*, in Java 1.6 implementiert). Im Wesentlichen evaluieren wir funktionale (Qualität der Normalisierung) und nichtfunktionale (Performanz) Aspekte der Kostennormalisierung. Die Experimente wurden auf einem standard Balde (OS Suse Linux) mit zwei Prozessoren (jeder davon ein Dual Core AMD Opteron Processor 270 mit 1994 MHz) und 8.9 GB RAM ausgeführt. Zur Erhebung von Statistikinformationen führten wir Integrationsprozesse des DIPBench [BHLW08a] (und Derivate dieser Prozesse) mit der `WFPE` und unterschiedlichen Workload-Charakteristika aus. Im Rahmen dieser experimentellen Evaluation analysierten wir systematisch die folgenden Einflussfaktoren: Anzahl der Prozessinstanzen $n$, Kardinalität der Input-Daten $d = |ds_{in}(P)|$, Verarbeitungsmodell $v$, Anzahl der Operatoren $m$ sowie die Menge der Statistik-Tupel $N$.

### 5.1 Funktionale Evaluierung der Kostennormalisierung

Zum Zwecke der Evaluierung der Qualität der Kostennormalisierung nutzen wir zwei unterschiedliche Prozesstypen. Der Prozesstyp $P13$ wurde innerhalb des DIPBench spezifiziert und adressiert die Extraktion von Bestellungen und Bestellpositionen aus einer konsolidierten Datenbank und das Laden dieser Daten in ein Data-Warehouse. Dabei impliziert der Skalierungsfaktor Datenmenge $d$, mit $d = 1.0$, $70.000$ Bestellungen und $350.000$ Bestellpositionen. Der Prozesstyp $P_{m5}$ besteht aus einer Sequenz von fünf Operatoren

Tabelle 4: Workload-Skalierung im Rahmen der funktionalen Evaluierung

| **Prozessinstanz $n_i$** | **Skalierung $p$ in Experiment 1** | **Skalierung $d$ in Experiment 2** |
|---|---|---|
| $0 < n_i \leq 30.000$ | $p = 1$ | $d = 0.0001$ |
| $30.000 < n_i \leq 60.000$ | $p = 2$ | $d = 0.0002$ |
| $60.000 < n_i \leq 90.000$ | $p = 3$ | $d = 0.0003$ |
| $90.000 < n_i \leq 120.000$ | $p = 4$ | $d = 0.0004$ |
| $120.000 < n_i \leq 150.000$ | $p = 3$ | $d = 0.0003$ |
| $150.000 < n_i \leq 180.000$ | $p = 2$ | $d = 0.0002$ |
| $180.000 < n_i \leq 210.000$ | $p = 1$ | $d = 0.0001$ |

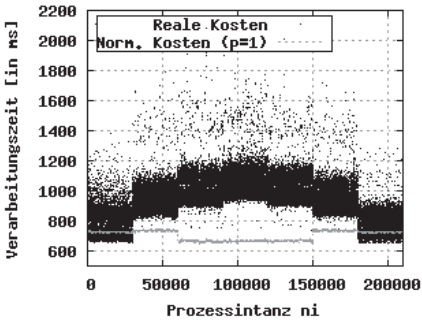
($m = 5$), welche (1) eine Anfrage vorbereiten, (2) ein $100kb$ großes XML-Dokument laden, (3) ein Schema-Mapping des Dokumentes durchführen, (4) eine weitere Anfrage vorbereiten und letztendlich (5) das Ergebnis versenden. Zur Skalierung der Operatoranzahl $m$ werden diese Operatoren periodisch wiederholt.

Die Abbildung 6 zeigt die Qualität der Kostennormalisierung für variierende Workload-Charakteristika (Grad der Parallelität $p$ / Datenmenge $d$) sowie die Qualität im Hinblick auf unterschiedliche Verarbeitungsmodelle.

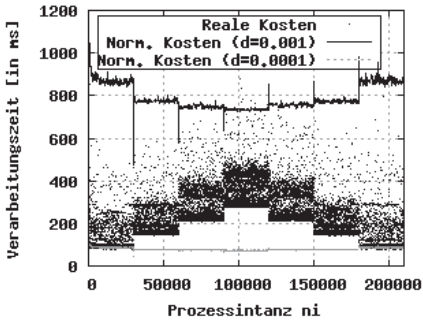
**Experiment 1:** In diesem Experiment (Abbildung 6(a)) wurden $n = 210.000$ Prozessinstanzen von $P13$ mit einer Datenmenge von $d = 0.001$ mit der `WFPE` ausgeführt. Hierbei wurde der Grad der Parallelität $p$ – im Sinne der gleichzeitigen Initiierung von $p$ Prozessinstanzen – variiert (siehe Tabelle 4, Spalte 2). Anschließend wurden die Erhobenen Statistikdaten mit Hilfe der Normalisierungsalgorithmen auf $p = 1$ normalisiert (jeweils in einem Intervall von 210 Instanzen). Dabei war die maximale Ressourcenauslastung $R_o = 0.75$. Es ist zu erkennen, dass die Normalisierung für $p = 2$ sehr gute Ergebnisse liefert, während es für $p > 2$ zu einer Unterschätzung kommt. Dies ist mit der sehr guten Skalierung der Hardware bei steigender Parallelität zu erklären.

**Experiment 2:** Weiterhin wurden in einem zweiten Experiment (Abbildung 6(b)) abermals $n = 210.000$ Prozessinstanzen von $P13$ (mit konstantem $p = 1$) ausgeführt. Dabei variierten wir die Datenmenge $d$ entsprechend Tabelle 4, Spalte 3. Die Normalisierung (auf $d = 0.001$ und $d = 0.0001$) erfolgte ähnlich zu Experiment 1. Offensichtlich ist eine hohe Qualität der Normalisierung zu erkennen, wobei lediglich geringe Ausreißer an den Grenzen des Workload-Wechsels auftraten.

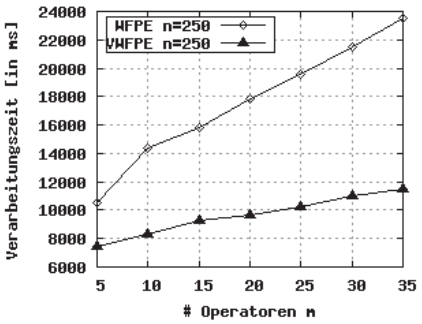
**Experiment 3:** Im dritten Experiment hinterfragten wir die Normalisierung unterschiedlicher Verarbeitungsmodelle. Während die `WFPE` standardmäßig ein instanzbasiertes Verarbeitungsmodell (jede Instanz ein Thread, kein Pipelining) nutzt, ermöglicht sie auch die vektorisierte Prozessausführung `VWFPE` (Verarbeitungsmodell *pipes and filter*), wobei jeder Operator als Thread ausgeführt wird und Pipelining von Nachrichten zwischen Operatoren erfolgt. Die Abbildung 6(c) zeigt den Durchsatzvergleich (Ausführung von 250 Prozessinstanzen) der beiden Ausführungsmodelle für den Prozesstyp $P_{mX}$ (mit $m = X$), wobei wir $X$ von 5 bis 35 Operatoren skalierten und das Experiment 20 mal wiederholten. Der bessere Durchsatz wird durch die höhere Parallelität erzielt. Nun war zu untersuchen, ob die Normalisierung auch über Verarbeitungsmodelle hinweg gute Ergebnisse liefert. Hierbei erwarteten wir, dass die Einzelverarbeitungszeiten der `VWFPE` höher als die der `WFPE` wären und die Normalisierung der `VWFPE`-Zeiten ähnlich der `WFPE`-Zeiten wären. Die Abbildung 6(d) zeigt das Resultat dieses Experiments. Offensichtlich kann die



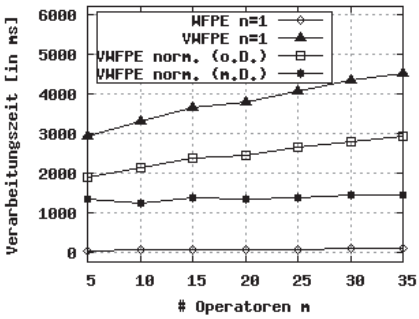

(a) Kostennormalisierung Parallelität $p$




(b) Kostennormalisierung Datenmenge $d$




(c) Durchsatzvergleich Verarbeitungsmodelle




(d) Kostennormalisierung Verarbeitungsmodell

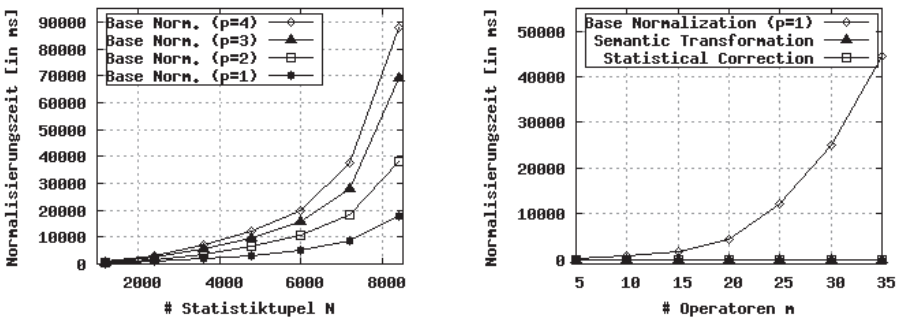
Abbildung 6: Qualität der Kostennormalisierung bei unterschiedlichen Einflussfaktoren

Normalisierung nicht das Problem unterschiedlicher Verarbeitungsmodelle beheben und neigt hier zu extremen Überschätzungen. Dieses negative Ergebnis ist auch plausibel zu erläutern. So funktioniert die Normalisierung der Verarbeitungszeiten einzelner Operatoren auch für unterschiedliche Verarbeitungsmodelle. Die Normalisierung von Prozessplänen ist aber viel komplexer, da hier die Verarbeitungszeit extrem vom Füllstand der Warteschlangen (jede Kante im Operatorgraph wird durch eine Queue repräsentiert) zwischen den Operatoren abhängt. Folglich kann es bei hohem Füllstand zu sehr langen Gesamtzeiten und damit zu starken Überschätzungen kommen. Als Schlussfolgerung dieses Ergebnisses bleibt zu erwähnen, dass die Normalisierung unterschiedlicher Verarbeitungsmodelle eine offene Forschungsfrage darstellt, der möglicherweise mit einer durchsatzbasierten oder operatorbasierten Normalisierung begegnet werden kann.

### 5.2 Nichtfunktionale Evaluierung der Kostennormalisierung

Neben dem Aspekt der Qualität der Kostennormalisierung stellt sich im Kontext der kontinuierlichen Kostennormalisierung natürlich auch die Frage der Effizienz der Algorithmen. Diese wird mit der nichtfunktionalen Evaluierung untersucht.

**Experiment 4:** Im vierten Experiment wird zunächst der Algorithmus *Base Normalization* hinsichtlich der Skalierung mit zunehmender Anzahl an Statistiktupeln $N$ (mit $N = m \cdot n$)




(a) Performance unter Skalierung $n$ (b) Performance unter Skalierung $m$

Abbildung 7: Effizienz der Kostennormalisierung

evaluiert. Dabei wurden die Statistikdaten aus Experiment 1 wiederverwendet. Da die Anzahl der parallelen Instanzen einen Einfluss auf die Laufzeit des Algorithmus hat, haben wir den Grad der Parallelität $p$ in das Experiment eingeschlossen. Die Abbildung 7(a) visualisiert das Ergebnis und bestätigt die quadratische Komplexität von $O(m^2n^2)$. Weiterhin zeigt das Ergebnis eine steigende Laufzeit mit steigendem Grad der Parallelität, da hier mehr Teilabschnitte zu normalisieren sind. Die beiden anderen Algorithmen verarbeiten aggregierte Daten und hängen damit nur von $m$ (linear von $N$) ab.

**Experiment 5:** In einem fünften und letzten Experiment verwendeten wir die Statistikdaten aus Experiment 3 wieder, um den Einfluss der Anzahl an Operatoren $m$ auf die Laufzeit der Algorithmen zu analysieren. Die Abbildung 7(b) zeigt das Ergebnis, wobei klar zu erkennen ist, dass der Algorithmus *Base Normalization* die Gesamtlaufzeit zweifelsohne dominiert. Dabei ist wiederum die quadratische Komplexität dieses Algorithmus zu erkennen. Da die beiden anderen Algorithmen ausschließlich auf aggregierten Daten $N'$ mit $N' \ll N$ arbeiten, hängen diese nur von $m$ ab. Die Laufzeit dieser Algorithmen steigt weiterhin linear mit der Anzahl an Operatoren $m$.

## 6 Verwandte Arbeiten

Im Rahmen verwandter Arbeiten wollen wir zum einen Anwendungsgebiete aufzeigen, in welchen die systemübergreifende Kostennormalisierung zum Einsatz kommen kann. Darüber hinaus wollen wir angrenzende Arbeiten darstellen, welche ebenfalls die Notwendigkeit der Kostennormalisierung respektive die Normalisierung der Arbeitslast zum Gegenstand haben. Generell lässt sich hierbei einschätzen, dass (1) Vorarbeiten (unsere eigenen eingeschlossen) im Bereich der Generierung von Integrationsprozessen ein Anwendungsgebiet darstellt und, dass (2) bislang noch kein Ansatz existiert der die *systemübergreifende* Kostennormalisierung adressiert.

*Generierung und Deployment von Integrationsprozessen.* Im Kontext der Generierung und des Deployments von Integrationsprozessen sind im Wesentlichen drei Ansätze hervorzuheben. Das Orchid-Projekt [DHW+08] befasst sich mit der Generierung von ETL-Jobs auf Grundlage von deklarativen Mapping-Spezifikationen. Dabei wird hier ein sogenanntes Operator Hub Model (OHM) verwendet, um die Verarbeitungssemantik von ETL-

Jobs plattformunabhängig abzubilden. Das Orchid-Projekt beschränkt sich momentan ausschließlich auf die Generierung von ETL-Prozessen für IBM-ETL-Werkzeuge. Konzeptionell scheint die Erweiterung auf den gesamten (herstellerunabhängigen) ETL-Bereich problemlos möglich. Im Gegensatz dazu, adressiert eine andere Arbeit [AN08] die Verbindung der Generierung von ETL-Prozessen und des Model Managements, indem plattformunabhängige Operatoren für das Deployment von ETL-Prozessen vorgestellt wurden. Unser GCIP Framework (Generation of Complex Integration Processes) hingegen adressiert die Modellierung von plattformunabhängigen Integrationsprozessen [BWHL08], die Generierung für *unterschiedliche* Integrationssystemtypen (wie zum Beispiel für FDBMS, EAI und ETL) sowie die Anwendung von Optimierungstechniken [BWHL08,BHLW08b] im Rahmen der modellgetriebenen Generierung. Die Gemeinsamkeit der drei Projekte liegt in der Auswahlmöglichkeit des optimalen Integrationssystems (Bestimmung des Ziels der Generierung). Als Grundlage für eine derartige Optimalitätsentscheidung muss es eine systemübergreifende Kostennormalisierung geben. Folglich stellen alle drei genannten Projekte mögliche Anwendungsgebiete des vorgestellten Ansatzes dar.

*Kostennormalisierung in verteilten Datenbanksystemen.* Während die Statistiknormalisierung in der Mathematik und des Operations Research ein oft verwendetes Mittel zur Herstellung der Vergleichbarkeit ist (beispielsweise im Rahmen der Data Envelopment Analysis [SCPP07]), existieren im Bereich der Normalisierung von Verarbeitungsstatistiken sehr wenige Arbeiten auf angrenzenden Gebieten. Ähnliche Probleme existieren im Bereich der Anfrageverarbeitung in verteilten Datenbanksystemen. Im Wesentlichen sind an dieser Stelle zwei Gruppen von Ansätzen zur Kostennormalisierung zu unterscheiden. Die Gruppe der *hierarchischen Ansätze* versucht eine Hierarchie von Kostenbewertungen über die Quellsysteme in statischer Weise aufzubauen. So werden systemübergreifend logische Statistiken zum Vergleich verwendet, während die einzelnen Wrapper und Quellsysteme für die Überführung der physischen Statistiken in logische Statistiken verantwortlich sind. Ein interessanter Ansatz dieser Kategorie [NGT98] definiert einen sogenannten hierarchic cost formula tree, wobei die Kostenbewertung einer Anfrage in unterschiedliche Aspekte untergliedert wird. Ein anderes Beispiel für die Kategorie ist das Garlic-Projekt [ROH99,JSHL02] wobei relationale und nicht-relationale Datenquellen Statistikinformationen bereitstellen und nur bei Fehlen dieser Statistiken Schätzungen an zentraler Stelle vorzunehmen sind. Im Gegensatz zu diesen statischen Ansätzen adressiert die Gruppe der *dynamischen Ansätze* die adaptive Kostenbewertung der einzelnen Datenquellen, wobei die Kostenmodelle dynamisch zur Laufzeit, basierend auf dem Vergleich von geschätzten und realen Verarbeitungsstatistiken, verändert werden. Als Beispiele sind hier die Realisierung von verteiltem Caching [SW97], die Kalibrierung der Koeffizienten eines Kostenmodells für IRO-DB [GST96] und die evolutionäre Anpassung des Anfragekostenmodells [RZL04] zu nennen. Diese Ansätze der verteilten Datenbanksysteme können nicht alle identifizierten Probleme der systemübergreifenden Kostennormalisierung beheben, da hier lediglich relative Kosten innerhalb der einzelnen Systeme miteinander verglichen werden und damit keine systemübergreifende Vergleichbarkeit von Statistiken hergestellt wird.

## 7 Zusammenfassung und Ausblick

Ausgehend von der Generierungsmöglichkeit von Integrationsprozessen für unterschiedliche Integrationssysteme wurden in dieser Arbeit die wesentlichen Probleme der sys-

temübergreifenden Kostennormalisierung identifiziert. Im Detail wurde eine generelle Methodik zur Extraktion und Normalisierung von Statistiken aus konkreten Integrationssystemen aufgezeigt. Hierzu diskutierten wir zunächst ein plattforminvariantes Kostenmodell und stellten drei aufeinander aufbauende Algorithmen zur Kostennormalisierung vor. Die experimentelle Evaluierung hat gezeigt, dass die Normalisierung (mit Einschränkungen) qualitativ gute Ergebnisse liefert und damit die Vergleichbarkeit von Integrationssystemen sichert (Annahme 3). Dies ist in unterschiedlichen Anwendungsgebieten einsetzbar, wobei jedoch noch offene Probleme existieren. So sind die Minimierung von statistischen Fehlern im Rahmen der Kostenvervollständigung, die Einbeziehung von Hintergrundlast andere Systeme, inkrementelle Normalisierungsalgorithmen sowie unterschiedliche Verarbeitungsmodelle genauer zu untersuchen.

## Literatur


[AN08] Alexander Albrecht and Felix Naumann. Managing ETL Processes. In *NTII*, 2008.

[BBW+07] Matthias Böhm, Jürgen Bittner, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. Ein Nachrichtentransformationsmodell für komplexe Transformationsprozesse in datenzentrischen Anwendungsszenarien. In *BTW*, 2007.

[BF07] Susanne Busse and Johann Christoph Freytag. Entwurf von Informationsintegrationssystemen auf der Basis der Merkmalsmodellierung. In *BTW*, 2007.

[BHLW08a] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. DIPBench Toolsuite: A Framework for Benchmarking Integration Systems. In *ICDE*, 2008.

[BHLW08b] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. Workload-Based Optimization of Integration Processes. In *CIKM*, 2008.

[BWHL08] Matthias Böhm, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. Model-Driven Generation and Optimization of Complex Integration Processes. In *ICEIS (1)*, 2008.

[DHW+08] Stefan Dessloch, Mauricio A. Hernández, Ryan Wisnesky, Ahmed Radwan, and Jindan Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, 2008.

[GST96] Georges Gardarin, Fei Sha, and Z.-H. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System. In *VLDB*, 1996.

[JSHL02] Vanja Josifovski, Peter M. Schwarz, Laura M. Haas, and Eileen Tien Lin. Garlic: a new flavor of federated query processing for DB2. In *SIGMOD*, 2002.

[Mak07] Mazeyar E. Makoui. *Anfrageoptimierung in objektrelationalen Datenbanken durch kostenbedingte Termersetzungen*. PhD thesis, Universität Hannover, 2007.

[NGT98] Hubert Naacke, Georges Gardarin, and Anthony Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *ICDE*, 1998.

[RH05] Naveen Reddy and Jayant R. Haritsa. Analyzing Plan Diagrams of Database Query Optimizers. In *VLDB*, 2005.

[ROH99] Mary Tork Roth, Fatma Ozcan, and Laura M. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *VLDB*, 1999.

[RZL04] Amira Rahal, Qiang Zhu, and Per-Åke Larson. Evolutionary techniques for updating query cost models in a dynamic multidatabase environment. *VLDB J.*, 13, 2004.

[SCPP07] Hyeonju Seol, Jeewon Choi, Gwangman Park, and Yongtae Park. A framework for benchmarking service process using data envelopment analysis and decision tree. *Expert Syst. Appl.*, 32, 2007.

[Sto02] Michael Stonebraker. Too Much Middleware. *SIGMOD Record*, 31(1), 2002.

[SW97] Markus Sinnwell and Gerhard Weikum. A Cost-Model-Based Online Method for Distributed Caching. In *ICDE*, 1997.

# Web-Prospector – An Automatic, Site-Wide Wrapper Induction Approach for Scientific Deep-Web Databases

Saqib Mir[1], Steffen Staab[2], Isabel Rojas[3]
[1, 3] EML Research
Schloss-Wolfsbrunnenweg 33
D-69118 Heidelberg, Germany
[1, 2] Institute for Computer Science,
University of Koblenz-Landau,
D-56016 Koblenz, Germany
{saqib.mir, isabel.rojas}@eml-r.villa-bosch.de, staab@uni-koblenz.de

**Abstract:** Wrapper induction techniques traditionally focus on learning wrappers based on examples from one *class* of Web pages, i.e. from Web pages that are all similar in structure and content. Thereby, traditional wrapper induction targets the understanding of Web pages generated from a database using the same generation template as observed in the example set. Applying such techniques to Web sites generated from biological databases, however, we found that there is a need for wrapping of structurally diverse web pages from *multiple classes* making the problem more challenging. Furthermore, we observed that such scientific web sites do not just provide mere data, but they also tend to provide schema information in terms of data labels – giving further cues for solving the web site wrapping task. In this paper we present a novel approach to automatic information extraction from whole Web sites that considers the novel challenge and takes advantage of the additional clues commonly available in scientific deep Web databases. The solution consists of a sequence of steps: 1. classification of similar Web pages into classes, 2. discovery of these classes and 3. wrapper induction for each class. Our approach thus allows us to perform unsupervised information retrieval from across an entire Web site. We test our algorithm against three real-world biochemical deep Web sources and report our preliminary results, which are very promising.

## 1. Introduction

Hundreds of freely-accessible databases are available on the Web in the Life Sciences domain, covering areas such as Genomics, Proteomics, Systems Biology and Micro Array Gene Expression, to name a few. These databases often provide complementary data, pertaining to narrow specialized sub-domains. Life Science researchers thus need to search, collect and aggregate data from multiple online resources. This Web site *hopping* is time consuming and error-prone, whereby a user must learn search interfaces of various Web sites, perform multiple copy-paste actions, create temporary text-files and manually link extracted records of interest.

"Deep Web" research aims to *virtually* integrate such Web-accessible databases, provide a unified query interface and, typically, aggregate query results. Deep Web data integration consists of a number of distinct sub-tasks (See [CC06] for a survey):

1. *Source Searching and Clustering* – Searching domain-specific databases on the Web. ([BF05], [HTC04], [Lu06])

2. *Interface Extraction* – Learning query capabilities of forms ([Ku03], [ZHC04])
3. *Interface Matching* – Determining semantic correspondences between query-form attributes across sources. ([Wa04], [Ma05], [WDT06], [HC03])
4. *Interface Merging & Query Translation* – Deriving a unified query interface, translating queries from this interface to source interfaces. ([He03], [KLC05])
5. *Wrapper Generation/Data Extraction* – Learning templates to extract data from result pages. (See section 5 for related work)

We focus on unsupervised wrapper induction and data extraction in this paper. Automatic wrapper induction has received considerable attention in recent years. However, most techniques learn wrappers for one class of Web pages. They assume structurally and content-wise similar pages are manually provided as an input for their wrapper induction methods.

As we shall explain in section 2, in our target domain, data are spread across multiple pages of a Web site, which often differ considerably in their structure and layout (template) as well as content. We therefore need an approach to automatically group similar pages in a Web site for our wrapper induction process. Additionally, we need to automatically discover the Web-site structure, so that we may predict which wrapper to use for a Web page encountered in that Web site. These requirements go beyond traditional wrapper induction methods. We term this compound problem of Web-page classification, site-structure discovery and wrapper induction as *Site-Wide Wrapper Induction*. Though this task is extremely hard in general, in our target domain, i.e. Web-accessible Life Science databases, we may benefit from additional cues in terms of labeling of data. Consequently, we restrict our attention to Web pages from Web sites with labeled data.

In order to solve the challenge of Site-Wide Wrapper Induction in our target domain, we provide the following original contributions in this paper:

I. A novel approach for unsupervised wrapper induction to extract labeled data (Section 3.1). The approach is unsupervised as sample pages (required for induction) belonging to the same *class* or *template* are collected automatically, and no labeling is required.
II. An original integrated approach for Web-page classification and site structure discovery (Section 3.2)
III. An automatic mechanism for detecting and correcting errors in our wrapper learning process (Section 3.3)

The rest of this paper is organized as follows. Section 2 makes some observations about deep Web scientific sources. Based on these observations, we formulate our problem and present our approach for site-wide wrapper induction in section 3. Section 4 presents our results, followed by a review of the related work and comparison to our contributions in section 5. We conclude the paper in section 6 with a brief discussion about future work.

## 2. Life Science Databases on the Web: Observations and Implications

We observe the following about result pages of Life Science Web sources:

***I) Structured Data*** – The results are highly structured. This owes to the fact that the backend relational schemas are very complex, and entities in scientific domains generally have complex relations and associations.

***II) Highly Dynamic Page Structures*** – Data fields that are NULL are often omitted from the results displayed, resulting in pages with widely varying structure. A wrapper induction method which ties its learning process to the page structure only would require numerous training pages, covering all possibilities of data arrangement. One drawback of such an approach would be the need for a large number of input seed queries to probe the deep Web source. Additionally, this would make the learning process slow and put strain on the Web database. A related observation is that Web pages undergo frequent updates [KK02]. An approach which circumvents the need to learn the page structure would hence be desirable.

***III) Labeled Data*** – Scientific data require precision and clear annotation. A natural consequence of this observation is that scientific data are labeled and, often, annotated to controlled vocabularies. This differs from other domains such as E-commerce, where many data fields are often unlabeled because they have become self-explaining in the public domain (e.g. price, title). This labeling can be exploited to not only help determine data regions, but can also serve as anchors for these data regions, allowing us to disregard the portion of the page which does not contain data. We further observe that labels for the same real-world entity can be different across Web pages of the same source. Finally, labels of real-world entities, such as names of biological concepts, rarely change, which can be beneficial with respect to wrapper maintenance.

***IV) Rich Site Structure*** – Data is scattered across multiple Web pages. This gives such Web sites a comprehensive structure. Therefore, the wrapper must be able to navigate through the result pages to extract data. We also observe that some data fields, together with their labels, reappear on multiple pages. This reoccurrence can be used for mutual reinforcement, to detect and correct errors in the wrapper induction process.

One final observation is that of Web service API access. While some Life Science databases do provide such APIs, our survey[1] of 100 online databases showed that only 11 sources provide programmatic access, and even among these the coverage of the database in some cases is not complete. Therefore, Web pages still remain the primary form of data dissemination.

These observations serve to clarify the two broad characteristics of our work: Firstly, at the Web site level, the challenge is to extract data from a number of pages, generated from many templates. This requires determining homogenous clusters of pages having similar templates so that we can induce wrappers for these clusters. Another implicit requirement is that of learning the structure of a Web site through navigational steps. This is essential because our system needs to know which wrapper to apply during data extraction while traversing through the Web site. Secondly, at the Web page level, the

[1] Databases indexed by the Nucleic Acids Research Journal (http://www3.oup.co.uk/nar/database/c/). Complete survey available at http://sabiork.villa-bosch.de/index.html/survey.html

presence of labeled data gives us the opportunity to segment data records and fields based on these labels, and to accommodate the dynamic structure of pages by using these labels to extract data, rather than analyzing and learning the entire Web page structure in a regular expression-like syntax. However, these labels must themselves be identified. Although the vocabulary for labels converges across different sources in a domain [CHZ04], it is not trivial to manually provide a set of possible labels (which can number in their hundreds) to aid in identification of data regions. Therefore, a desirable approach would be to automatically identify these labels.

## 3. The Web-Prospector Wrapper Induction Approach

We follow the divide-and-conquer approach and present our algorithm for page-level wrapper induction in section 3.1. Subsequently, in section 3.2, we describe how this algorithm is used in our site-wide wrapper-induction method. In section 3.3, we discuss a technique to automatically detect and correct certain erroneous results of our induction algorithm presented in section 3.1.

### 3.1 Page-Level Wrapper Induction

Our wrapper induction algorithm relies on multiple sample *instance pages* from a *class of pages*. We borrow this terminology from [CMM01], which describes *a class of pages* in a site as a collection of pages which are generated by the same server-side script or program. Different inputs to this script result in different *instance pages*. We clarify this further using Figure 1, our running example, which shows two instance pages of a class of pages[2]. The wrapper induction algorithm is shown in Figure 2.

| **Entry** | C00221 Compound |
|---|---|
| **Name** | beta-D-Glucose |
| **Formula** | C6H12O6 |
| **Mass** | 180.0634 |
| **Reaction** | R00026 R01520 R01521 R01522 R01600 R01601 R01602 R02187 R02887 R03256 R04783 R06077 R06092 R06110 R06144 |
| **Enzyme** | 1.1.1.47 1.1.3.4 1.1.3.5 2.7.1.1 2.7.1.2 2.7.1.63 3.1.6.3 3.2.1.21 3.2.1.23 3.2.1.85 5.1.3.3 |

| **Entry** | C00185 Compound |
|---|---|
| **Name** | Cellobiose; 1-beta-D-Glucopyranosyl-4-D-glucopyranose |
| **Formula** | C12H22O11 |
| **Mass** | 342.1162 |
| **Reaction** | R00026 R00306 R00952 R01441 R01442 R01443 R01444 R01445 R02365 R02886 |
| **Enzyme** | 1.1.99.18 2.4.1.20 2.7.1.85 3.2.1.4 3.2.1.21 3.2.1.74 3.2.1.91 5.1.3.11 |

Figure 1: Our running example – The results obtained by probing KEGG Compound with C00221 and C00185 respectively.

[2] From KEGG [Ka02]. Portions of these pages have been removed to simplify the discussion and to save space, while remaining true to the challenges encountered

We note that, upon querying, the initial response pages generated by a deep Web source belong to the same class[3]. Therefore, we can probe a source with different inputs, and use the resulting initial pages to learn a wrapper, without having to cluster similarly structured pages. In fact, for a given Web site, the site-wide wrapper induction process is bootstrapped by using these initial result pages and learning a wrapper for this class.
Briefly, the algorithm compares text entries on the sample pages and identifies some (possibly not all) data entries among them. These data entries are subsequently used to identify bigger data regions, so that more data entries can be discovered. A label is then selected for each data entry from text entries outside the data regions, based on vicinity. Our approach is based on the DOM[4] representation of Web pages, and uses XPath[5] for performing the above operations on the DOM tree. The output of the wrapper is a collection of XPath expressions, each pointing to a label and associated data region.

```
   Input: n Web pages P
   Output: R: L => X_gr        //L is a set of labels. X_gr is a set of XPaths to data entries. R is a map from
                               each label l in L to each data XPath dx in X_gr.
   Start
   For each sample page p_i in P{
1        For each text entry t in p_i
2                If t is unique to p_i
                                Add t to D_i;
                        Else
                                Add t to O_i; }
3  For each p_i in P{
           X^d_i = get_XPath(D_i);
           X^o_i = get_XPath(O_i);}
4  D_i', O_i', X^o_i', X^d_i' = reclassify(X^o_i, X^d_i);      // grow the data regions, and reclassify data
   For each XPath dx in X^d_i'{
5          Find closest XPath lx in X^o_i';          // search for XPath of most suitable label in O
           If the corresponding text (label) in O_i' is not in R
                   X = {dx};                         // X is a set containing all data XPaths associated
                                                        with one label
           Else
                   X = X ∪ {dx};
             R = lx => X;}              // XPath of label is mapped to set of corresponding data XPaths
   For each lx in R{
6          Generalize corresponding X to X_g ;    // Create a single XPath from all paths in set X
7          Find relative path X_gr from lx to X_g ;   // relative path from label to data
           Replace X with  X_gr ;
           Replace lx with l';                       // replace Xpath with the corresponding label
   }
   End
```

Figure 2: Wrapper Induction Algorithm

[3] In certain cases, probing a source with an imprecise keyword leads to a disambiguation step. This is a separate research issue and we don't address it in this paper. We assume exact keywords are used to perform the search, as explained in Figure 2.

[4] W3C. Document Object Model. http://www.w3.org/DOM/

[5] W3C. XML Path Language (XPath 2.0) Recommendation. http://www.w3.org/TR/xpath20/

We now explain each step of the algorithm in detail using our running example. Each step is annotated to its corresponding location in the algorithm in Figure 2.
1. The two HTML pages are converted to well-formed XHTML using an HTML-parsing library, i.e. TagSoup[6], so that standard XML tools can be applied to them. Additionally, these pages are subjected to pre-processing, where certain visual formatting features are removed. This is done so that a logically single text entry is located under the same text node in the DOM tree. For example, if there is an underline or bold HTML tag on one word in a sentence, it splits the text of the sentence into more than one text node. While visually this formatting might be important, logically it segments single entries into multiple ones, which is undesirable. Finally, each page is *screen-scraped* to obtain a set T of values contained in all text nodes. Both sets ($T_1$ and $T_2$ in our example) thus contain a union of presentation text, labels and data entries.

2. We compare both sets to initially classify some data entries. Mutually exclusive entries in $T_1$ and $T_2$ are classified as data entries ($D_1$, $D_2$), and the remaining as non-data entries, or "*Other"* ($O_1$, $O_2$). For example:

```
D1 = {C00221, beta-D-Glucose, …, R01520, 1.1.1.47,…}
D2 = {C00185, Cellobiose,…, R00306, 1.1.99.18,… }
O1 = {Entry, Name,…, Reaction, R00026, Enzyme,…, 3.2.1.21}
O2 = {Entry, Name,…, Reaction, R00026, Enzyme,…, 3.2.1.21}
```

Notice that since the data entry R00026 occurs in both instance pages, it is erroneously classified as *Other* at this point.

3. We compute XPath expressions for each entry in the above sets. The expression determines the unique path along the DOM tree for the XHTML file, from the root node to the node containing the entry. For example, the XPath for C00221 is:

```
html/body/…/code[1]/table[1]/tr[1]/td[1]/code[1]/text()
```

4. We use the XPath expressions to reclassify some data entries which might have been wrongly classified in the previous step (such as R00026, 3.2.1.21). This can be considered as *growing* of a data region, whereby data entries are used to reclassify other entries in their vicinity as data, based on their relative positions in the DOM tree. This reclassification step compares an XPath of a data entry with that of an entry not classified as data by applying the following two rules:

*Rule 1 (Last Element Node Rule): If two XPaths are identical and differ only at the ordering of the last element node, and this last element node in the data XPath precedes the last element node in the non-data XPath, the non-data entry is re-classified as data.*

This rule can be explained from an example in Figure 3(a). As shown in the figure, this rule uses data elements to automatically grow data regions towards the right in a table-row. While Figure 3 only shows the example of a table, it is important to emphasize that since this rule is independent of tag names, it works on tags other than those associated to HTML tables, for example, downwards in a list or in a succession of anchor tags. For instance, for the latter case, the XPaths for successive anchor tags (even if they are separated by line breaks) could be: `html/body/a[1]`, `html/body/a[2]`

[6] A SAX-compliant HTML Parser. http://home.ccil.org/~cowan/XML/tagsoup/

It is easy to see from the above example that this rule grows data regions in many types of HTML structures.

*Rule 2 (Penultimate Element Node Rule) If the two XPaths are identical and differ only at the ordering of the penultimate element node, and this penultimate element node in the data XPath precedes the penultimate element node in the non-data XPath, the non-data entry is re-classified as a data entry*

This rule is similar to Rule 1, except it grows data regions down a table-column as shown in Figure 3(b)

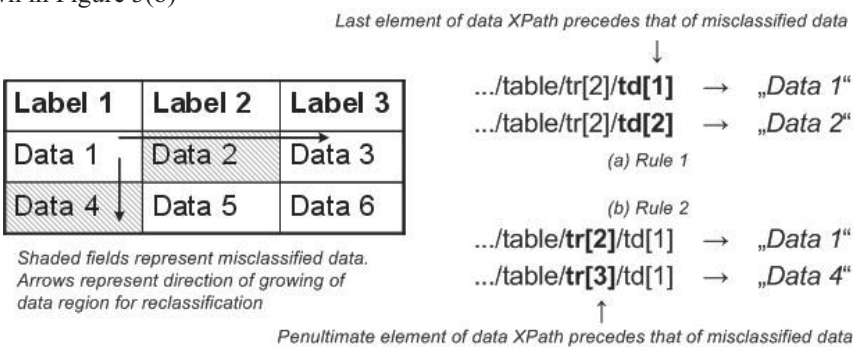



Figure 3: Growing of data region using (a) Rule 1 - Last Element Node Rule (b) Rule 2 - Penultimate Element Node Rule

We note that our rules for growing data-regions operate in only two directions. This is based on the observation that labels generally occur above or towards the left of data [WL03]. Therefore, we restrict our re-classification in these two directions, to avoid incorrectly re-classifying labels as data. There might also be occasions where the above observation does not hold true, though as noted in [WL03], they are rare. In such a case, our rules will erroneously classify labels as data.

After this re-classification step, we have the modified sets:

$D_1'$ = {C00221, beta-D-Glucose, …, R01520, 1.1.1.47,…, 3.2.1.21}
$D_2'$ = {C00185, Cellobiose,…, R00306, 1.1.99.18,…, 3.2.1.21}
$O_1'$ = {Entry, Name,…, Reaction, R00026, Enzyme,…}
$O_2'$ = {Entry, Name,…, Reaction, R00026, Enzyme,…,}

Note that R00026 is not re-classified (incorrectly) because there are no data entries which can grow the data region in its direction. The sets of non-data entries, $O_1'$ and $O_2'$, now contain both presentation text, as well as labels for our data entries.

5. For each data entry in a data set, we now select the closest non-data entry as its label. This can be achieved by comparing XPath of the data entry against the XPaths of the non-data entries. The closer a non-data entry is to a data entry, the more element nodes in their corresponding XPath expressions will be matched before a mismatch. The closest element will have the longest common leading path, which is classified as the label. For example, the XPath for data entry 1.1.1.47 in our example is given by:

```
html/…./table[1]/tr[8]/td[1]/…/code[1]/a[1]
```

Some XPath expressions for the set of non-data entries include, for example;

```
html/…./table[1]/tr[6]/th[1]/…/code[1]/   ("Reaction")
```

`html/…./table[1]/tr[8]/th[1]/…/code[1]/ ("Enzyme")`

The latter XPath has the longest sequence of matching nodes with the XPath of our data element (indicated by the bold-face font above). Therefore, the label ("Enzyme") and corresponding XPath are associated with data entry 1.1.1.47 and its XPath (This association is represented by the "=>" symbol in Figure 5).

Note that multiple data elements can be associated with a single label in such a manner, as shown in Table 1. The last row of the table shows that the data entry R00026, which has been misclassified previously, has been selected as a label for data entries R01520, R01521 etc, as it found to be the closest non-data entry.

| **Label – With XPath** | **Data Entry** | **XPath of Data Entry** |
|---|---|---|
| **Enzyme** - html/…./th[1]/…/code[1]/ | 1.1.1.47 | html/…/td[1]/…/code[1]/a[1] |
| | 1.1.3.4 | html/…/td[1]/…/code[1]/a[2] |
| | ….. | ….. |
| **R00026** - html/…/th[1]/…/code[1]/a[1] | R01520 | html/…/td[1]/…/code[1]/a[2] |
| | R01521 | html/…/td[1]/…/code[1]/a[3] |
| | …. | …. |

Table 1: Two labels inferred, with corresponding data entries and their XPath expressions.

6. The XPaths of data entries classified to the same label are then generalized to form a single XPath expression. The XPaths for data entries in Table 1, for example, can be generalized as given below:

```
html/…./table[1]/tr[8]/td[1]/…/code[1]/a[position()≥1]/text()
html/…./table[1]/tr[6]/td[1]/…/code[1]/a[position()≥2]/text()
```

The last XPath expression above, for example, selects all text entries pointed to by a collection of anchor tags, starting from the second anchor tag. This is required as sample pages may only contain a small number of multiple data entries associated to a label. What is required is that we recognize and generalize that multiple number of data entries are associated for that label, rather than the number of data entries *seen* by the wrapper induction algorithm.

7. A relative path from the label to its corresponding generalized data path is computed. For the "Enzyme" label in Table 1, the relative path to its data is:

```
../…./../ td[1]/…/code[1]/a[position()≥2]/text()          (1)
```

Finally, the XPath for the label is replaced with an *anchored* XPath expression, i.e., an XPath which directly access the text node, and does not utilize ancestor nodes. For the "Enzyme" label:

```
//*[text()='Enzyme']                                      (2)
```

Concatenating (1) with (2) gives us, for label "Enzyme":

```
//*[text()='Enzyme']../…./td[1]/…/code[1]/a[position()≥2]/text()
```

The wrapper, thus, comprises of a collection of labels associated with a generalized anchored XPath expression to extract corresponding data.

**Discussion** It is worth noting here that the wrapper learnt by our algorithm is not tied to the structure of a class of pages. The wrapper anchors, or pivots, to a particular label, and finds a relative path from the label to associated data entries. As noted in section 2, data pages returned by Web databases can be very dynamic, where some labels can be

omitted completely. Our approach is beneficial in this case, as well as in the case that a Web site undergoes a template redesign, for instance. As long as the relative path from a label to its corresponding data remains the same, there would be no need to re-learn the wrapper. The only other limitation is that of labels remaining constant, and as we mentioned in section 2, changes in names of real-world entities, such as biological concepts, is extremely rare. Finally, recall that the wrapper-induction process for our running example results in the misclassification of "R00026" as a label. We discuss a technique to automatically detect and possibly correct such errors in section 3.3.

### 3.2 Site-Wide Wrapper Induction

As we noted in section 1, data-intensive sites, such as those in the Life Sciences domain, have their data scattered across multiple pages. Therefore, we need a wrapper-induction strategy that extracts data from multiple pages, which might belong to different page classes. This implies that we need to not only discover which pages returned by the server belong to the same class, but also to distinguish between classes and the navigational steps between them. Here, we make the following observations:

1. Not all pages of a Web site contain data, for example, pages pointed to by navigational menus, help pages, contact information etc. Therefore, we do not wish to discover all page classes. Rather, we wish to perform *targeted crawling* to only seek out *data-pages* and discover their classes.
2. We observe the concept of *link-collection* [CMM03], which refers to anchor links in a page (class) that share the same path in the DOM tree, from the root element to their parent or grandparent element. As a result, these hyperlinks appear grouped together in the rendered page. In our running example of Figure 1, the links on the reaction names form a link-collection, as well as those on the enzyme names. A link collection may be a singleton as well, comprising only a single hyperlink. We also note that hyperlinks in such a collection might not point to the same class of pages. For example, links in a navigation bar or those in categorization menus. However, link-collections that have been *classified* as being over data regions point to the same class of pages. For example, all hyperlinks on enzyme names point to "enzyme details" pages.
3. Pages belonging to the same class contain similar set of labels. However, due to the highly dynamic nature of pages, some labels may be omitted (e.g. NULL values in databases), but their ordering typically remains the same, as they are generated by scripts. If the order of labels on two pages is different, then their page-structures will most likely be different as well. Furthermore, it is highly unlikely that a site will have two different templates to display the same set of labels in the same order.

We, therefore, base our approach for site-wide wrapper induction on these assumptions:
1. Given the initial result output of a deep Web source, all data-intensive pages can be reached by iteratively following link-collections that occur on data regions. This assumption allows us to do targeted crawling for data-intensive pages, and eliminate navigation bars and menus etc.
2. A pre-classified link-collection points to the same class of pages.
3. Classes of pages can be distinguished from each other based on the labels they contain, and their order.

We now model our site-wide wrapper-induction problem as follows:
A page class $C_i$ is defined by $C_i = SEQ_i$, where $SEQ_i = (\ell_{i1}, \ell_{i2},\ldots)$, a sequence of labels $\ell_{i1}$, $\ell_{i2}$, … described in page class $C_i$ in this order of arrangement. These labels may have link-collections associated with them. Two classes $C_i$ and $C_j$ are considered not equal if $SEQ_i \neq SEQ_j$

Our site model is given by a collection of navigation steps:
$R_{ijn} = C_i \rightarrow \ell_{im} \rightarrow C_j$ $(i \neq j, i, j, n, m \geq 0)$
Where $\ell_{im}$ is the m-th label in page class $C_i$, the associated link-collection of which points to pages of class $C_j$. The site model is schematically depicted in Figure 4.

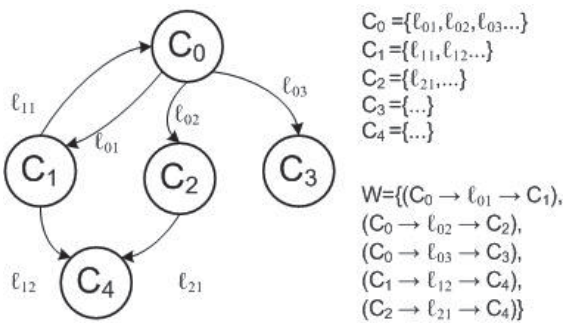



Figure 4: Schematic representation of a Web site Model as a labeled directed graph, with page-classe C composed of labels ℓ, the navigation steps between classes represented with → symbol.

The goal of the site-wide wrapper induction algorithm is to find the following:
1. $C_i \neq C_j$ $(i \neq j, i, j \geq 0)$
2. $R_{ijn} = C_i \rightarrow \ell_{im} \rightarrow C_j$ $(i \neq j, i, j, n, m \geq 0)$
That is, all possible data-intensive page classes, and the navigation steps between them.

The algorithm for site-wide wrapper induction is presented in Figure 5. We explain the algorithm using our running example of Figure 1. As in section 3.1, each step explained below is annotated to its corresponding location in the algorithm in Figure 5.

1. The algorithm starts with an input of the initial page class $C_0$, which corresponds to the initial response pages of the Web source. In our case, this is the page class that is generated by the sample pages of our running example, shown in Figure 1.
2. For each label in this class, the corresponding link-collections are followed. Let's assume we follow the link-collection of "Enzyme". Portions of sample results are shown in Figure 6.
3. According to our assumption 2, these pages belong to the same class. We learn a wrapper for this page class using our algorithm in Figure 2, with these sample pages as input. We note that not all links in a link-collection need to be followed. Our initial experiments have shown that ~9 sample pages yield a very good result (See section 4).
4. If this wrapper learning process results in a new class, according to our assumption 3, we add this new class to our sets, and define its corresponding navigational steps. In our example, a new class is created, $C_1$ = (Entry, Name, Class,…), as well as a navigation step $R = (C_0 \rightarrow$ "Enzyme" $\rightarrow C_1)$.
5. The above steps are repeated for each new page class that is learnt in step 3.

```
1  Input: S={C0}, C0=(ℓ01, ℓ02,…); // Set of all page classes discovered. C0 corresponds to the
                                    initial results page of a deep Web source.
   W = {};                          //Set of navigation steps between page classes.
   Output: S={Ci} (i≥0)
   W={Rij} (i,j≥0, i≠j)
   Start
   S´ = S;
   Do{
2      For each C in S´{            // for each class in our set
          For each ℓ in C{          // for each link-collection associated with a label
                    Follow ℓ;       // follow the link-collection
3                   induceWrapper Cnew;   // induce wrapper
4                   if (Cnew ∉ S) {          // if this is a new class
                                 Add Cnew in S and S´;}   // add it to our set
                    R = (C →ℓ→Cnew);      // form the navigation step
                    Add R in W;}          // add the step
                        }
           Remove C from S´;
5  }While(S´ ≠ NULL)             // all classes' link-collections have been explored
   End
```

Figure 5: Site-Wide Wrapper Generation Algorithm

| Entry | EC 2.7.1.2 Enzyme |
|---|---|
| Name | glucokinase;<br>glucokinase (phosphorylating) |
| Class | Transferases;<br>Transferring phosphorus-containing groups;<br>Phosphotransferases with an alcohol group as acceptor |

| Entry | EC 1.1.1.47 Enzyme |
|---|---|
| Name | glucose 1-dehydrogenase;<br>D-glucose dehydrogenase (NAD(P)+);<br>hexose phosphate dehydrogenase |
| Class | Oxidoreductases;<br>Acting on the CH-OH group of donors;<br>With NAD+ or NADP+ as acceptor |

Figure 6: Small excerpts of pages obtained from following "Enzyme" link-collection

### 3.3 Error-Detection by Mutual Reinforcement

The natural residual output of our site-wide wrapper-induction approach is labeled data. These labels and data can be used to automatically detect and possibly correct errors in our wrapper-induction method for a page-class. We observe that some data entries reappear on different page classes. For example, the enzyme classification numbers in Figures 1 and 6. If the reappearing entries have been correctly classified as data across different page-class wrapper induction runs, then this enforces our confidence that the classification is correct. On the other hand, if, for example, some entries are classified as labels or presentation text by some wrappers and data by others, then this clearly points to a misclassification. This indicates that not enough sample pages were available to distinguish between data, labels and presentation text. We can address this by providing more samples for these page classes. We call such a mismatch as *label-data mismatch*.

For example, recall from Section 3.1 that the data entry R00026 was misclassified as a label in our running example (for page class $C_0$). When we follow the "Reaction" link-collection, we come across the page shown in Figure 7. While learning the wrapper for this class of pages ($C_1$), R00026 will be (correctly) classified as data. Based on this mismatch, we introduce more learning pages for both $C_0$ and $C_1$. In our example, any page for class $C_0$ which does not contain "R00026" as an entry will force our algorithm to classify this entry as data, thereby correcting the label to "Reaction" as well. The other type of mismatch is *label-label mismatch*, where the same data entry is assigned different labels across page classes. Recall our observation from section 2 that the same data entries can be labeled differently across different page classes. This can be observed from Figures 1 and 7, where R00026 is labeled as "Reaction" and "Entry", respectively. Based on this observation, it is not possible to detect whether a label-label mismatch was an error or a correct classification.

| **Entry** | R00026 Reaction |
|---|---|
| **Name** | beta-D-Glucoside glucohydrolase |
| **Definition** | Cellobiose + H2O <=> 2 beta-D-Glucose |
| **Equation** | C00185 + C00001 <=> 2 C00221 |

Figure 7: Top-most portion of "Reaction" page of R00026 From KEGG

We slightly modify our site-wide wrapper induction algorithm to incorporate automatic error detection and correction for label-data mismatches. We introduce this mutual reinforcement step each time a new page class is created. The entries classified as data in this new class are compared with labels of previously formed page classes. If a mismatch is found, more sample pages for this new class, and conflicting page class, are introduced for wrapper generation until the mismatch is resolved.

## 4. Results and Evaluation

We have developed a prototype in Java which implements our algorithms. We use it to perform some preliminary experiments on three real-world biochemical sources, namely KEGG[Ka02], ChEBI[De08] and MSDChem[Go04]. All these sources provide basic qualitative data, and are often used for reference or annotation in more specialized domains, such as Genetics, Proteomics and Systems Biology. We use a simple random sample of input values for search forms of these sites in order to probe and induce their initial results page[7]. A Web-crawler based on httpUnit[8] was manually configured to fill the search forms with these values and submit them. We first report the results of our page-level wrapper-induction algorithm in section 4.1, and in section 4.2, describe our findings on site-wide wrapper-induction.

### 4.1 Page-Level Wrapper Induction Results

We perform two evaluations of the wrapper induction algorithm: The first evaluation (section 4.1.1) is that of the XPaths retrieved, which is beneficial for analyzing the

[7] The values can be collected from downloadable flat files or Web services provided by each source.

[8] A Java library for automated testing of Web sites. http://httpunit.sourceforge.net/

specific cases under which our algorithm misclassifies labels and data, and can help in improving the algorithm. It is also useful for determining how many samples are required in general by our algorithm for learning a wrapper for a class of pages. The second evaluation (section 4.1.2) is of the accuracy of the learnt wrapper in a traditional information retrieval evaluation.

### 4.1.1 XPath Analysis

We determine the precision and recall of our algorithm in retrieving the labels and corresponding data present in a page class. The total number of labels in each class was determined a priori by a domain expert.[9] In addition, the expert also determined the cardinality of the data associated with each label – specifically whether the label had one or possibly multiple data entries as its values. (Recall from section 3.1 that the algorithm *generalizes* multiple XPaths into a single expression for multiple data entries associated with the same label).

The correctness of each XPath returned by the algorithm is determined by the following three factors: (1) The correctness of the label, (2) that of the relative path from the label to the data, and (3) that of the generalization in case of a cardinality relation of greater than one between the label and data. We manually check these a posteriori. Verifications of labels and of generalization of the XPath are done manually. Note that since the relative path between a label and its data is always constant, we can verify the retrieved relative path by executing the XPath on a single test page by observing if it returns the correct data. The results are shown in Table 2. For each page-class, Table 2 shows the total number of label-data pairs contained in that class (#LD), the number of sample pages used for wrapper induction (#s), the number of label-data pairs successfully learnt, i.e. our *true-positives* (TP), the number of pairs not learnt, i.e. our *false-negatives* (FN) and the ones incorrectly learnt, i.e. *false-positives* (FP). We report our precision P and recall R in the last two columns respectively.

Before discussing the results, we briefly explain some conditions under which false-negatives and false-positives occur. False-positives generally occur when there is unlabeled data present in the pages. These data usually occur at the top of a page, such as a heading or a large caption, and are often *redundant* data entries, as they reappear as labeled data later in the same page (e.g, compound and reaction identification numbers etc). The other occurrence of false-positives is when data entries are misclassified as labels (as "R00026" in our example, section 3.1). This *may* also result in a corresponding false-negative for the *missed* label (as "Enzyme" due to the misclassification of "R00026"). Lastly, false-negatives also occur when sample pages used for the induction process simply do not contain the labels. This is a limitation for all wrapper induction approaches – you can only learn what you see.

As Table 2 indicates, our best results are obtained for KEGG Reaction, with a precision and recall of 100%, with 6 sample pages. This owes to the fact that there are frequent pages in KEGG Reaction which contain all labels. The results for KEGG Compound and ChEBI are quite similar. In both cases, our wrapper was unable to learn one label

[9] The total number of labels can be manually obtained from help pages and FAQs of the sources.

("Sequence" and "IN Number" respectively) for each source. Manual inspection revealed that none of our 15 sample pages contained those labels, which leads us to believe that they occur rarely. As noted above, this is a general problem for learning systems, and can only be removed with more sample pages.

| Source | #LD | #S | TP | FN | FP | P | R |
|---|---|---|---|---|---|---|---|
| KEGG Compound http://www.genome.jp/kegg/compound/ | 10 | 3 | 5 | 5 | 2 | 71.4% | 50% |
| | | 6 | 8 | 2 | 2 | 80% | 80% |
| | | 9 | 9 | 1 | 1 | 90% | 90% |
| | | 12 | 9 | 1 | 0 | 100% | 90% |
| | | 15 | 9 | 1 | 0 | 100% | 90% |
| KEGG Reaction http://www.genome.jp/kegg/reaction/ | 10 | 3 | 8 | 2 | 0 | 100% | 80% |
| | | 6 | 10 | 0 | 0 | 100% | 100% |
| ChEBI http://www.ebi.ac.uk/chebi/ | 22 | 3 | 18 | 4 | 3 | 85.7% | 81% |
| | | 6 | 20 | 2 | 0 | 100% | 90.9% |
| | | 9 | 21 | 1 | 0 | 100% | 95.4% |
| | | 12 | 21 | 1 | 0 | 100% | 95.4% |
| | | 15 | 21 | 1 | 0 | 100% | 95.4% |
| MSDChem http://www.ebi.ac.uk/msd-srv/msdchem/cgi-bin/cgi.pl | 30 | 3 | 30 | 0 | 1 | 96.7% | 100% |
| | | 6 | 30 | 0 | 1 | 96.7% | 100% |
| | | 9 | 30 | 0 | 1 | 96.7% | 100% |
| Average (based on final wrappers for each source) | | | | | | 99.1% | 96.3% |

Table 2: Results for page-wide wrapper induction algorithm. (LD is label-data pairs, s is samples)

| | |
|---|---|
| Formula | C10 H16 N5 O13 P3 |
| Defined at | 1999-07-08 |
| Last modified at | 2007-08-16 |
| EBI name | ADENOSINE-5'-TRIPHOSPHATE |
| EBI Id | not assigned |
| Additional name | ADENOSINE-5'-TRIPHOSPHATE |
| Classification | NUCLEOTIDES |
| Cas reg number | not assigned |
| Therapeutic category | not assigned |
| Merck Id | not assigned |
| Polymer topology | not assigned |
| Polymer code | not assigned |
| Polymer sub type | not assigned |
| Hetgroup type | NON-POLYMER |
| Obsoleted | not assigned |
| Parent | not assigned |
| Topological variant | not assigned |

Figure 8: Portion of MSDChem page for "ATP", showing unassigned values

The results for MSDChem are quite interesting, as they demonstrate the usefulness of our *data-region growing* approach. Unlike other sources, pages in MSDChem have a very static structure – no labels are omitted from the pages when corresponding data entry is NULL, as shown in Figure 8. This means that the frequency of data fields being NULL (or "Not Assigned" in this example) is very high. Such fields are not classified as data in our algorithm, as they are constant across many pages. However, we note that the label-data pairs are arranged in a (invisible) table, as shown in Figure 8. Therefore, through rule 2 in Section 3.1, an entry classified as data at the top of the data column in Figure 8 reclassifies all entries below it in the column as data as well. This accounts for a perfect recall, with only 3 sample pages. Overall, we observe that we can get very high

precision and recall (~97%, ~96% respectively) from ~9 samples. The precision can be improved with more samples, especially if they contain rarely occurring labels.

### 4.1.2 Wrapper Evaluation

We verify the accuracy of our wrappers by applying them to sets of five test pages. We manually count the total number of data entries and note corresponding labels across these five test pages a priori, and determine the precision and recall of our algorithm in retrieving these data entries and classifying them with the right label. These results are summarized in Table 3. The wrapper for the page class belonging to KEGG Reaction has a perfect precision and recall. This reinforces our observation in the previous section that there are frequent pages in this class which contain all labels. Our algorithm is thus able to correctly induce a wrapper for this class with only a few samples. The wrappers for KEGG Compound and ChEBI are unable to retrieve data entries corresponding to the labels which were not learnt (see Table 2). The wrapper for MSDChem has FPs as a result of false classification of redundant data entries to some presentation text in the learning phase.

| SOURCE | #T | (TP) #RC | (FN) #NR | FP | | PRECISION | RECALL |
|---|---|---|---|---|---|---|---|
| | | | | #R | #IR | | |
| KEGG Compound | 186 | 184 | 2 | 0 | 0 | 100% | 98.9% |
| KEGG Reaction | 45 | 45 | 0 | 0 | 0 | 100% | 100% |
| ChEBI | 204 | 203 | 1 | 0 | 0 | 100% | 99.5% |
| MSDChem | 150 | 150 | 0 | 0 | 5 | 96.7% | 100% |
| Average | | | | | | 99.1% | 99.6% |

Table 3: Results from applying wrappers to five sample pages each. (T = Total number of data entries across 5 test pages, RC = Retrieved and Classified correctly, NR = Not Retrieved, R = Retrieved but not classified correctly, IR = Incorrect Retrieval)

## 4.2 Site-Wide Wrapper Induction

In this section, we present our results for site-structure discovery, which together with the wrapper induction algorithm constitutes our site-wide wrapper induction approach. As in section 4.1, we perform two evaluations: The first determines to what extent we were able to determine the Web site structures (section 4.2.1), which gives us cues to modify and improve our algorithm. The second determines the accuracy of our Web site wrapper induction algorithm, by applying the algorithm to small portions of the site to retrieve data (section 4.2.2).

### 4.2.1 Analysis of Retrieved Site Models

We manually model all three sources, which involves manually determining classes for data pages for a source, and the navigation steps for generating these classes[10]. We then compare our algorithm's results and measure its accuracy by comparing the navigation steps generated (section 3.2) with the manually determined steps.

[10] We ignore classes of pages which describe content using Flash or Java applets etc., as our system can only process HTML pages. This restriction limits the amount of information our algorithm can extract. However, such content is still quite limited in our target domain, and the vast amount of data is typically available in textual form on HTML pages.

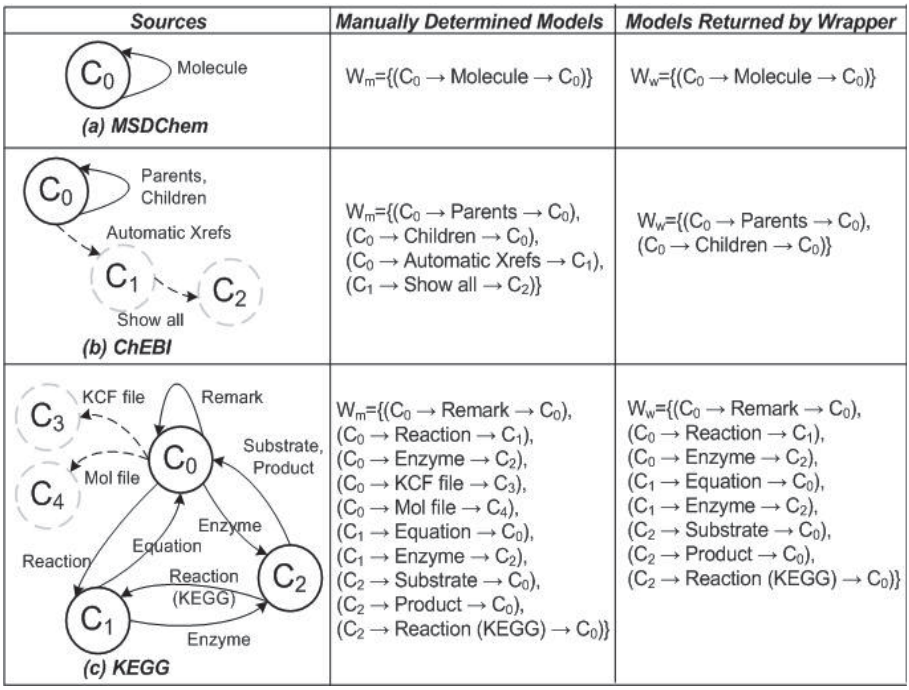


Figure 9: Manual and retrieved site models for (a) MSDChem, (b) ChEBI (c) KEGG (partial). Solid lines: Portions of the model discovered correctly. Dashed lines: portions not retrieved.

These results are partially shown in Figure 9, along with schematic representations of the Web site models, with the complete results in Table 4. MSDchem and ChEBI have relatively simple models. KEGG on the other hand has a very complex model. It actually consist of a number of back-end database schemas, each having its unique Web interface, with extensive links between all interfaces to form a large Web portal, with more than 30 page classes. For this paper, we restrict our manual model to a specific sub-site (KEGG Compound, Drug, Reaction, RPair, Enyzme and Orthology). We restrict our algorithm from exploring the portal outside this boundary, allowing it to discover navigation steps within this sub-site. It is immediately apparent from Table 4, based on perfect precision, that our assumption that all link-collections associated with data regions point to classes of pages containing data, is indeed correct. However, the relatively low recall seems to suggest that we need to relax the restriction that only link-collections associated with data regions should be followed. Indeed, in the site models that our algorithm retrieves, we notice that certain page-classes have not been discovered. Therefore, we probably need to adjust our algorithm to follow link-collections that are also close to data regions, but not necessarily directly on them.

| SOURCE | #NS | TP | FN | FP | P | R |
|---|---|---|---|---|---|---|
| MSDChem | 1 | 1 | 0 | 0 | 100% | 100% |
| ChEBI | 4 | 2 | 2 | 0 | 100% | 50% |
| KEGG | 21 | 16 | 5 | 0 | 100% | 76.19% |

Table 4: Results for site-structure discovery. (#NS is total number of navigation steps)

#### 4.2.2 Site-Wide Wrapper Evaluation

We apply our site-wide wrappers to the three sources to extract data. We limit the execution so that our system extracts data from only five test instances of each class of the Web site. We manually count the total number of data entries and note corresponding labels across all test pages a priori, and determine the accuracy of the algorithm similarly to section 4.1.2, with results displayed in Table 5.

| SOURCE | #C | #C' | #T | (TP) #RC | (FN) #NR | FP | | P | R |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | #R | #IR | | |
| MSDChem | 1 | 1 | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| ChEBI | 3 | 1 | 412 | 247 | 165 | 0 | 0 | 100% | 59.9% |
| KEGG | 10 | 7 | 1618 | 1422 | 109 | 87 | 0 | 94.2% | 87.8% |
| Average | | | | | | | | 97.1% | 73.9% |

Table 5: Site-wide wrapper evaluation. (#C = Total number of classes, #C' = Number of classes discovered, T = Total number of data entries across 5 test pages, RC = Retrieved and Classified correctly, NR = Not Retrieved, R = Retrieved but not classified correctly, IR = Incorrect Retrieval)

We observe that for MSDChem, even though the navigation steps constituting the site model are correct, the site-wide wrapper induction fails. Upon closer inspection, we notice that the navigation step from a page instance actually results in the same page instance. For example, following the "Molecule" link-collection (Figure 9) from a page of ADP results in the same page. This implies that the MSDChem Web site consists of a large number of leaf nodes only, having no hyperlinks connecting them to each other. For ChEBI, we have a perfect precision, but a low recall. This indicates that the two classes our algorithm failed to retrieve (Figure 9) had rich data regions. Our algorithm also fails to retrieve 3 classes in the KEGG sub-site, though a relatively higher recall suggests these missing classes did not contain as big a data region as in the case of ChEBI. Furthermore, we have some misclassifications in some page wrappers for KEGG which slightly lower the precision for the corresponding site-wide wrapper for KEGG.

## 5. Related Work

### 5.1 Wrapper Induction

Wrapper generation has received considerable attention for a long time now. The earliest approaches, including [Ku98] and [MMK00] required training examples. Due to the large size of the Web and its dynamic nature, supervised techniques do not scale well. Recent attempts have focused on fully automatic wrapper induction techniques. The reader is instructed to read [La02] for a survey on wrapper induction techniques. RoadRunner [CMM01 ,CMM04] is an automatic wrapper induction algorithm that is closest to our approach, as it uses multiple sample pages of a page-class. However, unlike our approach, it compares the structures of the sample pages to learn a regular expression, which takes into consideration the mismatches between text and HTML tags across the samples. This regular expression based wrapper is thus tied directly to the page structure. As we noted in section 2, pages from deep Web sources are often very dynamic, where concepts that are NULL are often omitted. RoadRunner would thus

require a large number of sample pages, covering all possible types of such omissions, so that its regular expression can accommodate for this dynamic behavior.

Lixto [BFG01] and W4F [SA01] use XPath-like languages "Elog" and "HEL" respectively, and both offer visual tools for creating wrappers in an unsupervised manner. The user selects data of interest in a Web page, and a path from the root of the page to the target node is generated in the respective languages. Therefore, manual identification of data elements is required for each page, which can be labourious for pages containing numerous data entries. ANDES [MJ02] is based on XPath and requires the user to manually provide XPath expressions to extract data. [An06] builds on ANDES to induce the XPath expressions using tree traversal patterns but requires annotated examples. IEPAD [CCL03], DeLA [WL03], ViNTs [MRY05], DEPTA [ZL06] and ViPER [SL05] are unsupervised wrapper induction techniques that are all based on one common assumption: Data regions in Web pages are constituted by at least two spatially consecutive records that are structurally and visibly similar. This assumption partially holds for result pages of search engines, online listings and E-commerce Web sites, but not for scientific repositories on the Web, as is apparent from our example in Figure 1. Even in the case of E-commerce sites and listings, the initial response pages of a search do exhibit a repetitive structure comprising of records, but the *details* pages describing each result do not exhibit this repetitiveness. All approaches cited above perform wrapper induction on a single class of pages, whereas our approach attempts to automatically classify pages in a Web site into appropriate classes, learn wrappers for each class and discover rules for applying these wrappers on Web pages encountered on the Web site. IDE [ZL07] extracts structured data from different classes of Web pages. It starts with one labeled training page, indicating the information to be extracted. It proceeds to extract corresponding data from test pages based on the similarity between a consecutive sequence of tags before and after the labeled data and the data in the test pages. Whenever extraction fails for a page, it is manually labeled. However, this requires foreknowledge about which information must be extracted, and assumes that the same information is present and to be extracted from all classes. This is very useful in E-Commerce, but not always true in scientific sources, as shown by examples in Section 3.

### 5.2 Site-Structure Discovery

We are only aware of one approach to automatic site-structure discovery [CMM03], which also constitutes the main motivation for our approach. The focus of their work is slightly different from ours – It tries to efficiently discover the entire site-structure, whereas we focus on discovering only portions of the site which contain data extracted from the backend database. Their approach to clustering of Web pages into classes is based on the assumption that pages belonging to the same class contain link-collections that are in a structurally similar arrangement and position. Based on structural similarity of these link collections, they group Web pages into classes. This is a good assumption for sites that do not have *leaf* pages which do not have any links, such as help pages, FAQs, contacts, legal disclaimers etc. In the absence of hyperlinks, all these pages would be classified into a single class (because their link-collections have the same structure), even though these pages may exhibit considerable structural variations. Our approach is also based on an assumption over link-collections, but contrary to their assumption, we

assume that link-collections that have been classified as belonging to the same concept point to pages which belong to the same class.

## 6. Conclusions and Future Work

We have described a novel wrapper induction technique to extract labeled data from data-intensive Web pages of deep Web sources. The approach takes advantage of the peculiarities typically associated with scientific Web sites, most notably that they contain labeled data. Our approach is unique in that it automatically classifies structurally similar pages into classes which can then be used for learning wrappers. Navigation steps that are retrieved during the site-wide wrapper induction phase are used to associate wrappers to classes of pages, allowing us to automatically select and apply a wrapper for a page in the Web site. The approach is fully automatic, the samples required for page-level wrapper induction are collected automatically and do not require any manual labeling. The approach does not need fine-tuning of any heuristics or parameters, but does require the presence of labels. Our approach is less prone to structural changes or updates of the Web pages, as it does not marry the induced wrapper to the pages structures, and only requires the relative path in the HTML tree between the label and data. Our ultimate goal is to be able to integrate and query data from multiple deep Web sources. We briefly mention our future work in this direction: As mentioned in section 2, data extracted from a single source may have multiple labels. Furthermore, different sources may also have different labels for the same entity. A first step is to homogenize this labeling. This is a *matching* problem in traditional data integration systems. Secondly, data extracted from applying the wrapper lack a schema associated with it, whereas querying and using these data requires a rich schema, similar to that of the hidden database, or an ontology. The arrangement of different labels, repetitive patterns and cardinalities of the data might give us some clues to discovering the hidden schema.

**Acknowledgements**
The first author was supported by the Klaus Tschira Scholarship, grant number 09.005.2004.

## Bibliography

[An06] Anton, T.: XPath-Wrapper Induction by generalizing tree traversal patterns. In Workshop on Web Mining, in ECML/PKDD 2006.

[BF05] Barbosa, L.; Freire, J.: Searching for Hidden-Web Databases. In WebDB, p 1-6, 2005.

[BFG01] Baumgartner, R.; Flesca, S.; Gottlob, G.: Visual Web Information Extraction with Lixto. Proc. 27th Interntnl. Conference on Very Large Data Bases:119 – 128, 2001.

[CC06] Chang, K. C.-C.; Cho, J.: Accessing the Web: From Search to Integration. In Proceedings of the 2006 ACM SIGMOD Conference, June 2006.

[CCL03] Chang, C.-H.; Hsu, C.-N.; Lui, S.-C.: Automatic information extraction from semi-structured web pages by pattern discovery. SCI expanded, 35(1):129–147, 2003. Special Issue on Web Retrieval and Mining.

[CHZ04] Chang, K. C.-C.; He, B.; Zhang, Z.: Mining Semantics for Large Scale Integration on the Web: Evidences, Insights and Challenges. SIGKDD Explorations, 6(2):67-76, December 2004.

[CMM01] Crescenzi, V.; Mecca, G.; Merialdo, P.: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In VLDB p. 109-118, 2001.

[CMM03] Crescenzi, V.; Merialdo, P.; Missier, P.: Fine-grain web site structure discovery. WIDM 2003: 15-22.

[CMM04] Crescenzi, V.; Mecca, G.; Merialdo, P.: Improving the expressiveness of ROADRUNNER. SEBD 2004: 62-69.

[De08] Degtyarenko, K. et. al..: ChEBI: a database and ontology for chemical entities of biological interest. Nucleic Acids Res. 36, (2008) D344–D350.

[Go04] Golovin, A. et. al.: E-MSD: an integrated data. Nucleic Acids Research, 32 (Database issue), D211-D216. 2004.

[HC03] He, B.; Chang, K. C.-C.: Statistical Schema Matching across Web Query Interfaces. In SIGMOD Conference, pages 217-228, 2003.

[He03] He, H.; Meng, W.; Yu, C. T.; Wu, Z.: WISE-Integrator: An Automatic Integrator of Web Search Interfaces for E-Commerce. In VLDB, pages 357-368, 2003.

[HTC04] He, B.; Tao, T.; Chang, K. C.-C.: Organizing structured web sources by query schemas: a clustering approach. In CIKM, pages 22-31, 2004.

[Ka02] Kanehisa, M.. The KEGG database. Novartis Found Symp 2002,247:91-101; discussion 101-3, 119-28, 244-52.

[KK02] Knoblock, C.; Kambhampati, C.: Information Integration on the Web. In AAAI 2002.

[KLC05] Kabra, G.; Li, C.; Chang, K. C. C.: Query Routing: Finding Ways in the Maze of the DeepWeb. WIRI 2005: 64-73

[Ku98] Kushmerick, N.: Wrapper Induction for information extraction. Proc. 15th Intl. Conference on Artifical Intelligence, ICAI 1998.

[Ku03] Kushmerick, N: Learning to Invoke Web Forms. In CoopIS/DOA/ODBASE, pages 997-1013, 2003.

[La02] Laender, A. H. F.; Ribeiro-Neto, B.; Silva, A. S. D.;Teixeira, J. S.: A brief survey of web data extraction tools. ACM SIGMOD Record, 31(2):84–93, 2002.

[Lu06] Lu, Y.; He, H.; Peng, Q.; Meng, W.; Yu, C. T.: Clustering e-commerce search engines based on their search interface pages using WISE-Cluster. Data Knowl. Eng. 59(2): 231-246, 2006.

[Ma05] Madhavan, J.; Bernstein, P. A.; Doan, A.; Halevy, A. Y.: Corpus-based Schema Matching. In ICDE, pages 57-68, 2005.

[MJ02] Myllymaki, J.; Jackson, J.: Robust Web Data Extraction with XML Path Expressions. IBM Research Report, May 2002.

[MMK00] Muslea, I.; Minton, S.; Knoblock, C.: Stalker: Learning extraction rules for semistructured, web-based information sources. Proc. AAAI-98: Workshop on AI and Information Integration. AAAI Press, 1998.

[MRY05] Meng, W.; Raghavan, V.; Yu, C.: Fully automatic wrapper generation for search engines. In WWW14, May 2005.

[SA01] Sahuguet, A.; Azavant, F.: Building intelligent Web applications using lightweight wrappers. Data Knowl. Eng. 36(3): 283-316, 2001.

[SL05] Simon, K.; Lausen, G.: ViPER: augmenting automatic information extraction with visual perceptions. CIKM 2005: 381-388.

[Wa04] Wang, J.; Wen, J.-R.; Lochovsky, F. H.; Ma, W.-Y.: Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. In VLDB, p 408-419, 2004.

[WDT06] Wu, W.; Doan, A.; Yu, C. T.: WebIQ: Learning from the Web to Match Deep-Web Query Interfaces. In ICDE, page 44, 2006.

[WL03] Wang, J.; Lochovsky, F. H.: Data extraction and label assignment for web databases. In WWW12, pages 187–196. ACM Press, 2003.

[ZHC04] Zhang, Z.; He, B.; Chang, K. C.-C.: Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In SIGMOD Conference, pages 107-118, 2004.

[ZL06] Zhai, Y.; Liu, B.: Automatic Wrapper Generation Using Tree Matching and Partial Tree Alignment. In AAAI 2006, Boston, USA, July 16 - 20, 2006.

[ZL07] Zhai, Y.; Liu, B.: Extracting Web Data Using Instance-Based Learning. WWW2007: p113-132

# Easy Tasks Dominate Information Retrieval Evaluation Results

Thomas Mandl

Information Science
University of Hildesheim
Marienburger Platz 22, 31141 Hildesheim, Germany
mandl@uni-hildesheim.de

**Abstract:** The evaluation of information retrieval systems involves the creation of potential user needs for which systems try to find relevant documents. The difficulty of these topics differs greatly and final scores for systems are typically based on the mean average. However, the topics which are relatively easy to solve, have a much larger impact on the final system ranking than hard topics. This paper presents a novel methodology to measure that effect. The results of a large evaluation experiment with 100 topics from the Cross Language Evaluation Forum (CLEF) allow a split of the topics into four groups according to difficulty. The easy topics have a larger impact especially for multilingual retrieval. Nevertheless the internal test reliability as measured by Cronbach's Alpha is higher for more difficult topics. We can show how alternative, robust measures like the geometric average distribute the effect of the topics more evenly.

## 1 Introduction

Information Retrieval systems require the interaction between complex systems like weighting schemes and natural language processing. The resulting quality of a search system cannot be predicted because of this complexity and because components have a different effect for each language, document collection or usage context. Reliable evaluation is crucial on the other hand in order to optimize systems. The quality of information retrieval systems is most often evaluated based on a well accepted methodology [BV05]. Comparable conditions for systems are created by providing reference collections of text documents for which topics are constructed. These topics need to represent some potential user information need. The resulting documents of the competing systems are pooled and the documents in the pool are judged for relevance by human jurors. This method is called the Cranfield paradigm and forms the model for most of the scientific comparisons of search systems. In the last two decades, a large number of evaluation results have become available. They form the basis for research on the reliability of the Cranfield paradigm. The highest cost in the Cranfield evaluation paradigm is associated with the human effort to create relevance judgments. Most parameters within the model have been well researched over the last years.

Questions driving this research were: Are relevance judgments worth the resources dedicated to them? Do they lead to reliable system comparisons? Can fewer judgments also lead to the same results? The role of the human jurors has been explored by measuring their subjectivity. The interrater reliability can be shown to affect the absolute performance values but it only marginally modifies the ranking of the systems. Effort can be saved either by having fewer topics or by judging fewer documents per topic. Obviously, if more topics are developed, the reliability of the results is higher. Research needs to find an optimal balance between reliability of a test and the cost involved.

It has often been observed that the variance between topics is substantial. That means, both very easy and very hard topics typically occur in an evaluation set. The deviation between systems are mostly much smaller. Systems perform similarly whereas topics vary greatly. This observation has led to several approaches for improving systems. On one hand, systems should mainly optimize for the difficult topics which might have a larger impact on user satisfaction than easy topics. On the other hand, systems should try to guess which topics might be difficult and apply appropriate methods to them.

In this paper, we propose a new way of looking at evaluation results with regard to topic difficulty. This method is applied to the results of the Robust Task 2006 of the Cross Language Evaluation Forum (CLEF). We split the available topics into four groups according to their difficulty. The difficulty of a topic (search task) is understood as the mean or maximum average precision for this topic over several competing systems. This measure is not always intuitive and does typically not correlate with the number of relevant documents overall found for that topic.

The split proposed in this paper allows to measure the correlation of each subset with the results of the full set and to measure the internal test reliability of a test based solely on the subset. This analysis shows that the correlation between the full result and the ranking based on the easiest topics which are best solved by systems is usually higher than the correlation between the complete set and the harder topics. This is especially true for the multilingual sub-task.

## 2 Related Work: Evaluation Research

The evaluation of retrieval systems is directed towards a reliable ranking of systems in order to show which approaches and components work well for a particular retrieval task. The ranking of the systems is desired to be valid and reliable. The best system should also work well under real conditions.

The number of topics has often been an issue of discussion. Detailed analysis with small subsets of available retrieval results has led to the conclusion that 50 topics can produce a reliable result [BV02] and even 25 topics are sufficient [SZ05]. An approach to estimate how many topics are necessary is the leave-out technique. The topic set is subsequently reduced and the ranking of the systems is calculated based on the remaining subset. As long as the ranking remains stable and not too many position changes occur, one assumes that the reduced topic set would have also been sufficient.

Such an approach assumes that the ranking based on the complete set of topics represents the ground truth. We also take this approach in the analysis presented in the following section. However, it needs to be noted that this goodness measure for topics is a problematic assumption. Presumably, each topic tests some features of the system. These features may be how well a systems handles frequent or infrequent words, how it deals with topics with few or many hits, how it handles named entities or how well it deals with specific issues of stemming. Each of these capabilities may be relevant during the real use of the system and therefore each topic could be considered as a valid test for a system. In practice, it is of course unknown which features a topic would test. It typically is the case that some topics agree more on the ranking of the systems than others. Some correlate more than others and might be redundant. Network analysis techniques have been applied to identify the authoritativeness of a topic [MR07]. Identifying an optimal topic set would require more knowledge on the usage scenario as well as the type and frequency of topics during real use.

There is also research which calls for more topics but a smaller pool which contains much fewer documents [CAS07]. Such a shallow pool requires less human effort for relevance assessment, nevertheless, increasing the number of topics is supposed to boost reliability. The human effort could even be further decreased if it is directed toward topics and documents which allow a better ranking of systems during the assessment [MWZ07]. This idea is based on the observation that it would not make sense to judge a document which all systems have retrieved because this document does not discriminate between systems. Further research is necessary to investigate if that is the case for many evaluation settings. Many different performance measures have been suggested for information retrieval. In recent years, binary preference (BPref) has been widely adopted as a metric for test designs where only a small portion of the documents can be assessed [BV04]. Often, alternative measures correlate highly. Then one might argue that new measures are not necessary. However, when the results in the system rankings differ strongly, then these metrics measure different aspects of retrieval systems [Ro06]. It is not yet well understood what these aspects are and in which cases the use of a variety of measures makes sense.

A measure for the reliability of information retrieval tests has been suggested from the perspective of social science test theory [BL07]. The measure Cronbach's Alpha is a variance based value which uses the results of all base experiments for calculating the overall reliability of a test. Values above 0.8 are typically considered as indicators for reliable tests and indicate that the test is internally consistent and does not measure several factors. The values for various experiments within the Text Retrieval Conference (TREC) lie between 0.85 and 0.93 [BL07]. For a retrieval test, Cronbach's Alpha relates the sum of the variance of the systems with the variance of the AP sum for each topic. It will also be used in the following sections.

Retrieval systems as well as evaluation measures are desired to be robust. Robustness can have many meanings in everyday life. A general definition is the following one: "Robust … means … capable of functioning correctly, (or at the very minimum, not failing catastrophically) under a great many conditions" (reference.com).

For an information retrieval system there are many conditions or contexts of use. Consequently, robust IR means the capability of an IR system to work well (and reach at least a minimal performance) under a variety of conditions (topics, difficulty, collections, users, languages …). The evaluation of robust retrieval has been motivated by the fact that the variance for topics has been very large even for top performing systems. Even these good systems achieve only poor results for some topics. Improving on these topics would greatly enhance their overall quality as perceived by the user. Users remember poor performance often better than excellent performance. It is important to "ensure that all topics obtain minimum effectiveness levels" [Vo05]. In the special case that the average precision for one topic is 0, a value of 0.0001 is assumed.

As pointed out by Robertson, the geometric mean can also be considered as the mean of the logarithms of the basic results [Ro06]. Instead of the logarithm, another non-linear transformation could be used to stress the importance of the difficult topics. At the moment, the decision is arbitrary. In the long term, the transformation should be based on the observation of user behavior.

In an analysis with the smaller subsets of topics and a correlation analysis with the overall result of multilingual retrieval runs, it was shown that robust evaluation measures lead to different results than traditional measures [MW08]. Further analysis of the topic difficulty and its influence on the overall retrieval results are necessary.

In this paper, we propose and use a method for analyzing retrieval results in order to detect the relation between the difficulty of a topic and its influence on evaluation results. The same method is applied for a scenario with a robust evaluation measure. That way we can show that the geometric mean is a better aggregating measure especially for multilingual retrieval.

## 3 Context: Robust Task at the Cross Language Evaluation Forum

An evaluation track for robust retrieval has been established at the Text Retrieval Conference (TREC). This track does not only measure the average precision over all queries but also emphasizes the performance of the systems for difficult queries. In order to perform well in this track is more important for the systems to retrieve at least a few documents for difficult queries than to improve the performance in average [Vo05].

The robust task is user oriented because users often remember bad topics better than positive experiences. In order to allow a system evaluation based on robustness, more queries than for a normal ad-hoc track are necessary. The concept of robustness was extended in TREC 2005. Systems need to perform well over different tasks [Vo05].

The Cross Language Evaluation Forum (CLEF, www.clef-campaign.org) is a large European evaluation initiative which is dedicated to cross-language retrieval for European languages [Ma08, AN09]. A robust task has been organized for the first time at CLEF 2006 [Ma06].

The robust task uses test collections previously developed at CLEF. These collections contain data in six languages (Dutch, English, German, French, Italian and Spanish) and were almost constantly used at CLEF 2001, CLEF 2002 and CLEF 2003. There are approximately 1.35 million documents and 3.6 gigabytes of text in the collection. The robust task at CLEF 2006 received 133 runs by eight groups for ten sub-tasks. These tasks included monolingual retrieval as well as cross-lingual retrieval where query and document language are different from each other [Nu07, MW08]. For the multilingual task, the participants chose one query language and need to search all document collections. The result set needs to rank documents in all languages in one list. The usage scenario is a searcher who can read in a foreign language but who can form a more efficient query in his mother tongue.

## 4 Method

The following methodology was applied. From the robust task, all sub-tasks with at least ten submitted experiments (runs) were selected for the analysis. For the bilingual tasks, not enough experiments were submitted. If more than ten runs were available for a task, only the best ten were included in the analysis. Most sub-tasks contain some runs which need to be considered as negative outliers. This happens when research groups accidentally do not index the full collection or try some very different approaches for their experiments. These outliers are eliminated. The goal of retrieval experiments is to find the best system and not the perfect ordering for weak experiments. For all these runs, the performance of each system for each topic was known. Each sub-task is based on a different document collection and a translated version of the same topics.

In the next step, the topics were ordered according to their difficulty. For that, we adopted the most often used definition of topic difficulty: the mean average performance of all systems for that one topic [EGK02, Kw05]. Other measures like maximum and minimum performance for that topic and variance over systems for the topic as a potential measure for room for improvement were tried. The topic orderings based on these measures did not lead to consistent results. In future studies, other orderings based on non-linear transformations favoring the systems performing best for that topic could be tested.

The correlations between these topic orderings for individual sub-tasks are very low and not even weak. The largest absolute value is a correlation of 0.23 between multilingual and monolingual English. Most other values lie below 0.1. This confirms previous findings that topics are not difficult inherently but only in connection with a particular collection. After the ordering of the topics, four groups of topics based on difficulty were formed. For each of these quartiles of topics, a ranking of the systems was established. Because the robust task comprises 100 topics, an analysis with four sub-groups of topics is possible. The split leads to a size of subsets of topics which still has a considerably high reliability. This will be shown by the calculation of the internal test reliability.

The rankings of the systems were subsequently compared. They were determined for each of the four groups once by the mean average of the results for the 25 topics and once by the geometric mean. The rankings within the group were then compared to the ranking based on the full set by the mean average and the geometric average respectively. We also were interested in the correlation between the mean average (MAP) and the geometric average (GMAP) for the full topic sets.

Finally, the internal test reliability was calculated by Cronbach's Alpha measure. This was done for the full and the partial topic sets. It should be noted that there is only one Cronbach's Alpha and not one for MAP and one for GMAP. Cronbach's Alpha is based on the individual observations and MAP and GMAP can be seen as different ways to reach one quality measure through different aggregation of the individual observations.

## 5 Results and Discussion

The main observation from the resulting ranking comparison is that the easiest topics correlate better with the full ranking than harder topics. This is especially obvious for the multilingual sub-task. As figure 1 shows, the correlation is lowest for the hardest topics and higher for each following quartile of topics. It is obvious, that the easy topics have a higher influence on the final result or they have a higher goodness in correlating well with the full result. In a situation where the topic difficulty would be known before the evaluation and a reduction of the set would be necessary due to resource limitations, only easy topics would be selected. The easy tasks would determine the evaluation results. This seems to be negative for the evaluation as a whole. Hard topics are more likely to have a high impact on user satisfaction. Bad results lead to unsatisfied users. From the system developer's point of view, system failures for hard topics can give more insight into weaknesses of systems [HB04].

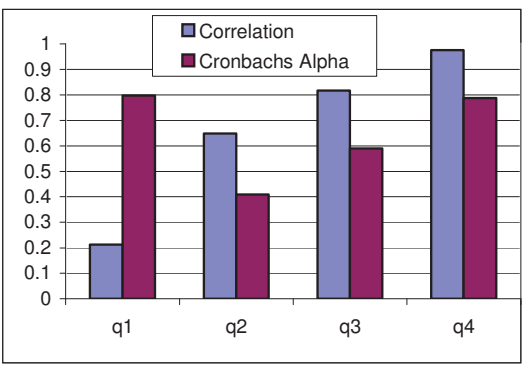



Figure 1: Correlation and Cronbach's Alpha for the Multilingual sub-task

As mentioned in the state-of-the-art overview above, robust measures can capture this better than the mean average of all topics. Figure 5 illustrates this effect. When the geometric mean (GMAP) is used to aggregate the results, the correlation between the subsets and the full set of topics is much more evenly distributed over the four quartiles. In addition, the correlation strength is lower for each set. The proposed analysis method illustrates the strength of the effect and the advantage of using the geometric mean of the individual observations.

The phenomenon observed for the multilingual sub-task cannot be seen as clear for the monolingual tasks which were included in the analysis. Figures 2, 3 and 4 show that the easiest quartile exhibits at least the second strongest correlation over all sub-tasks. The highest correlation always occurs in one quartile in the easier half. This shows that easier topics also earn a higher goodness for these tasks.

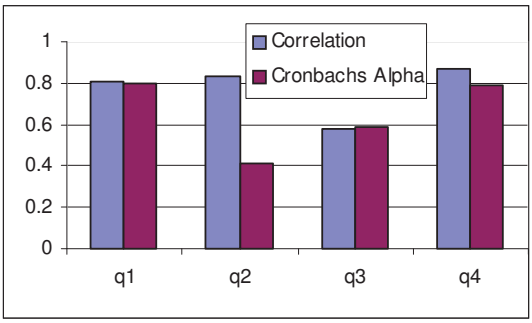



Figure 2: Correlation and Cronbach's Alpha for the Monolingual English sub-task

At the same time we can observe another interesting trend. The subset with the hardest and the easiest topics always have the best internal test reliability. The topics in the middle range are not so well suited to discriminate between systems. This finding needs to be considered in the context of the previous observation that hard topics do not deserve enough attention. At the same time they are good in discriminating between systems. Obviously there is a difference between the sub-tasks concerning the variances between the individual observations.

Why do the subsets for the mono- and multilingual tasks exhibit different phenomena? Cross-lingual retrieval requires an additional translation step. Typically, the query is automatically translated into the document languages before retrieval is initiated. This leads to lower performance. Due to the errors associated with machine translation, a decrease of about 10% to 20% in MAP is typically observed. That means, for the most often used definition of topic difficulty there are more hard topics in the multilingual retrieval set. This difference in the distribution of the average precision values for the topics is shown in figure 6. These topics might not be hard inherently but due to the necessary translation. When there are more hard topics in the set, the aggregation method is of more importance. Mean and geometric average for the multilingual task lead to more diverging results than for the monolingual tasks.

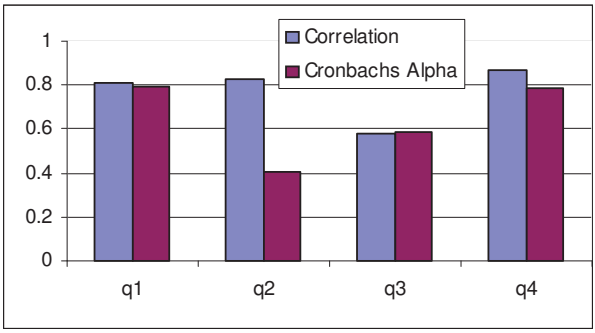



Figure 3: Correlation and Cronbach's Alpha for the Monolingual Spanish sub-task

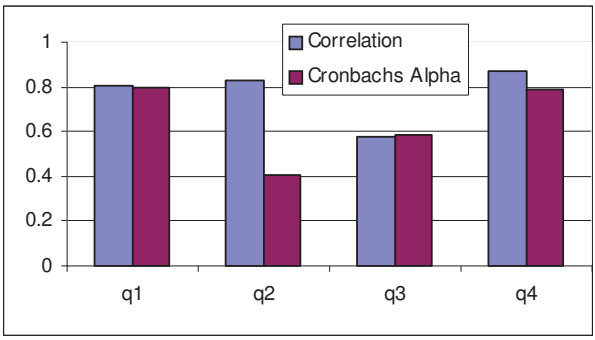



Figure 4: Correlation and Cronbach's Alpha for the Monolingual French sub-task

## 6 Conclusion and Future Work

The analysis presented here hints that an evaluation of multilingual retrieval focusing on robustness leads to substantially different results than standard evaluation measures. The analysis showed that the topic difficulty in cross-lingual retrieval is distributed in a manner that robust measures like the GMAP need to be considered. The influence of a topic set is measured as the correlation with the full set in the proposed method. In future studies, alternative measures should also be considered.

The current robust task (at CLEF 2008 and 2009) analyzes the effect of word sense disambiguation data on the robustness. For that end, annotated collections and topics are provided. Systems can extract the knowledge that a word is ambiguous and what meaning it has in each sentence. The effect of this information on the retrieval quality is currently being analyzed [AN09].

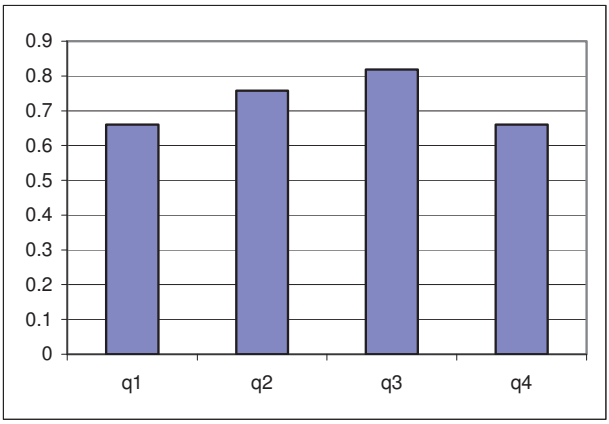



Figure 5: Multilingual Task: Correlation of the GMAP measure between small sets and full set

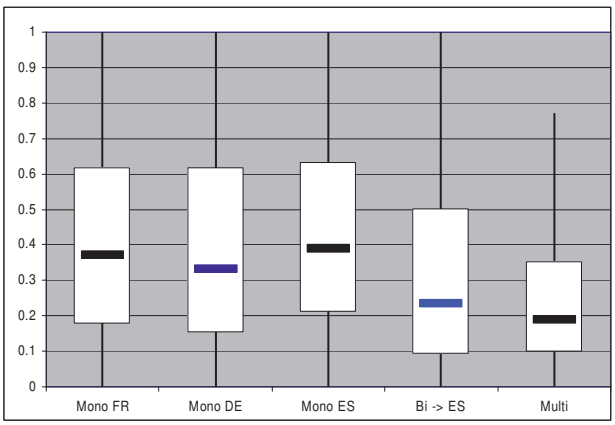



Figure 6: Distribution of Topic Difficulty for Several Tasks [MW08]

## References

[AN09] Agirre, E.; Di Nunzio, G.; Ferro, N.; Mandl, T.; Peters, C.: CLEF 2008: Ad Hoc Track Overview. In: Evaluating Systems for Multilingual and Multimodal Access: 9th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, Aarhus, Denmark, Revised Selected Papers. Berlin et al.: Springer [LNCS]. *To appear.*

[BL07] Bodoff, D.; Li, P.: Test Theory for Assessing IR Test Collections. In: Proc Annual Intl. ACM Conf. on Research and Development in Information Retrieval (SIGIR) Amsterdam. 2007. 367-374.

[BV02] Buckley, C.; Voorhees, E.: The Effect of Topic Set Size on Retrieval Experiment Error. In: Proc Annual Intl. ACM Conf. on Research and Development in Information Retrieval (SIGIR) Tampere, 2002. 316-323.

[BV04] Buckley, C.; Voorhees, E. M.: Retrieval Evaluation with Incomplete Information. In: Proc Annual Intl ACM Conf. on Research and Development in Information Retrieval (SIGIR) 2004. 25-32

[BV05] Buckley, C.; Voorhees, E.: Retrieval System Evaluation. In: TREC: Experiment and Evaluation in Information Retrieval. Cambridge & London: MIT Press. 2005. 53-75.

[CAS07] Carterette, B.; Allan, J.; Sitaraman, R.: Minimal test collections for retrieval evaluation. In: Proc 29th Annual Intl. ACM Conf. on Research and Development in Information Retrieval (SIGIR) Seattle. 2006. 33-40.

[EGK02] Eguchi, K., Kando, N. and Kuriyama, K.: Sensitivity of IR Systems Evaluation to Topic Difficulty. In: Proc. of the Third International Conference on Language Resources and Evaluation (LREC 2002) 585-589.

[HB04] Harman, D.; Buckley, C.: The NRRC reliable information access (RIA) workshop. Proc 27th Annual Intl. Conf. on Research and development in information retrieval (SIGIR) 2004. 528-529.

[Kw05] Kwok, K.: An Attempt to Identify Weakest and Strongest Queries. In: SIGIR Workshop Predicting Query Difficulty. 2005. http://www.haifa.il.ibm.com/sigir05-qp

[Ma06] Mandl, T.: Benutzerorientierte Bewertungsmaßstäbe für Information Retrieval Systeme: Der Robust Task bei CLEF 2006. In: Effektive Information Retrieval Verfahren in Theorie und Praxis: Proc d. Fünften Hildesheimer Evaluierungs- und Retrievalworkshop (HIER 2006) http://web1.bib.uni-hildesheim.de/edocs/2006/519937899/meta/

[Ma08] Mandl, T: Recent Developments in the Evaluation of Information Retrieval Systems: Moving Toward Diversity and Practical Applications. In: Informatica – An International Journal of Computing and Informatics vol. 32. 2008. pp. 27-38.

[MW08] Mandl, T.; Womser-Hacker, C.; Ferro, N.; Di Nunzio, G.: How Robust are Multilingual Information Retrieval Systems? In: Proc. ACM SAC Symposium on Applied Computing. Fortaleza, Brazil. 2008. pp. 1132-1136.

[MR07] Mizzaro, S.; Robertson, S.: HITS hits TREC – exploring IR evaluation results with network analysis. In: Proc. 30th Annual Intl. ACM Conf. on Research and Development in Information Retrieval (SIGIR) Amsterdam. 2007. 479-486

[MWZ07] Moffat, A.; Webber, W.; Zobel, J. Strategic System Comparisons via Targeted Relevance Judgments. In Proc. 30th Annual Intl. ACM Conf. on Research and Development in Information Retrieval (SIGIR) Amsterdam. 2007. 375-382.

[Nu07] Di Nunzio, G.; Ferro, N.; Mandl, T.; Peters, C. CLEF 2006: Ad Hoc Track Overview. In: 7th Workshop of the Cross-Language Evaluation Forum, CLEF 2006, Alicante, Spain, Revised Selected Papers. Berlin et al.: Springer [LNCS 4730] 2007. 21-34.

[Ro06] Robertson, S.: On GMAP: and other transformations. Proc 15th ACM Intl. Conf. on Information and Knowledge Management (CIKM) Arlington, VA. 2006. pp. 872-877.

[SZ05] Sanderson, M.; Zobel, J. Information Retrieval System Evaluation: Effort, Sensitivity, and Reliability. In Proc 28th Annual Intl. Conf. on Research and Development in Information Retrieval (SIGIR) Salvador, Brazil. 2005. 162-169.

[Vo05] Voorhees, E. The TREC robust retrieval track. ACM SIGIR Forum 39 (1) 2005. 11-20.

# SQL à la Carte – Toward Tailor-made Data Management

Marko Rosenmüller[1], Christian Kästner[1], Norbert Siegmund[1], Sagar Sunkle[1], Sven Apel[2], Thomas Leich[3], and Gunter Saake[1]

[1]School of Computer Science, University of Magdeburg, Germany
[2]Department of Informatics and Mathematics, University of Passau, Germany
[3]METOP GmbH, Magdeburg, Germany
[1]{rosenmue,ckaestne,nsiegmun,sunkle,saake}@ovgu.de,
[2]apel@uni-passau.de, [3]leich@metop.de

**Abstract:** The size of the structured query language (SQL) continuously increases. Extensions of SQL for special domains like stream processing or sensor networks come with own extensions, more or less unrelated to the standard. In general, underlying DBMS support only a subset of SQL plus vendor specific extensions. In this paper, we analyze application domains where special SQL dialects are needed or are already in use. We show how SQL can be decomposed to create an extensible *family of SQL dialects*. Concrete dialects, e.g., a dialect for web databases, can be generated from such a family by choosing SQL features à la carte. A family of SQL dialects simplifies analysis of the standard when deriving a concrete dialect, makes it easy to understand parts of the standard, and eases extension for new application domains. It is also the starting point for developing tailor-made data management solutions that support only a subset of SQL. We outline how such customizable DBMS can be developed and what benefits, e.g., improved maintainability and performance, we can expect from this.

## 1 Introduction

The *Structured Query Language* (SQL) is the de facto standard for accessing database systems. Since its introduction in the 70s and its first standardization in 1986, SQL has grown enormously in size and complexity [CW00]. From conceptually simple queries like the standard selection-projection-join operations and aggregation, SQL contains an ever growing number of further constructs pertaining to new areas of computing, e.g., for multi media or stream processing [CW00, SBc+07]. As a result, special dialects or subsets of SQL have been proposed or standardized, e.g., Structured Card Query Language (SCQL) for smart cards [Int99], TinySQL for sensor networks [MFHH05], or StreamSQL and Oracle's CQL for stream processing [ZJM+08]. All major database vendors conform only to parts of the ISO/ANSI SQL standard and often provide proprietary extensions for their systems. For developers it is obvious to add new extensions or at least to chose a custom subset of SQL, because the entire SQL standard is too large and complex for a full implementation. Because there are no guidelines for developers as which subset is adequate, this is typically an ad-hoc decision.

Currently, SQL dialects are decoupled from the SQL standard or included as extension packages. Either way, the complexity of SQL increases: a user has to analyze parallel unrelated SQL specifications to derive the actually provided functionality and many unneeded language features inherited from the base language (e.g., SQL/Foundation) are provided anyway. The resulting complexity makes it difficult to learn and use SQL. Independent of the actual task, which may be as simple as querying a single value, developers have the full arsenal ranging from simple select-from-where queries over recursive queries, object-oriented and XML extensions, control-flow statements, window functions, cube expressions, and many more. It is unrealistic to expect an application developer to know all these mechanisms and their side effects (e.g., termination of recursive queries).

We strive for a simpler, but still flexible query language on the basis of SQL. Instead of a single query language that covers all possible use cases, we aim at a family of query languages. This way, we can select a SQL dialect or even configure the query language according to the requirements of a problem domain or scenario – we select SQL features à la carte to define a concrete dialect. In a data warehouse scenario, we need a different query language than for web applications and sensor networks. For example, when we query only a single table or value, e.g., in a sensor network, we do not need the complexity caused by joins, XML extensions, or recursive queries. Furthermore, if we can exclude data types of variable length, we can use a simpler underlying database implementation. If we only allow read access, e.g., as needed for navigation systems in cars, we could even provide a SQL subset without *insert*, *update*, or *delete* statements. A read-only SQL variant could also be used to provide some form of security in the query language.

For a given scenario, an application developer can decide which SQL features are necessary and then select a database management system that provides the needed functionality or use a customizable DBMS provided for special domains, e.g., embedded systems. On the other hand, a database developer can create a DBMS matching the requirements of selected SQL features.

In this paper, we put results from prior research on extensible database systems and tailor-made data management and recent developments in software engineering into a new context. In order to realize SQL à la carte, we need a customizable query language, a customizable SQL parser and optimizer, and potentially even a customizable underlying storage system. We explain how such components can be built and how an architecture for customizable DBMS looks like.

## 2 Prior Work on Tailor-made Data Management

The need for variability is apparent in database research. There is an impressive body of research on extensions or alternative solutions of almost every single part of a DBMS or even alternative implementations of the entire DBMS from scratch. Often researchers only implement a single component and not the entire system. Many of these extensions or alternative solutions also make changes to SQL by introducing new keywords. However, extensions of the query language are mostly ad-hoc. For every extension a new syntax is

invented which is often not compatible with existing systems.

### 2.1 Tailor-Made Data Management Components

There is a large number of proposals to tailor DBMS for a special use case. We present an overview of those systems according to the five layer architecture [HR83] in order to illustrate the relation between the approaches and the DBMS architecture (see Table 1).

Layer five of a DBMS represents the data access interface. The sheer size and complexity of the SQL standard and special application scenarios are the main reasons for a number of SQL derivatives that emerged in the last 10 years. These derivatives are specialized variants for application scenarios like SmartCards [Int99], sensor networks [MFHH05], stream processing [ZJM+08], or for different programming languages [Hib08]. Parts of the query optimization have also been addressed by query optimizer generation [GD87, GM93] and specialized query engines[1], e.g., XML query engines [FH08] or query engines for embedded systems [SR05].

The next layer implements the record-oriented abstraction level of a DBMS. Specialized implementations address the transaction management system, e.g., to efficiently perform transactions in XML databases [HH07] or long-running business transactions [AMK+08]. In addition, special sorting algorithms may be used to provide minimal power consumption or to maximize performance [RSRK07].

Layer three represents access path management. Indexes have often been in the focus of research and many different tailored variants have been developed, e.g., specialized B-Trees for efficient data operations [Gra06, Gra04], R-Trees for multi-dimensional data access [Gut88], and other indexes that are applicable even in SmartCards [BBPV00].

Buffer management, which is part of layer two, also provides possibilities for optimizations. Specialized algorithms address for example embedded systems with restricted main memory and multi-media DBMS [RZ95].

Finally, variants of the file management subsystem, located in layer one, are used to support different storage hardware like in FlashDB [NK07]. To sum up, there are currently a large number of tailor-made data management components in use.

### 2.2 Tailor-Made Data Management Systems

The solutions presented above address variability in components of a DBMS. In order to build DBMS that provide different implementations of such components, there is much effort in creating extensible or customizable DBMS, in which special algorithms can be plugged. Customizability of DBMS is in the focus of database research for more than 20 years [DG01]. It aims at reusing functionality in different variants of a DBMS that are used for different application scenarios. Examples are kernel systems, which provide

[1] Query optimization cannot solely be assigned to layer five.

| Layer | Tailor-made solutions |
|---|---|
| L5 | SQL dialects or extensions (smartcards [Int99], sensor networks [MFHH05], hibernate [Hib08], stream processing [ZJM+08]), query optimizer generation [GM93], XML query optimization [FH08], lightweight query processing [SR05] |
| L4 | XML transactions [HH07], flexible transaction management [AMK+08], JouleSort [RSRK07] |
| L3 | R-Tree [Gut88], RingStorage [BBPV00], Specialized B-Tree [Gra06, Gra04] |
| L2 | Buffer management for media data [RZ95] |
| L1 | Flash Storage [NK07] |

Table 1: Exemplary tailor-made data management components according to the five layer architecture.

basic functionality and can be extended to fully functional DBMS (e.g., the EXODUS storage manager [CDF+86]), different kinds of extensible systems (e.g., user defined data types in Postgres [Sto87]), generators (e.g., the EXODUS optimizer generator [GD87]), or component DBMS [GD95]. Other approaches suggest to use very small components to ease development and reuse [CW00]. Extensibility is also achieved by commercial systems today and can be used to develop specialized systems to some degree [SBc+07].

Some approaches are based on the *software product line* concept that is understood as a general approach to build customizable software. The idea behind software product lines is to build a number of similar programs from reusable assets [PBvdL05]. These assets can be components but also smaller building blocks like source code fragments. Novel programming paradigms like *aspect-oriented programming (AOP)* [KLM+97] and *feature-oriented programming (FOP)* [Pre97] can be used to build customizable DBMS as product lines [NTN+04, TSH04, LAS05]. However, in most cases the query language cannot be modified and extensions are limited to the host system.

Furthermore, extending a single system for new application scenarios has been criticized as insufficient [SBc+07]. The reason is often that traditional RDBMS do not provide acceptable performance for special use cases. Examples are XML databases, stream processing systems, multimedia databases, data warehouses, and text databases. For these scenarios new DBMS have been developed from scratch instead of extending existing systems. This is similar in embedded systems that cannot be used with general purpose DBMS because of limited resources which also results in redevelopment of data management solutions. For example, *PicoDBMS* aims at smart cards [BBPV00] and *DELite* addresses special storage mechanisms and query processing in embedded devices [SR05]. Too often, when developing a system from scratch, a new query language dialect is invented that is incompatible with other systems.

### 2.3 Discussion

The analysis of previous work suggests that there is a need for tailor-made solutions for parts of a DBMS and also for entire DBMS. While variability in the implementation can be hidden from the user, variability in the query language can not. The user is directly confronted with different incompatible dialects. This makes it not only difficult to learn the query language, but also to exchange components or compare database implementations.

We want to address the issue of database variability from the user's perspective. Variability in the query language is needed and should be planned and coordinated. In the next sections, we propose a product line approach that allows a DBMS developer to create SQL variants tailored to the application scenario's requirements and avoids the complexity of DBMS extensions that come with their own language.

## 3 A Family of SQL Dialects

In order to provide a small, extensible subset of the SQL standard, we develop a family of SQL dialects. Each dialect of the family provides only the functionality needed for a particular application scenario. Starting with a minimal subset of SQL, e.g., that supports select-from-where queries, different SQL dialects can be created by adding functionality. For example, we can add grouping functionality, transactions, or queries needed only for special domains. The differences between the dialects can be described using the concept of *features* [CE00]. Features represent the functionality provided by a SQL dialect. For example, a feature TRANSACTIONS represents the queries needed for transaction processing (e.g., `BEGIN TRANSACTION`), i.e., it represents a number of keywords and language constructs of SQL. Similarly, in a stream processing system, a feature SLIDINGWINDOW represents support for sliding window queries. This way, the features describe the functionality of a SQL dialect in an abstract way. A concrete dialect is defined by a selection of features that describe the functionality needed for this dialect. For example, features READ, representing select-from-where queries, and SLIDINGWINDOW, used for querying streams, define a very simple SQL dialect that can be used for a stream processing system.

Based on a family of SQL dialects, a DBMS which fits to a particular SQL dialect has to be chosen or created. The functionality provided by the DBMS depends on the functionality of the actually used dialect as shown in Figure 1. For example, if the SQL dialect supports transactions, we have to parse transactional queries and the DBMS itself has to provide transactions. To create such a DBMS, a developer might implement it from scratch; however, reuse of existing functionality, e.g., by using a customizable DBMS, should be preferred. Developing such customizable solutions for a particular domain is a challenging task and will be in the focus of the next section.

In the following, we present three examples for application domains where we need different SQL dialects. We integrate these scenarios in an initial family model of SQL. In Section 4, we then present approaches to build DBMS that can be used to process a number of similar SQL dialects for differing application scenarios.

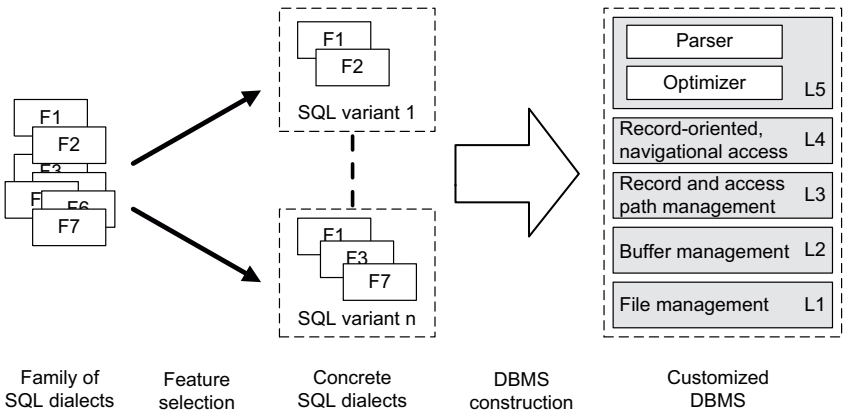



Figure 1: Defining SQL variants based on the selection of features. A DBMS has to implement the functionality to process queries of a particular variant.

### 3.1 Examples for SQL Dialects

As examples we chose web databases, sensor networks, and stream processing systems. We present an overview which functionality of SQL is needed for each scenario and illustrate how they can be integrated in a domain model.

**Web Databases.** Web databases, i.e., systems needed to store data for simple web applications, usually only require a small subset of SQL including selection-projection-join queries, aggregation, transactions, etc. Mechanisms like stored procedures, triggers, prepared statements, or views are usually not needed. An example often used for web development is MySQL. In earlier versions, MySQL was often used for web development, despite its lack of advanced features like transactions or stored procedures. During its evolution, such features were added, however, as reaction, some MySQL developers forked a new release called Drizzle[2] that slims down the MySQL code to a small lightweight data-management solution with SQL but without some of the advanced features. Developing such a slim web DBMS has a number of benefits. Examples are easier performance optimizations, reduced maintenance, reduced implementation effort, and easier development of client applications. This is especially important for web applications where developers are usually no database experts. In contrast to high-end systems like Oracle or DB2, web databases are often small systems that are free of charge and easy to install and maintain. Supporting only a subset of the DBMS functionality allows a vendor to offer cheap DBMS for this market segment and gives even end users the chance to maintain the system.

**Sensor Networks.** Similar to the web database example, sensor networks only require a small subset of SQL, e.g., as provided with TinySQL [MFHH05]. Basic functionality

[2]https://launchpad.net/drizzle

includes select-from-where queries. However, also new query types are required. For example, the TinySQL dialect explicitly supports spatial queries to access only nodes in a user defined region and temporal queries to access data for a given time interval that are not supported by general purpose DBMS.

**Stream Processing.** The last example is a stream processing DBMS that also uses special SQL extensions, e.g., StreamSQL or Oracle's CQL [ZJM+08]. In these systems, multiple streams might be supported and aggregation functions are usually executed on windows defined on the data stream. Support for multiple streams is not necessary for every stream processing application and might be omitted to simplify the query engine. Furthermore, different window definitions are possible. Since most stream processing applications use only sliding windows, there might be a simple SQL dialect that provides only this type of window. Thus, even in a single domain, different SQL dialects might be needed.

### 3.2 A Domain Model of SQL

The examples above show that each dialect provides certain queries that are not required for others. Basic functionality is needed in all dialects, e.g., select-from-where queries, but even functionality like simple data manipulation is not always needed (e.g., *insert* in a stream processing scenario).

In order to derive a single domain model that describes a whole family of SQL dialects, we will use features to describe the dialects and then combine the descriptions of all dialects in one domain model. Features are well known from domain analysis [KCH+90] and software product line engineering [CE00]. A feature is a distinguishable characteristic of software that is of interest to some stakeholder [CE00]. It describes the differences and commonalities between systems in a domain. Features and dependencies between features are depicted/organized in a hierarchical way in *feature diagrams*.

The feature concept can be used to describe differences in the domain of query languages which results in a description of a family of SQL dialects. In Figure 2, we define an initial feature diagram on the basis of differences and commonalities of the three scenarios discussed in Section 3.1. The root of the tree represents the described concept and the remaining nodes are features. For example, feature TRANSACTIONS represents functionality needed for transaction processing. A feature can also have sub-features (i.e., child nodes), e.g., feature CORE combines features READ and WRITE which represent basic functionality. Optional features are denoted with an empty dot (e.g., WRITE) meaning that the feature is not needed for every SQL dialect. In contrast, mandatory features, denoted with a filled dot, are part of every variant. Between grouped features (e.g., subfeatures of CORE) there can be OR-relations, which means that the features can be selected in combination but at least one of the features has to be selected.[3]

The SQL dialect needed for a web database can now be defined by listing the needed features, i.e., DATATYPES, STOREDPROCEDURES, TRANSACTIONS, JOIN, AGGREGATION, and all

[3] There are further relations between features (e.g., XOR) not used here.

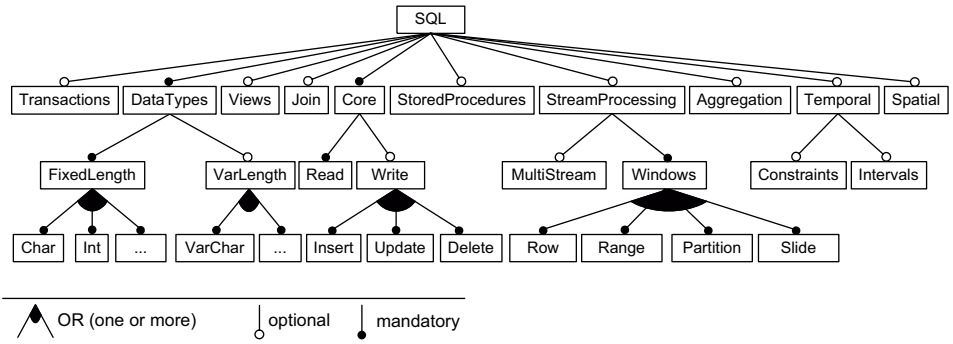



Figure 2: Feature diagram for a subset of SQL that supports core functionality and several extensions.

the CORE sub-features. Similarly, we can describe SQL dialects for a sensor network using the sub-features of CORE as well as spatial and temporal queries (provided by features SPATIAL and TEMPORAL). For sensor networks that do not provide position information, also dialects without spatial queries are supported by omitting the optional feature SPATIAL. Finally, a SQL dialect for stream processing should provide core functionality without write support. Stream extensions are provided by feature STREAMPROCESSING with optional feature MULTISTREAM (representing operations on multiple streams) and feature WINDOWS which summarizes different window functions. A small dialect might provide only sliding windows and omit support for other window types. The OR-relation between window types means that all combinations of windows might be selected to derive a valid SQL dialect, but at least one has to be selected. This is an important requirement since it avoids invalid SQL dialects without any window operations in a stream processing setting.

OR relations are not the only dependencies between features. There can be additional dependencies like *requires* or *excludes* relationships between features that avoid invalid feature combinations. Such dependencies can be inevitable (*domain dependencies*) or caused by the implementation, e.g., to simplify the development of a DBMS. For example, TEMPORAL queries might require a DATETIME data type. In Figure 2, such dependencies are not shown because most of them depend on the concrete implementation of a DBMS.

## 4 Processing the Query

With a family of SQL dialects we can tailor the query language to a given scenario. However, a tailor-made query language itself has little value, but it has to be backed up by an underlying DBMS that can process the queries. In this section, we discuss the implications of tailoring query languages for the underlying DBMS and propose three approaches how to cope with the variability of the query language.

### 4.1 Requirements on the DBMS

Depending on a given SQL dialect, the underlying DBMS has to provide the corresponding functionality to process the queries for that dialect. We propose that a given DBMS does not support all SQL dialects (but still potentially many dialects), but is tailored to a specific dialect. For example, if a dialect supports aggregation, the query engine has to understand, optimize, and execute aggregation queries. Such changes not only affect the query processor but potentially several other layers of the DBMS. In Table 2, we give an exemplary overview of potential changes required to implement Aggregation, Views, Write-Access, and Joins (cf. Fig. 2). Its purpose is to illustrate that certain features of the query language can have consequences for the implementation of the entire DBMS. The table is neither complete regarding necessary extensions to a DBMS, nor does it cover all features of SQL.

From Table 2 it can be seen that changes required by different dialects cut across the entire DBMS architecture. First, the queries have to be *parsed* and *translated* into a query execution plan. Hence, a parser is needed that can parse the queries defined by a specific dialect. For example, if a dialect does not support aggregation we do not have to parse `GROUP BY` statements nor do we have to consider aggregations in query execution plans. Next, many optimizations are only needed for special dialects. For example, if joins are not present in a dialect, query optimization can be strongly simplified and must consider only lists instead of trees. Similarly, if aggregation is supported, the optimizer must consider adequate algebraic rules and cost optimizations.

Similar changes can be found in other parts of a DBMS. For example, algorithms for aggregation and joins have to be provided if these features are selected on the language level and also specialized access paths for joins have to be considered depending on the SQL dialect. Even the lowest layers can be affected, as in those cases where write operations or variable length data types are not needed in which all layers can be simplified. The changes needed for a SQL language feature often affect the entire DBMS implementation.

### 4.2 Toward Tailor-made DBMS

If SQL features can cut across the entire DBMS, how can we map a family of SQL dialects to a DBMS implementation? There are several possible scenarios, which all provide benefits over the current status quo of a huge SQL standard and uncontrolled, incompatible, specialized SQL dialects:

- In the first scenario, an existing DBMS is mapped to one of the SQL dialects. For example, if we have a DBMS for sensor networks, the DBMS developer could ex-post describe the supplied language features as a feature selection of the SQL language model. Our sensor DBMS could support features CORE, SPATIAL, and TEMPORAL (cf. Fig. 2). This information can be used for communication with stakeholders and customers. It allows for comparing a specific DBMS with other implementations and a customer can easily check if her functional requirements are met by a given implementation.

| Layer | | Aggregation | Views | Write-Support | Joins |
|---|---|---|---|---|---|
| Data Access | Parsing | 'group by', 'min', 'avg', ... | 'create view', ... | 'insert', 'update', 'delete', ... | 'join', multiple columns, ... |
| | Translation | introduce grouping operator $\gamma$ in relational algebra | resolve views | | introduce $\bowtie$ operator in relational algebra, ... |
| | Optimization | algebraic rules & heuristics for $\gamma$, cost estimations for $\gamma$ | view materialization | | several algebraic rules & heuristics, cost estimations, select join strategy & algorithm, dynamic programming, ... |
| Operations & Access Path | | operations for sorting, min, avg, ... | | write operations (data and all access paths) | join algorithms, join indexes, ... |
| Page management | | temporary pages | | TID concept, segment space management | |
| Buffer management | | temporary buffer pools | | dirty page management | |
| OS | | | | write support | |

Table 2: Impact of SQL language features on the DBMS architecture.

The benefit of this scenario is its power in communicating differences and commonalities of DBMS between stakeholders.

- When implementing a new DBMS, developers chose to support a subset of SQL language features, i.e., one or more SQL dialects as specified by the feature model. For example, if we want to implement a sensor network DBMS, a DBMS domain engineer could ex-ante define the SQL language features which she wants to support. This way, we would not have to define an own SQL subset or even proprietary SQL extensions, but could reuse this family of SQL dialects. Developers are not directly confronted with the full SQL standard, but have clearly defined subsets that appear feasible to implement.

  The main benefit of this scenario is that developers can reuse an existing decomposition of the SQL standard. This reduces the likelihood that they invent their own language or implement just an arbitrary, potentially even undocumented SQL subset when they are faced with the overwhelming full SQL standard. Together with the first scenario, this reduces the overall complexity compared to uncontrolled, proprietary SQL dialects.

- Finally, it is also possible not to map a single system to one or more SQL dialects, but to introduce variability in the DBMS implementation as well. If we have a flexible, extensible family of DBMS systems, e.g., developed from core systems [DG01], component systems [Sel08], or software product lines [NTN+04, TSH04, LAS05, KAB07, RSS+08], we can directly map variability of the query language to variability of the query processor's implementation. Such approaches allow a developer to reuse functionality when implementing a number of similar DBMS and bear the potential of decreasing development costs and time-to-market and increasing software quality. This, however, does not necessarily mean that we want to develop a product line of DBMS that can match all possible SQL dialects, ranging from embedded systems to data warehouses. Such "one size fits all" approach is technically extremely challenging, if possible at all. Nevertheless, it is possible to tailor a family of database systems for certain domains, e.g., different embedded scenarios. Such a variable DBMS would not support all dialects, but a larger subset of dialects.

In the remainder, we aim at the third scenario, because it raises the most challenges, but also provides the most opportunities. We discuss how to implement customizable DBMS to match the requirements of multiple SQL dialects and not only a single dialect. We present an overview following the five layer DBMS architecture top-down.

### 4.2.1 Parsing SQL Dialects

Parts of a DBMS depend on the functionality of the supported SQL dialect, they are only needed if a particular feature of the family of SQL dialects is used. In contrast, a SQL parser is needed for every DBMS and functionality of the parser does not significantly differ for different SQL dialects, except that different parts of the SQL grammar have to be parsed. Hence, reuse of parser code for different SQL dialects is desirable and easy to

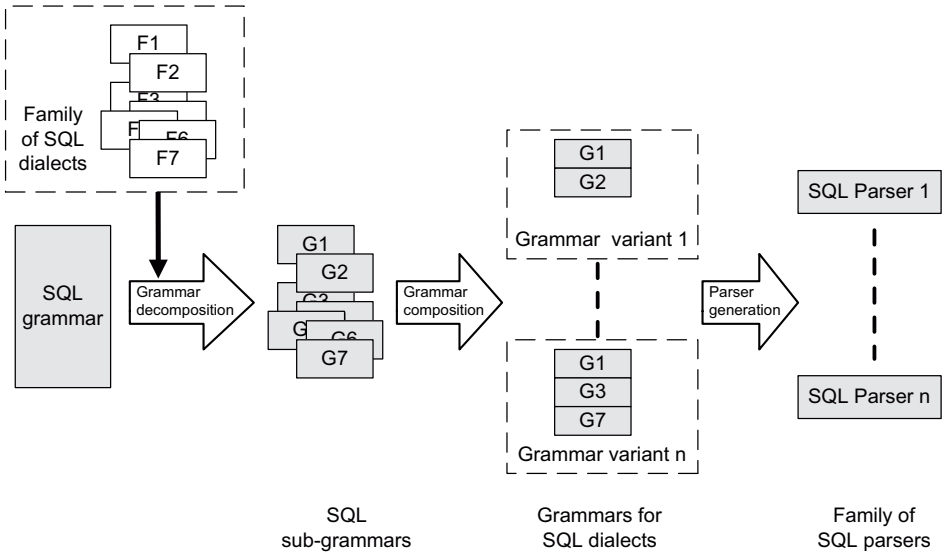



Figure 3: Generating a family of SQL parsers by decomposing the SQL grammar.

achieve. SQL parsers can be created using parser generators that receive a SQL grammar as input and generate a program that can parse SQL queries according to this grammar. To create a parser for a particular dialect, we need the grammar for that dialect and thus can automatically generate the according parser.

In order to create a grammar according to a SQL dialect, an approach for grammar composition can be used as shown in Figure 3. The idea is (1) to decompose a full SQL grammar into sub-grammars according to the features derived from the domain analysis of the SQL standard and (2) to compose sub-grammars on the basis of the features a user selects for a specific SQL dialect. Composed grammars are then used to generate SQL parsers using parser generators.

In [SKS+08, Sun07], we have presented a decomposition of SQL:2003 and described an approach to generate variants of SQL parsers based on the language features. We used a more fine-grained decomposition of SQL than depicted in Figure 2 which was driven by the SQL syntax resulting in hundreds of features. Based on a family of SQL dialects (similarly as presented in Fig. 2), we have also decomposed the SQL:2003 grammar according to the features of the family retrieved from the standard.[4] That is, for each feature we have extracted the according part of the SQL grammar into a sub-grammar for that feature. For example, a feature AGGREGATION (cf. Fig. 2) that represents the `GROUP BY` statement and some other syntactical extensions can be extracted into a separate sub-grammar. A SQL dialect that only supports core operations and aggregation is then created by composing the sub-grammars for READ and WRITE operations (i.e., `SELECT`, `UPDATE`, etc.) and

[4]The extracted features of SQL:2003 are based on the BNF grammar specification of SQL:2003 and other information given in SQL/Foundation. We have not further decomposed extension packages like XML.

AGGREGATION and generating the parser from that composed grammar.

When processing a SQL query, a parser internally creates a syntax tree which represents the query and is the basis for generating a query execution plan. Hence, a generated parser has to be extended with functionality that translates the syntax tree into a query tree which is the representation needed for query optimization.

### 4.2.2 Query Optimization

Based on a generated parser that can be tailored to a SQL dialect, a complete SQL query processing engine can be created that processes only the queries of a chosen SQL dialect. Query optimization and execution have been one of the main challenges when creating customizable and extensible DBMS for more than 20 years. For example, the Volcano project addressed this issue and has shown that customizability is possible but complex [GM93]. These solutions use query optimizer generators to introduce variability needed for customization. When processing queries of a SQL dialect, the optimization process depends on the features selected for this dialect. As a result, transformation rules that specify the needed transformations for optimizer generators, have to be defined depending on the features of a given dialect, e.g., for grouping operations (cf. Table 2). Furthermore, whole parts of an optimizer are only needed if a particular SQL feature is included in a concrete dialect, e.g., considering materialized views during optimization is only needed if views are available in the dialect.

Novel software engineering techniques, not available 20 years ago (e.g, software product lines and feature-oriented programming), can help in developing customizable query engines and reduce the complexity of building variable solutions. This was already shown for other parts of DBMS and we currently apply this to query processors. Using a software product line approach, also allows for providing an abstract view on query processors with respect to variability and enables systematic reuse between different variants.

As a first step, customizable solutions do not have to provide full customizability according to a whole SQL family, but only parts of a query processor might be addressed. Further variability can then be introduced in a step wise manner. Customizable query optimizers might also not address the whole space of SQL in general but only smaller sub-parts, i.e., a number of similar dialects.

Because of the strong interdependence of query optimization and execution, e.g., indexing, buffer management, pages, data types, and information in the data dictionary, a customizable query processor cannot be developed without taking the underlying layers of a DBMS into account. As already shown in Table 2, there are components in all parts of a DBMS that are affected. However, customizability of the query engine does not enforce a customizable DBMS. For example, a query engine can be used for SQL dialects with or without support for Joins as long as the underlying DBMS supports the needed join algorithms. That is, for a number of SQL features we can have a fixed implementation of the lower layers of a DBMS that includes all of the features. This inhibits some of the benefits expected from a small SQL dialect (e.g, reduced functionality in all layers of a DBMS); however, it still reduces the functionality and complexity of the query processor. The other solution, a fully

customizable DBMS, is most challenging but also most promising with respect to expected benefits, i.e., a complete DBMS that perfectly fits to a chosen SQL dialect. We will analyze such customizable DBMS in the last part of this section.

### 4.2.3 Query Execution

We have shown how a customizable query processor can be developed and we will now outline how this might be continued in the lower layers of a DBMS. There are implementation techniques, e.g, using components, that achieve customizability at least for parts of a DBMS. In the following, we will not show how a concrete solution could look like, but analyze how customizability should be used in the context of creating a DBMS that can be tailored to a particular SQL dialect. There are a number of techniques for implementing this customizability and we will now review some of them.

**Components.** The idea of customizable or variable DBMS is to reuse functionality in different variants of a DBMS [DG01]. Variability provided by such systems is usually limited to exchanging parts of a DBMS that can be modularized very well, e.g, with components [Sel08]. Functionality that is scattered over the entire DBMS is hard to modularize and cannot be easily exchanged with a different implementation. This, however, is not a barrier for developing customizable DBMS in general. As already stated for query optimization, also a customizable DBMS does not have to provide all the variability provided by the SQL dialects that it implements. Benefits can already be achieved by exchanging selected parts of a DBMS. For example, to support a DBMS with and without join operations, temporary pages and temporary buffer pools can be developed as optional parts of a DBMS (cf. Table 2). Hence, choosing a SQL dialect without joins results in a DBMS without these optional components. Thus already existing component DBMS can be used to realize the underlying variable DBMS. Using components to implement the DBMS, however, this is only possible for functionality that can be modularized and is not cutting across the whole system.

**New Programming Paradigms.** Some features of a DBMS affect large parts of the source code and thus also cut across other components. In order to achieve customizability of such *crosscutting* features, these can be implemented in a modular way using new programming paradigms. For example, Nyström et al. used aspect-oriented programming (AOP) in combination with components to modularize crosscutting features that effect multiple other components [NTN+04]. Tešanović et al. used AOP to refactor the C version of Oracle's Berkeley DB to modularize crosscutting features that had been implemented with C preprocessor statements before [TSH04]. Feature-oriented programming (FOP) is an alternative programming paradigm that can be used for modularization of crosscutting features. In a number of case studies, we have used AOP and FOP to refactor and implement DBMS product lines [LAS05, KAB07, RSS+08] and have shown that FOP can be used without negative impact on performance as sometimes found with components.

**Combining a Family of SQL Dialects and Customizable DBMS.** Independent of the concrete implementation technique (e.g., using components or FOP), the product line concept can be used to generate the underlying DBMS according to the features of a SQL dialect. A DBMS product line is also described using a feature model as it is done for our family of SQL dialects. However, the features of a DBMS product line, e.g., as in the FAME-DBMS product line [RSS+08], differ from the features found in the SQL family. For example, a feature RECOVERY, representing recovery functionality of a DBMS, does not have a representation in the SQL family. Both product lines, the DBMS product line and the family of SQL dialects, can be integrated by combining their feature models, i.e., features of the SQL family have to be mapped to features of the underlying DBMS product line. For example, if a feature JOIN of our SQL family is selected, the DBMS has to provide a join algorithm which might be implemented in a feature JOIN of the DBMS. This is similar for feature TRANSACTIONS which requires a transaction management system in the DBMS. Such dependencies between the SQL feature model and a DBMS feature model can then be represented as constraints between features of both models. When defining a concrete SQL dialect by selecting the needed SQL features, we can then automatically decide which features of the DBMS are needed. Features, representing functionality not defined by the SQL family (e.g., RECOVERY), additionally have to be selected when needed to complete the DBMS configuration.

## 5 Discussion and Perspective

In the following, we discuss some issues that are important when developing tailor-made DBMS and present some perspectives for further developments.

**Granularity of a SQL Decomposition.** In Section 3.2, we have provided a coarse-grained decomposition of SQL but we could also show that a much more fine-grained decomposition is possible [SKS+08]. Finding the appropriate granularity for a SQL decomposition and answering the question what *good* SQL dialects are, are some of the challenges to be tackled in future research. One of the involved problems is the resulting complexity of a DBMS implementation which increases for a fine-grained decomposition. On the other hand, a fine-grained decomposition results in SQL dialects that better fit to application scenarios. Furthermore, we only considered SQL/Foundation as a basis for decomposition and ignored extension packages like XML. These are already separated from SQL/Foundation but may also be further decomposed.

**Extensibility of the SQL Family.** Using a family of SQL dialects and a domain model that describes the family, it is easier to understand the standard due to a structured representation and clear separation of functionality described by the model. A SQL domain model can be used to communicate DBMS functionality between developers and also customers. However, it does not avoid arbitrary uncontrolled additions to the SQL syntax for new application scenarios. Nevertheless, it is easier to align extensions with the existing model and to

achieve compatibility of dialects. Furthermore, it is easy for developers to derive a concrete dialect by deciding which SQL features have to be included. Especially when creating small dialects, e.g., for embedded systems, web databases, and also for research prototypes and teaching SQL, a listing of needed features is a simple task and ensures that the dialect is compatible with the standard. A documentation that only describes functionality of the used dialect further simplifies the use of SQL. Generating such documentation variants is possible using a product line approach.

**One Size Fits all?** Stonebraker et al. suggested that a single DBMS is not appropriate for all application scenarios [SC05]. This also applies when using a customizable DBMS, e.g., implemented with a software product line, because with increasing functionality also the complexity of the implementation increases and the reuse between different DBMS decreases due to highly differing requirements. Hence, independently of the implementation technique, a customizable DBMS will usually not cover all possible SQL dialects but only a number of similar dialects that are promising with respect to reuse of functionality. For example, there might be a number of SQL dialects for data management in embedded systems (e.g., for sensor networks and smart cards) that are significantly different, but can be addressed with one customizable and extensible underlying storage system for embedded devices. On the other hand, there might be functionality that cannot be reused between systems because of completely different implementations. For example, it is hard to achieve any substantial reuse between DBMS for embedded systems and DBMS for server systems. For that reason, there will probably be only a high degree of reuse between similar systems of a domain (e.g., embedded systems) and developing one customizable DBMS for a number of different domains is not reasonable.

Developing a customizable DBMS, i.e., a product line of DBMS, always introduces additional implementation effort to achieve the needed reuse. Because of this reuse, however, the additional effort pays off if different similar products are needed and these products do not have to be developed in parallel. Reasons are reduced maintenance, lower development costs and time, and improved software quality. Hence, when developing customizable DBMS it is questionable which dialects such a DBMS should support. Finding the correct dialects to support and thus a good balance between reuse and complexity is a key for success when developing customizable DBMS.

## 6 Conclusion

Approaches that aim at providing tailor-made data management components can be found in all layers of DBMS including different SQL dialects for special domains which are often uncoupled from the SQL standard. Work on these tailor-made solutions, however, is usually uncoordinated and each solution comes with an own SQL dialect that sometimes does not reflect the underlying functionality.

Our goal is to provide a uniform foundation for tailor-made data management starting with the query language. With a family of SQL dialects we have shown how some of the

complexity issues of SQL can be addressed by structuring the SQL standard and providing extensibility to add new language features for special domains like sensor networks or stream processing systems. We have shown how we can create concrete SQL dialects by choosing SQL features à la carte from the family of SQL dialects. Concrete dialects tailor SQL as a query language for a particular domain or even a specific application scenario.

A family of SQL dialects is only a first step for achieving a uniform basis for data management that includes all application domains. Integrating existing research that addresses variability in all parts of a DBMS is needed and a complete family of SQL dialects has to be created. Query processing and the underlying DBMS layers may then be turned into customizable systems in a step-wise manner. In the long run, even the standardization process of SQL could be based on language features and a family of dialects. There are many challenges for every part of a DBMS and all application domains; however, it allows us to overcome the complexity problems of SQL and to build tailor-made data management solutions.

## Acknowledgments

We thank Ingolf Geist and Eike Schallehn for discussions about this work. The work of Marko Rosenmüller and Norbert Siegmund is funded by German Ministry of Education and Research (BMBF), project number 01IM08003C, and German Research Foundation (DFG), project number SA 465/32-1. Sven Apel's work is funded partly by German Research Foundation (DFG), project AP 206/2-1. The presented work is part of the projects ViERforES[5] and FAME-DBMS[6].

## References

[AMK+08] A.-B. Arntsen, M. Mortensen, R. Karlsen, A. Andersen, and A. Munch-Ellingsen. Flexible Transaction Processing in the Argos Middleware. In *EDBT Workshop on Software Engineering for Tailor-made Data Management (SETMDM)*, pages 12–17. University of Magdeburg, 2008.

[BBPV00] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDMBS: Scaling Down Database Techniques for the Smartcard. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 11–20. Morgan Kaufmann, 2000.

[CDF+86] M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, and E. J. Shekita. The Architecture of the EXODUS Extensible DBMS. In *Workshop on Object-Oriented Database Systems*, pages 52–65. IEEE Computer Society Press, 1986.

[CE00] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[5] http://vierfores.de/
[6] http://fame-dbms.org/

[CW00] S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1–10. Morgan Kaufmann, 2000.

[DG01] K. R. Dittrich and A. Geppert. Component Database Systems: Introduction, Foundations, and Overview. In *Component Database Systems*, pages 1–28. dpunkt.Verlag, 2001.

[FH08] J. A. M. Filho and T. Härder. Tailor-Made XML Synopses. In *International Baltic Conference on Databases and Information Systems (DB&IS)*, pages 25–36, June 2008.

[GD87] G. Graefe and D. J. DeWitt. The EXODUS Optimizer Generator. *SIGMOD Record*, 16(3):160–172, 1987.

[GD95] A. Geppert and K. R. Dittrich. Strategies and Techniques: Reusable Artifacts for the Construction of Database Management Systems. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 932 of *Lecture Notes in Computer Science*, pages 297–310. Springer Verlag, 1995.

[GM93] G. Graefe and W. J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 209–218, 1993.

[Gra04] G. Graefe. Write-Optimized B-Trees. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 672–683, 2004.

[Gra06] G. Graefe. B-tree Indexes for High Update Rates. *SIGMOD Record*, 35:39–44, 2006.

[Gut88] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Readings in Database Systems*, pages 599–609. Morgan Kaufmann, 1988.

[HH07] M. P. Haustein and T. Härder. An Efficient Infrastructure for Native Transactional XML Processing. *Data and Knowledge Engineering (DKE)*, 61(3):500–523, 2007.

[Hib08] Hibernate. Relational Persistence for Java and .NET, 2008. http://www.hibernate.org.

[HR83] T. Härder and A. Reuter. Concepts for Implementing a Centralized Database Management System. In *Proceedings of the International Computing Symposium*, pages 28–60, 1983.

[Int99] International Organization for Standardization (ISO). Part 7: Interindustry Commands for Structured Card Query Language (SCQL). In *Identification Cards – Integrated Circuit(s) Cards with Contacts*, ISO/IEC 7816-7, 1999.

[KAB07] C. Kästner, S. Apel, and D. Batory. A Case Study Implementing Features Using AspectJ. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 223–232, 2007.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Verlag, 1997.

[LAS05] T. Leich, S. Apel, and G. Saake. Using Step-Wise Refinement to Build a Flexible Lightweight Storage Manager. In *Proceedings of the East-European Conference on Advances in Databases and Information Systems (ADBIS)*, volume 3631 of *Lecture Notes in Computer Science*, pages 324–337. Springer Verlag, 2005.

[MFHH05] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[NK07] S. Nath and A. Kansal. FlashDB: Dynamic Self-tuning Database for NAND Flash. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 410–419. ACM Press, 2007.

[NTN+04] D. Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems*, pages 1–8. IEEE Computer Society, 2004.

[PBvdL05] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Verlag, 2005.

[Pre97] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 1241 of *Lecture Notes in Computer Science*, pages 419–443. Springer Verlag, 1997.

[RSRK07] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: a Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 365–376. ACM Press, 2007.

[RSS+08] M. Rosenmüller, N. Siegmund, H. Schirmeier, J. Sincero, S. Apel, T. Leich, O. Spinczyk, and G. Saake. FAME-DBMS: Tailor-made Data Management Solutions for Embedded Systems. In *EDBT'08 Workshop on Software Engineering for Tailor-made Data Management (SETMDM)*, pages 1–6, March 2008.

[RZ95] D. Rotem and J. L. Zhao. Buffer Management for Video Database Systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 439–448. IEEE Computer Society Press, 1995.

[SBc+07] M. Stonebraker, C. Bear, U. Çetintemel, M. Cherniack, T. Ge, N. Hachem, S. Harizopoulos, J. Lifter, J. Rogers, and S. B. Zdonik. One Size Fits All? Part 2: Benchmarking Studies. In *Third Biennial Conference on Innovative Data Systems Research*, pages 173–184, 2007.

[SC05] M. Stonebraker and U. Cetintemel. One Size Fits All: An Idea Whose Time Has Come and Gone. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 2–11, 2005.

[Sel08] M. Seltzer. Beyond Relational Databases. *Communications of the ACM (CACM)*, 51(7):52–58, 2008.

[SKS+08] S. Sunkle, M. Kuhlemann, N. Siegmund, M. Rosenmüller, and G. Saake. Generating Highly Customizable SQL Parsers. In *Workshop on Software Engineering for Tailor-made Data Management (SETMDM)*, pages 29–33, 2008.

[SR05] R. Sen and K. Ramamritham. Efficient Data Management on Lightweight Computing Devices. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 419–420. IEEE Computer Society Press, 2005.

[Sto87] M. Stonebraker. The Design of the POSTGRES Storage System. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 289–300. Morgan Kaufmann, 1987.

[Sun07] S. Sunkle. Feature-oriented Decomposition of SQL:2003. Master's thesis, Department of Computer Science, University of Magdeburg, Germany, 2007.

[TSH04] A. Tešanović, K. Sheng, and J. Hansson. Application-Tailored Database Systems: A Case of Aspects in an Embedded Database. In *Proceedings of International Database Engineering and Applications Symposium*, pages 291–301. IEEE Computer Society Press, 2004.

[ZJM+08] S. Zdonik, N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, M. Cherniack, U. Cetintemel, and R. Tibbetts. Towards a Streaming SQL Standard. *Proceedings of the VLDB Endowment*, 1(2):1379–1390, 2008.

# Lernen nutzerspezifischer Gewichte innerhalb einer logikbasierten Anfragesprache

Ingo Schmitt, David Zellhöfer
Brandenburgische Technische Universität Cottbus
Institut für Informatik, Postfach 10 13 44
D-03013 Cottbus, Germany
{schmitt|david.zellhoefer}@tu-cottbus.de

**Abstract:** Unpräzise Anfragen, wie solche nach Ähnlichkeit, stellen traditionelle Datenbanken vor zwei Herausforderungen: die Integration unpräziser Prädikate und die Gewichtung dieser innerhalb ihrer Anfragelogik. Letzteres wird besonders durch das subjektive Ähnlichkeitsempfinden des Nutzers notwendig. Die vorliegende Arbeit stellt ein theoretisches Framework vor, welches Termgewichte in eine quantenlogikbasierte Anfragesprache, die den Gesetzen einer Booleschen Algebra gehorcht, integriert.

Zur Formulierung von Gewichtungen wird nutzerseitig auf das Konzept der Präferenz zurückgegriffen. Intern jedoch werden numerische Gewichtswerte verwendet, um eine definierte Reihenenfolge von Ergebnisobjekten zu erzeugen. Im Rahmen eines Relevance-Feedback-Prozesses finden Verfahren des maschinellen Lernes Einsatz, um den Nutzer bei der Gewichtssetzung entsprechend seiner Anfrageintention zu unterstützen. Experimentelle Ergebnisse unterstreichen die Effektivität des Ansatzes und zeigen deutlich, dass maschinenbasierte Lernverfahren im Rahmen des Relevance-Feedbacks dem Nutzer bei der Recherche helfen.

## 1 Einleitung

Bezüglich einer Anfrage entscheiden traditionelle, auf Boolescher Logik basierende Datenbanksysteme pro Datenbankobjekt, ob es zum Ergebnis gehört oder nicht. Diese binäre Entscheidung genügt allerdings häufig nicht den Anforderungen seitens des Nutzers. Am Beispiel der Suche nach einer komfortablen, günstigen Kamera wird deutlich, dass sich die Kategorien „komfortabel“ und „günstig“ durch ihre Vagheit kaum durch Boolesche Werte abbilden lassen. Vielmehr sucht der Nutzer hier nach Produkten, die seinen Vorstellungen hinreichend gut entsprechen, so dass sich eine spezifische Reihenfolge dieser ergibt[1]. Weitere Beispiele für derartige Anfragen finden sich z.B. in [ACDG03].

Die Integration von vagen oder unpräzisen Prädikaten in logikbasierte Anfragesprachen stellt ein aktuelles Problem der Forschung dar [CW08, Wei07, LSDJ06, RJ05, CRW05]. Klassische Ansätze bedienen sich zumeist der Fuzzy-Logik [SS04, CMPT00, BP95, Zad88]. Der in dieser Arbeit vorgestellte CQQL-Ansatz basiert hingegen auf Gesetzen der Quantenlogik [Sch08, BN36]. Diese werden verwendet, um verschiedene Anfrageparadigmen,

[1] Hierbei kann es sich z.B. um ein Ranking anhand der Ähnlichkeit der Objekte bzgl. der Nutzeranfrage handeln.

wie das im Information-Retrieval(IR) häufig verwendete Vektorraummodell mit der Booleschen Logik, zu kombinieren.

Die Einbindung unpräziser Prädikate in Anfragen führt zu einem weiteren Problem: der Gewichtung. Untersuchungen im Information-Retrieval belegen, dass eine vom Anwender beeinflusste Gewichtung einzelner Terme die Nutzerzufriedenheit mit dem Anfrageergebnis erhöht [SFW83]. Psychologische Experimente stützen diese Ergebnisse und stellen das subjektive Empfinden von Ähnlichkeit heraus [BG93, Sel59]. Folglich ermöglicht erst die Angabe von Termgewichten die Anpassung einer Anfrage an die subjektive Ähnlichkeitswahrnehmung, der Präferenz, des Nutzers.

Die Möglichkeit der Präferenzangabe unterscheidet sich in zwei alternative Klassen: die qualitativen Ansätze [Cho03, Kie02] und die quantitativen, zu denen der in der Arbeit vorgestellte Ansatz gehört. Qualitative Ansätze basieren auf der Angabe von Halbordnungen, in welchen der Nutzer seine Präferenzen bezüglich einer Datenbank formuliert. Hierbei muss angemerkt werden, dass dies voraussetzt, dass der Nutzer bereits zum Zeitpunkt der Anfrageformulierung die Präferenzen kennt[2]. Der Skyline-Operator [KRR02, BKS01] kann ebenfalls dieser Gruppe zugeordnet werden. In Abgrenzung dazu wird beim quantitativen Ansatz eine totale Ordnung der Ergebnisobjekte auf der Grundlage von numerischen Score-Werten erzeugt, wie dies im Information-Retrieval üblich ist. Unterstützer der qualitativen Ansätze betrachten deshalb quantitative Verfahren als Spezialfall der qualitativen Verfahren[3]. Bei der Anfrageformulierung im vorgestellten, quantitativen Ansatz müssen Gewichte numerisch angegeben werden, was gerade bei komplexen Anfragen eine Hürde für den Nutzer darstellt. Durch geeignete Verfahren des maschinellen Lernens kann die Angabe konkreter Gewichte in unserem Ansatz jedoch verborgen werden und stellt so keine zusätzliche Beeinträchtigung der Nutzerinteraktion dar. Beide Klassen von Ansätzen haben jeweils ihre Berechtigung in unterschiedlichen Einsatzszenarien. Folgendes Beispiel soll den Vorteil des quantitativen Ansatzes in einem speziellen Szenario verdeutlichen. Kamera $A$ sei in einer Bewertungskategorie wesentlich besser als Kamera $B$. Gleichzeitig sei Kamera $B$ nur geringfügig besser als $A$ in einer anderen Kategorie. Qualitative Ansätze ermöglichen keine Unterscheidung zwischen „wesentlich“ und „geringfügig besser“, so dass die Ergebnismenge weniger ausdifferenziert wird.

Zur Einbindung von Gewichten in Scoring-Verfahren existiert u.a. ein Vorschlag von Fagin und Wimmers [FW00], dessen arithmetische Berechnung jedoch nicht in eine Anfragelogik eingebettet ist. Der vorliegende Gewichtungsansatz kann als dessen Verallgemeinerung gesehen werden, ist allerdings mächtiger, da er komplett in die Logik integriert ist und den Gesetzen einer booleschen Algebra gehorcht. Dies ermöglicht die weitergehende Nutzung logikbasierter Optimierungstechniken innerhalb von DBS.

Wie bereits vorgestellt, stellt das Setzen von konkreten Gewichten ein wesentliches Problem während der Nutzerinteraktion dar. Folgendes Anwendungsszenario soll dies verdeutlichen.

---

[2] Dass diese Präferenzen in jedem Fall angeben werden können, kann bezweifelt werden. So ist es z.B. beim Image-Retrieval kaum denkbar, dass ein Laie konkret angeben kann, wie wichtig ihm einzelne Feature-Werte sind.

[3] Da jede Totalordnung eine Halbordnung ist. Dem kann eingewandt werden, dass andererseits jede Halbordnung mit der Dushnik-Miller-Dimension $d$ durch den Schnitt von $d$ Totalordnungen ausgedrückt werden kann.

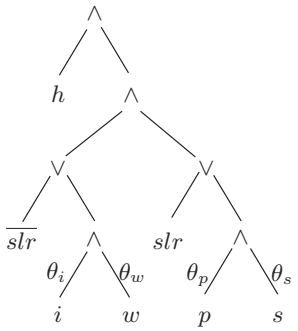



Abbildung 1: Gewichtete Anfrage

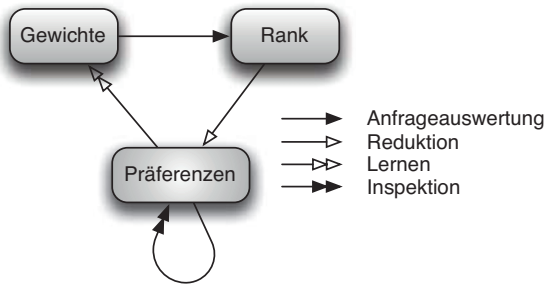



Abbildung 2: Zustandsdiagramm der Datenstrukturen

**Beispiel 1.1** *Ein Anwender sucht nach einer passenden Kamera. Diese soll gut bedienbar sein ($h$). Außerdem fordert er bei einer Spiegelreflexkamera (SLR) gute Bildqualität ($i$) und geringes Gewicht ($w$). Findet sich keine SLR, dann muss das Gerät preisgünstig ($p$) sein und eine geringe Auslöseverzögerung ($s$) bieten. Der relative Einfluss der Bedingungen $i, w, p$ und $s$ wird durch entsprechende Gewichtsvariablen ausgedrückt. Abb. 1 (links) zeigt den entsprechenden Bedingungsbaum.*

Des Weiteren wird angenommen, dass es sich bei $q^\Theta$ um eine gewichtete CQQL-Anfrage mit den Gewichtsvariablen $\Theta = \{\theta_i\}$ handelt. Die Anfragesprache CQQL wird in [Sch08] vorgestellt und in Abschnitt 2 kurz beschrieben. Sie erweitert den relationalen Bereichskalkül um die Behandlung von unscharfen Bedingungen und gewichteten Operanden von Konjunktionen und Disjunktionen. Eine Funktion $w$ ordnet in CQQL jeder Gewichtsvariable $\theta_i \in \Theta$ einen Wert aus dem Intervall $[0, 1]$ zu. Die initiale Gewichtungsfunktion $w^I$ setzt in diesem Fall alle Variablen auf den konstanten Wert 1.

Die Anfrage wird auf einer Menge von Anfrageobjekten $O$ bestehend aus $n$ Elementen ausgeführt. Die Evaluierung von $q^\Theta$ geschieht durch die Funktion $eval(q^\Theta, o, w)$, welche für jedes Objekt $o \in O$ einen Score aus $[0, 1]$ berechnet[4]. Die Ausführung von $eval(q^\Theta, o, w^I)$ über alle $o$ ergibt einen Rank (Permutation) $p(w^I) : \{1, \ldots, n\} \rightarrow O$, welche eine totale Ordnung der Objekte bezüglich ihrer Scores gegenüber der Anfrage ergibt. Des Weiteren wird angenommen, dass es eine Ziel-Gewichtungsfunktion $w^t$ für $q^\Theta$ gibt, welche einen Rank erzeugt, der genau dem Ähnlichkeitsempfinden des Nutzers entspricht.

Zum Zeitpunkt der initialen Anfrageformulierung ist $w^t$ in der Regel unbekannt. Dies führt dazu, dass die initiale Gewichtung $w^I$ häufig zu einem Rank führt, der dem subjektivem Empfinden des Nutzers widerspricht. Ziel ist es deshalb, sich dem Ziel-Rank, welcher durch $w^t$ erzeugt wird, sukzessive anzunähern (dies entspricht den im Information-Retrieval gebräuchlichen Relevance-Feedback [SB88, Roc71]). Hierbei ist festzuhalten, dass unterschiedliche Gewichte einen starken Einfluss auf das Ergebnis einer Anfrage haben können.

[4]Der Wert 1 gibt die maximale und der Wert 0 die minimale Erfüllung der Bedingung an.

Da das direkte Setzen der Gewichte, gerade bei komplexeren Anfragen, eine erhebliche Hürde für den Anwender darstellt, muss zur Verbesserung der Nutzerinteraktion eine Alternative bereitgestellt werden. Der vorliegende Ansatz greift hierbei auf das intuitiv verständliche Konzept der Präferenz zurück: Während der Interaktion mit dem System wird es dem Anwender mittels Präferenzen ermöglicht, die Reihenfolge der gerankten Ergebnisobjekte zu verändern. Die Angabe einer Präferenz geschieht dabei paarweise und bedeutet, dass ein konkretes Objekt $A$ vor einem Objekt $B$ im neuen Rank erscheinen soll. Anhand dieser Präferenzen lernt das vorgeschlagene System Gewichtswerte. Durch die wiederholte Erzeugung neuer Ranks anhand neu gelernter Gewichtswerte kann gezeigt werden, dass sich die Ergebnisobjekte immer mehr denen des Ziel-Ranks annähern.

Als Folgeproblem ergibt sich die Fragestellung nach der Präsentation der aktuellen Gewichte für den Nutzer. Wir schlagen hier die automatische Ableitung (Reduktion, siehe Abb. 2) von Präferenzen aus den ersten $k$ Objekten des erzeugten Rank vor. Diese müssen nach Rückumwandlung in Gewichte denselben Top-$k$-Rank erzeugen wie die ursprünglichen Gewichte. Diese abgeleiteten Präferenzen unterscheiden sich in der Regel von den vom Anwender bestätigten Präferenzen, da diese meist eine Änderung der ersten $k$ Objekte im neuen Rank bewirken: Objekte verschwinden aus dem Top-$k$-Rank und werden durch neue ersetzt. Bezüglich der Generierung des Top-$k$-Ranks sind beide Präferenzmengen jedoch äquivalent. Die abgeleiteten Präferenzen können nun in einer erneuten Nutzerinteraktion bei Bedarf modifiziert werden. Können alle abgeleiteten Präferenzen bestätigt werden, terminiert der Suchprozess. Voraussetzung einer Terminierung ist dabei die Konsistenz der spezifizierten Präferenzen und die Existenz einer Ziel-Gewichtungsfunktion $w^t$.

Hierbei ist es wichtig anzumerken, dass der Nutzer ausschließlich mit Präferenzen arbeitet und die zu Grunde liegenden Gewichtswerte nicht bemerkt.

### 1.1 Aufbau der Arbeit

Die Arbeit gliedert sich wie folgt. Kapitel 2 umreißt die Anfragesprache CQQL und grenzt diese gegenüber der Fuzzy-Logik ab. Das folgende Kapitel beschreibt die Integration von Gewichten in CQQL. Kapitel 4 erläutert das Prinzip der Präferenz und deren Bedeutung für die zu erlernenden Gewichte. Darauf folgt eine detaillierte Beschreibung der Umwandlung zwischen Präferenzen und Gewichten sowie deren Einbettung in den iterativen Relevance-Feedback-Prozess, welcher in Kapitel 6 diskutiert wird. Den Abschluss der Arbeit bilden Ergebnisse aus Experimenten und deren Bewertungen gefolgt von einem kurzen Ausblick auf kommende Arbeiten.

## 2 CQQL-Bedingungen und ihre Auswertung

CQQL steht für *commuting quantum query language* und basiert auf Ergebnissen der Quantenlogik [Zie05, BN36]. Die Anfragesprache erlaubt die Formulierung von logik-

basierten Anfragen, welche Boolesche und Score-basierte Bedingungen, die im IR üblich sind, vereinen. Score-basierte Bedingungen testen etwa, wie nah ein Attributwert einem Zielwert ist und geben jeweils einen Wert aus dem Intervall $[0, 1]$ zurück. Eine umfassende Darstellung der theoretischen Grundlagen von CQQL findet sich in [Sch08]. Syntaktisch lehnt sich CQQL an den relationalen Bereichskalkül an.

Beispiel 1.1 erfordert eine Anfrage mit Booleschen und Score-basierten Bedingungen auf der Relation

$$cameras(name, handling, SLR, image_quality, weight, price, shutter_lag)).$$

Die Bedingung *slr* ist eine klassische DB-Bedingung während die übrigen Score-basiert sind. Die Anfrage wird wie folgt formuliert:

$$\{name | \exists h, slr, i, w, p, s : cameras(name, h, slr, i, w, p, s) \wedge h \approx good \wedge \\ (\neg slr \vee (i \approx high \wedge_{\theta_i, \theta_w} w \approx low)) \wedge (slr \vee (p \approx low \wedge_{\theta_p, \theta_s} s \approx low)\}$$

Wie diese Anfrage beispielhaft zeigt, erweitert CQQL den relationalen Bereichskalkül um die Behandlung von Score-basierten Bedingungen, etwa $i \approx high$, und gewichteten Konjunktionen, etwa $\wedge_{\theta_p, \theta_s}$.

Aufgrund der Score-Werte kann die klassische Boolesche Logik in solch einem Szenario nicht verwendet werden. Auf den ersten Blick bietet sich der Rückgriff auf Fuzzy-Logik [SS04, CMPT00, BP95, Zad88] an. Fuzzy-Logik verfügt jedoch über Nachteile bei der Auswertung von Bedingungen, da sich die Ergebnisse dieser nicht immer mit der Erwartung seitens der Nutzer deckt [SZN08, KN07, Lee94]. Dies bezieht sich insbesondere auf das Verhalten der einzigen distributiven und idempotenten T-Norm (Konjunktion), der *min*-Funktion, deren Ergebnis immer genau einem Eingabeparameter entspricht, währende der andere ignoriert wird (Dominanzproblem). Weiterhin werden Gesetze der Booleschen Algebra, z.B. '$x \wedge \neg x =$ falsch' verletzt: $\min(0.5, 1 - 0.5) \neq 0$. Alternative T-Normen und T-Konormen ohne Dominanzproblem verletzen andererseits Idempotenz und Distributivität.

Die Semantik unseres Sprachvorschlags CQQL baut hingegen auf der Quantenlogik auf. Dies bedeutet, dass Konjunktion, Disjunktion und Negation folgend den Gesetzen der Quantenlogik ausgewertet werden. Konzeptionell wird dabei jeder Tupel als ein normierter Vektor in einem geeignet gewählten Hilbertraum modelliert, während eine Bedingung einem Vektorunterraum entspricht. Alle Vektorunterräume bilden zusammen mit der Halbordnung $\subseteq$ einen orthomodularen Verband. Dieser Verband liefert uns die Konjunktion, Disjunktion und die Negation. Basieren alle Vektorunterräume auf einer gemeinsamen Orthonormalbasis (die Projektoren der Vektorunterräume *kommutieren* also paarweise: $p_i p_j = p_j p_i$), dann erfüllt der entstehende Unterverband alle Gesetze der Booleschen Algebra. Bei der Auswertung einer Bedingung gegenüber einem Tupel wird das quadrierte innere Produkt zwischen dem Vektor und dem Vektorunterraum gemessen.

Sowohl Score-basierte als auch klassische Datenbankbedingungen lassen sich in diesem Modell nachbilden. Es kann sogar gezeigt werden, dass CQQL eine Verallgemeinerung des relationalen Bereichskalküls darstellt sowie den Gesetzen einer Booleschen Algebra genügt [Sch08].

Bevor eine CQQL-Bedingung algorithmisch ausgewertet werden kann, muss diese in eine spezielle syntaktische Form *normalisiert* werden. Der Normalisierungsalgorithmus basiert auf Booleschen Transformationsregeln und wird detailliert in [Sch08] diskutiert. Normalisierungsschritte sind Überführung in die Prenex-Normalform sowie die Anwendung des Distributivgesetzes, um gemeinsame Bedingungen in den Operanden einer Konjunktion oder Disjunktion zu isolieren: $(\varphi \wedge \varphi_1) \vee (\varphi \wedge \varphi_2) \Rightarrow \varphi \wedge (\varphi_1 \vee \varphi_2)$.

Aus den Gesetzen der Quantenlogik und Quantenmechanik lassen sich für jede normalisierte Konjunktion und Disjunktion eine Auswertung mit einfachen arithmetischen Operationen ableiten. Seien $eval(\varphi, o) \in [0; 1]$ die Auswertung der Bedingung $\varphi$ von Objekt $o$ und $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \neg\varphi$ *normalisierte*[5] Bedingungen, dann wird die CQQL-Evaluierung durch rekursive Anwendung der folgenden Vorschriften erreicht:

$$
\begin{array}{ll}
eval(\varphi_1 \wedge \varphi_2, o) = & eval(\varphi_1, o) * eval(\varphi_2, o) \\
eval(\varphi_1 \vee \varphi_2, o) = & eval(\varphi_1, o) + eval(\varphi_2, o) - eval(\varphi_1, o) * eval(\varphi_2, o) \\
 & \text{wenn } \varphi_1 \text{ und } \varphi_2 \text{ nicht exklusiv} \\
eval(\varphi_1 \vee \varphi_2, o) = & eval(\varphi_1, o) + eval(\varphi_2, o) \\
 & \text{wenn } \varphi_1 \text{ and } \varphi_2 \text{ exklusiv} \\
eval(\neg\varphi, o) = & 1 - eval(\varphi, o).
\end{array}
$$

Eine Disjunktion ist *exklusiv*, wenn sie die Form $(\varphi \wedge \ldots) \vee (\neg\varphi \wedge \ldots)$ hat. Jede Evaluierung einer komplexen Anfragebedingung kann durch eine Summe von Produkten (analog zur DNF) atomarer Bedingungsauswertungen auf Objektattributen ausgedrückt werden.

## 3 Gewichtung von Anfragebedingungen

Die Integration von Gewichten in CQQL gestaltet sich einfach. Die Operanden von Konjunktion und Disjunktionen einer Bedingung können mit Gewichtsvariablen $\theta_i$ ausgestattet werden. Vor der Auswertung der Bedingung werden die Gewichtsvariablen mittels der Gewichtungsfunktion $w$ auf Werte aus dem Intervall $[0, 1]$ abgebildet. Die Werte von Gewichtsvariable $\theta_i$ kontrollieren den Einfluss der Bedingung $\varphi_i$ in der gewichteten Konjunktion $\varphi_1 \wedge_{\theta_1,\theta_2} \varphi_2$ und der Disjunktion $\varphi_1 \wedge_{\theta_1,\theta_2} \varphi_2$.

Die Kernidee des vorgestellten Ansatzes besteht in der Anwendung einer syntaktischen Ersetzungsregel. Diese Transformation wandelt eine gewichtete Konjunktion oder Disjunktion in einen logischen Ausdruck ohne Gewichte um, wobei Gewichtswerte in konstante Score-Werte überführt werden:

$\varphi_1 \wedge_{\theta_1,\theta_2} \varphi_2 := (\varphi_1 \vee \neg\theta_1) \wedge (\varphi_2 \vee \neg\theta_2)$ und $\varphi_1 \vee_{\theta_1,\theta_2} \varphi_2 := (\varphi_1 \wedge \theta_1) \vee (\varphi_2 \wedge \theta_2)$

Abb. 3 illustriert den Transformationsschritt für Beispiel 1.1. Als Anfrage $q^{\Theta}$ erhalten wir

$$
\begin{array}{lll}
\{name & | & \exists h, slr, i, w, p, s : cameras(name, h, slr, i, w, p, s) \wedge h \approx good \wedge \\
 & & (\neg slr \vee ((i \approx high \vee \neg\theta_i) \wedge (w \approx low \vee \neg\theta_w))) \wedge \\
 & & (slr \vee ((p \approx low \vee \neg\theta_p) \wedge (s \approx low \vee \neg\theta_s)))\}.
\end{array}
$$

[5] Aufgrund der Forderung nach Normalisierung kann die Bedingung $x \wedge \neg x = \text{falsch}$ in CQQL gar nicht *direkt* ausgewertet werden. Tatsächlich wird sie innerhalb der Normalisierung aufgelöst.

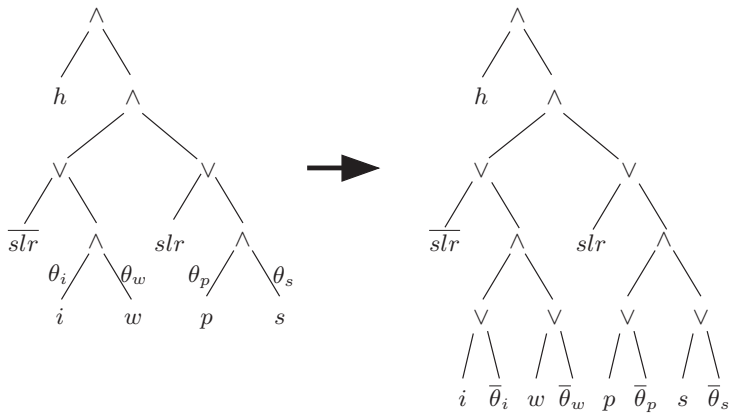



Abbildung 3: Transformation der Beispielanfrage in ihr logisches Äquivalent

Die arithmetische Auswertung der Anfrage nach der Normalisierung und der Anwendung der Regeln aus dem vorherigen Abschnitt ergibt

$$\begin{aligned} eval(q^{\Theta}, o) &= h(slr * iw + \overline{slr} * ps) \text{ mit} \\ iw &= (i + \overline{\theta}_i - i\overline{\theta}_i)(w + \overline{\theta}_w - w\overline{\theta}_w) \\ ps &= (p + \overline{\theta}_p - p\overline{\theta}_p)(s + \overline{\theta}_s - s\overline{\theta}_s), \end{aligned}$$

wobei $\overline{\theta}_x = 1 - \theta_x$ gilt, und die Werte $h, i, w, p, s$ die Score-Werte von Objekt $o$ bzgl. der atomaren Bedingungen sind.

Tabelle 1 gibt beispielhaft die Score-Werte für die atomaren Beispielbedingungen für vier Objekte an. In Abhängigkeit von den Gewichtswerten werden, wie in Tabelle 2 dargestellt, unterschiedliche Gesamt-Score-Werte für die vier Objekte entsprechend der Anfrageauswertung berechnet. Dabei zeigt sich, dass Objekt $o_2$ immer schlechter als $o_1$ bewertet wird. Dies ist nicht verwunderlich, da alle atomaren Score-Werte von $o_2$ nicht besser als die von $o_1$ sind. Durch keine Wertebelegung der Gewichte lässt sich daher erreichen, dass $o_2$ besser als $o_1$ bewertet wird.

Objekt $o_3$ wird bei den ersten Gewichtswerten schlechter als Objekt $o_4$ ausgewertet. Dies lässt sich umkehren, wenn die Bedingung $p$ durch ein Gewicht ausgeschaltet wird.

Insgesamt werden die Spiegelreflexkameras $o_1$ und $o_2$ schlechter als die einfachen Kameras $o_3$ und $o_4$ ausgewertet. Möchte man dieses Ergebnis verändern, so ist es nur nötig, die

| $o_i$ | $slr$ | $h$ | $i$ | $p$ | $s$ | $w$ |
|---|---|---|---|---|---|---|
| $o_1$ | 1 | 0,8 | 0,7 | 0,6 | 0,8 | 0,3 |
| $o_2$ | 1 | 0,4 | 0,5 | 0,5 | 0,6 | 0,2 |
| $o_3$ | 0 | 0,8 | 0,3 | 0,4 | 0,7 | 0,5 |
| $o_4$ | 0 | 0,9 | 0,6 | 0,8 | 0,6 | 0,6 |

Tabelle 1: Beispielhafte Score-Werte für atomare Bedingungen

| $\theta_i$ | $\theta_w$ | $\theta_p$ | $\theta_s$ | $o_1$ | $o_2$ | $o_3$ | $o_4$ |
|---|---|---|---|---|---|---|---|
| 1,0 | 1,0 | 1,0 | 1,0 | 0,168 | 0,04 | 0,224 | 0,432 |
| 1,0 | 1,0 | 0,0 | 1,0 | 0,168 | 0,04 | 0,56 | 0,54 |
| 0,5 | 0,5 | 1,0 | 1,0 | 0,442 | 0,18 | 0,224 | 0,432 |

Tabelle 2: Gesamt-Score-Werte in Abhängigkeit von Gewichten

| Name | Formel |
|---|---|
| Gew. Summe (gs) | $\theta_1 * x + \theta_2 * y$ |
| Gew. Disj. (gd) | $(x \wedge \theta_1) \vee (y \wedge \theta_2)$ |
| Gew. Konj. (gk) | $\left(x \vee \overline{\theta}_1\right) \wedge \left(y \vee \overline{\theta}_2\right)$ |

| Name | Arithm. Auswertung |
|---|---|
| Gew. Disj. (gd) | $\theta_1 * x + \theta_2 * y -$ $\theta_1 * x * \theta_2 * y$ |
| Gew. Konj. (gk) | $(\theta_1 + x - \theta_1 * x) *$ $(\theta_2 + y - \theta_2 * y)$ |

| x | y | $\theta_1$ | $\theta_2$ | gs | gd | gk |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 2 | 1 | 1 |

Abbildung 4: Vergleich verschiedener Gewichtungsformeln

Gewichtswerte für die Spiegelreflexbedingungen $i$ und $w$ gemeinsam herunterzusetzen. Dies bewirkt ein Anstieg der SLR-Kameras, da die Bedingungen $i$ und $w$ in der Konjunktion ihren Einfluss auf die Gesamt-Score-Werte verlieren.

Die Betrachtung der folgenden Extremfälle erleichtert das Verständnis der Ersetzungsregel. Ein *Nullgewicht ($\theta_i = 0$)* führt dazu, dass die mit Null gewichtete Bedingung keinen Einfluss auf das Gesamtergebnis hat. *Gleiche Gewichtswerte ($\theta_1 = \theta_2 = 1$)* hingegen führen zu einem Gesamtergebnis, welches sich so verhält, als ob keine Gewichte vorhanden wären.

Fagin und Wimmers fordern in [FW00] die Linearität der Gewichtung. Aufgrund der Linearität der CQQL-Auswertungen (Summe von Produkten) bewegt sich die Evaluierung bezogen auf eine Gewichtsvariable zwischen diesen beiden Fällen in CQQL ebenfalls linear. Abb. 4 stellt die gewichtete Konjunktion und Disjunktion der einfachen, gewichteten Summe, als Beispiel einer einfachen Heuristik, gegenüber und demonstriert das Verhalten der Gewichtung auf Booleschen Werten[6]. Damit soll auch gezeigt werden, dass unsere Gewichtung mittels einer syntaktischen Ersetzungsregel nicht auf die Sprache CQQL beschränkt ist, sondern etwa auch innerhalb der Booleschen Logik funktioniert.

Bereits existierende Gewichtungsansätze, wie z.B. von Fagin und Wimmers [FW00], führen arithmetische Auswertungen oberhalb einer logischen Bedingung aus. Bezüglich der Optimierung innerhalb eines DBS ist dieses Vorgehen nicht zu empfehlen, da so eine logische Optimierung erschwert bzw. unmöglich wird. Beispielsweise kann die Verletzung der Assoziativität und Distributivität gezeigt werden [SS03]. Fagin und Wimmers schränken die Freiheit der Gewichtung zudem gegenüber unserem Verfahren ein, da sie voraussetzen, dass für alle Gewichte $\theta_i \in [0, 1]$ einer gewichteten Konjunktion oder Disjunktion immer $\sum_i \theta_i = 1$ gilt.

Die innovative Idee unsers Gewichtungsansatzes besteht darin, dass sich eine Gewichtung *ausschließlich* mit den Mitteln einer Logik realisieren lässt.

[6] 0 für 'falsch' und 1 für 'wahr'

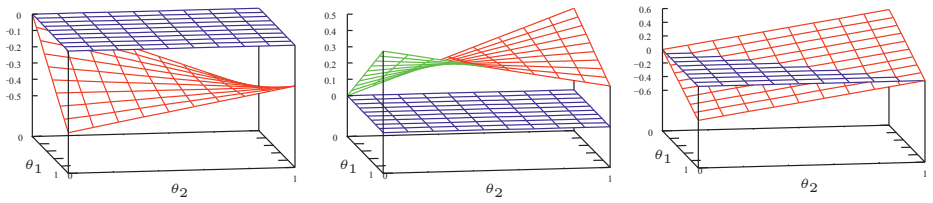



Abbildung 5: Fälle der Nützlichkeit der Präferenz $o_1 \geq o_2$ bezogen auf die 0-Hyperebene

# 4 Präferenzen

Die Angabe von Präferenzen, also ob ein Objekt eine Anfrage besser als ein anderes erfüllt, ist in vielen Fällen einfacher für den Nutzer als das explizite Setzen von Gewichten, um das subjektive Empfinden von Ähnlichkeit bezüglich einer Anfrage auszudrücken. Da binäre Präferenzen zwischen zwei Objekten intuitiv verständlich sind, greift der vorgestellte Ansatz auf diese im Rahmen der Benutzerschnittstelle zurück. Im Gegensatz zu anderen Verfahren [Cho07, Kie02] verbergen Präferenzen in dieser Arbeit konkrete Gewichte, welche intern Ranks erzeugen (siehe Abb. 2). Hierdurch wird eine feinere Ausdifferenzierung der Ergebnisobjekte in Form einer totalen Ordnung erzielt (siehe Kap. 1).

Präferenzen bezüglich einer gewichteten Anfrage $q^{\Theta}$ werden als $o_1 \geq o_2$ angegeben, wobei $o_1, o_2 \in O$ gilt. Eine Gewichtungsfunktion $w$ erfüllt eine Präferenz, wenn Folgendes gilt: $eval(q^{\Theta}, o_1, w) - eval(q^{\Theta}, o_2, w) \geq 0$. Eine Menge von definierten Präferenzen $P$ für $q^{\Theta}$ heißt *konsistent*, wenn ihr transitiver und reflexiver Abschluss eine Halbordnung darstellt. D.h., dass Zyklen wie $o_1 \geq o_2$ und $o_2 \geq o_1$ mit $o_1 \neq o_2$ nicht erlaubt sind. Im Folgenden wird angenommen, das $P$ konsistent ist und bereits bezüglich Reflexivität und Transitivität reduziert ist.

Abb. 5 illustriert das Konzept der Präferenz bezogen auf den (Hyper-)Einheitswürfel, der durch zwei Gewichte, also die beiden Achsen $\theta_1$ und $\theta_2$, aufgespannt wird. Die Präferenz bezüglich einer Anfrage kann dabei als Hyperebene über dem Einheitswürfel (im vorliegenden Fall über der Einheitesebene) gesehen werden. Je nach Gewichtungswerten und maximalem oder minimalem Funktionswert $\max$ bzw. $\min$ fällt jede Präferenz dabei in eine der folgenden Kategorien:

*Nicht erfüllbar*: Eine Präferenz ist nicht erfüllbar, wenn $max < 0$ gilt. D.h. die Präferenz kann durch keine Gewichtungwerte von $q^{\Theta}$ erfüllt werden (siehe Abb. 5 (links)).

*Nutzlos*: Eine Präferenz ist zum Lernen von Gewichten nutzlos, wenn $min \geq 0$ gilt. In diesem Fall stellt die Präferenz keine Einschränkung für die Gewichtswerte von $q^{\Theta}$ dar (siehe Abb. 5 (Mitte)).

*Nützlich*: In den übrigen Fällen ist die Präferenz nützlich (siehe Abb. 5 (rechts)).

Ein Beispiel für eine nicht erfüllbare Präferenz ist $o_2 \geq o_1$, wobei $o_1$ und $o_2$ die Objekte aus Tabelle 1 darstellen. Wenn man diese Präferenz umkehrt, also $o_1 \geq o_2$ fordert, erhält man eine nutzlose Präferenz.

Im folgenden Abschnitt wird gezeigt, wie nicht erfüllbare oder nutzlose Präferenzen erkannt und bereits während der Nutzerinteraktion unterbunden können. Des Weiteren wird deshalb angenommen, dass die Menge aller Präferenzen $P$ nur nützliche Elemente enthält.

### 4.1 $\theta$-Unabhängigkeit

Um Präferenzen anhand maximaler und minimaler Differenzen effektiv klassifizieren zu können, ist es notwendig zu wissen, ob eine Gewichtsvariable von anderen abhängt.

**Definition 4.1** *Folgende Funktion sei für eine gewichtete CQQL-Anfrage $q^{\Theta}$ definiert:*

$$\mathit{diff}(o_1, o_2, \theta, v, w) = eval(q^{\Theta}, o_1, w_{\theta,v}) - eval(q^{\Theta}, o_2, w_{\theta,v}).$$

*Dabei sind $o_1, o_2$ Datenbankobjekte, $\theta \in \Theta$ eine Gewichtsvariable, $v \in [0,1]$ ein konkreter Wert für $\theta$, $w_{\theta,v}$ eine Gewichtungsfunktion*[7] *und $eval()$ eine CQQL-Auswertungsfunktion. Eine Anfrage $q^{\Theta}$ heißt für ein $\theta$* $\theta$-abhängig *von anderen Gewichtsvariablen, wenn zwei beliebige Objekte $o_1, o_2$, zwei Gewichtswerte $v_1, v_2 \in [0,1]$ und zwei Gewichtungsfunktionen $w_1, w_2$ existieren, so dass*

$$\begin{aligned} &\mathit{diff}(o_1, o_2, \theta, v_1, w_1) \geq \mathit{diff}(o_1, o_2, \theta, v_2, w_1) \qquad \wedge \\ &\mathit{diff}(o_1, o_2, \theta, v_1, w_2) \leq \mathit{diff}(o_1, o_2, \theta, v_2, w_2) \end{aligned}$$

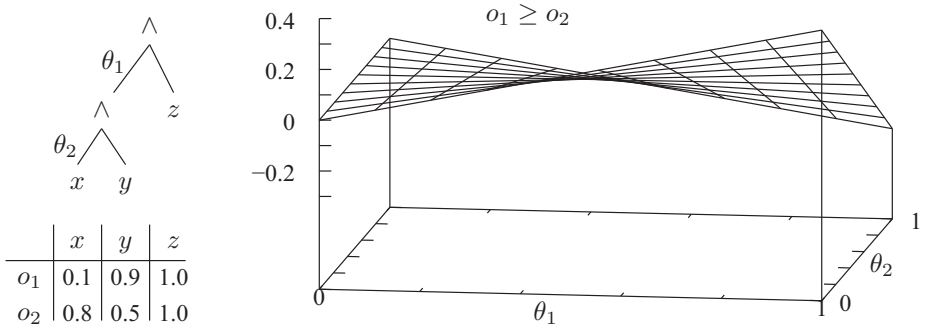
*gilt. Eine Anfrage $q^{\Theta}$ heißt* $\theta$-unabhängig*, wenn sie für kein $\theta$* $\theta$-abhängig *ist.*

$\theta$-Unabhängigkeit bedeutet also, dass die Monotonie von *diff* bezüglich unterschiedlicher Werte für das Gewicht $\theta$ unabhängig von anderen Gewichtsvariablen ist. Die tatsächliche Überprüfung der $\theta$-Unabhängigkeit einer Anfrage nach Def. 4.1 ist nicht praktikabel. Deshalb bietet die folgende Beobachtung die Möglichkeit eines effizienten Tests.

> Eine CQQL-Anfrage $q^{\Theta}$ ist $\theta$-unabhängig, wenn für jede gewichtete Konjunktion $w^{\wedge}_{\theta_1,\theta_2}(\varphi_1, \varphi_2)$ oder Disjunktion $w^{\vee}_{\theta_1,\theta_2}(\varphi_1, \varphi_2)$ keine Unterbedingung $\varphi_1$ oder $\varphi_2$ weitere gewichtete Konjunktionen oder Disjunktionen enthält.

Demzufolge garantiert der Verzicht auf verschachtelte, gewichtete Bedingungen $\theta$-Unabhängigkeit, da ja dann Gewichte in voneinander unabhängigen Teilbedingungen wirken. Abschnitt 4.2 diskutiert Besonderheiten bei verschachtelten, gewichteten Bedingungen. Abb. 6 illustriert das Verhalten einer $\theta$-abhängigen Anfrage mit geschachtelter Gewichtung. Die Anfragebedingung besteht aus zwei ineinander geschachtelten und gewichteten Konjunktionen. Die Score-Werte der Bedingungen $x, y, z$ für die Objekte $o_1$ und $o_2$ sind in der Tabelle angegeben. Die Präferenz $o_1 \geq o_2$ erzeugt als Auswertungsdifferenz die in der Abbildung dargestellte Hyperfläche. Dort nimmt die Different bei steigendem $\theta_1$ für $\theta_2 = 0$ zu und für $\theta_2 = 1$ ab.

[7] $w_{\theta,v} = w \setminus \{(\theta, w(\theta))\} \cup \{(\theta, v)\}$, d.h. alle Gewichte sind entsprechend der Gewichtungsfunktion $w$ gesetzt wobei $\theta$ auf $v$ gesetzt ist.




| | $x$ | $y$ | $z$ |
|---|---|---|---|
| $o_1$ | 0.1 | 0.9 | 1.0 |
| $o_2$ | 0.8 | 0.5 | 1.0 |

Abbildung 6: $\theta$-Abhängigkeit aufgrund geschachtelter Gewichtung für $o_1 \geq o_2$

Aufgrund der Monotonie ist es nun einfach, für eine Präferenz und eine $\theta$-unabhängige Anfrage das Maximum und Minimum von *diff* zu ermitteln. Das Maximum kann sich nur in einer Ecke $\{0,1\}^n$ des $n$-dimensionalen Einheitshyperwürfels, der durch $n$ Gewichte aufgespannt wird, befinden. Dieser gegenüber befindet sich das Minimum.

Der Algorithmus zum Finden des Maximums und zur Kategorisierung einer ungeschachtelten Präferenz findet sich in Abb. 7. Für jede Gewichtsvariable $\theta$ und ein beliebiges $w$ überprüft man lediglich *diff*$(o_1, o_2, \theta, 0, w) \geq$ *diff*$(o_1, o_2, \theta, 1, w)$. Ergibt der Test *wahr*, dann ist das $\theta$-spezifische Maximum $w_{max} = 0$ und Minimum $w_{min} = 1$ bzw. umgekehrt. Nachdem die entsprechenden Ecken gefunden worden sind, kann die maximale bzw. minimale Differenz $max$ und $min$ berechnet werden, um die Präferenz in die Kategorien *nicht erfüllbar, Nutzlos* oder *Nützlich* einzuordnen.

```
for each θ ∈ Θ do
  if diff(o1, o2, θ, 0, w) ≥ diff(o1, o2, θ, 1, w) then
    w_max(θ) = 0
    w_min(θ) = 1
  else
    w_max(θ) = 1
    w_min(θ) = 0
max = diff(o1, o2, θ, w_max(θ), w_max)
min = diff(o1, o2, θ, w_min(θ), w_min)
if max < 0 or min ≥ 0 then
  reject preference o1 ≥ o2
display (w_max)
```

Abbildung 7: Algorithmus zur Kategorisierung einer Präferenz $o_1 \geq o_2$

Im Falle einer nützlichen Präferenz enthält $w_{max}$ wichtige Informationen für den Anwender. Basierend auf einer Präferenz versucht der Lernalgorithmus die Gewichte in Richtung des Maximums zu verschieben. Demnach charakterisiert die Lage des Maximums

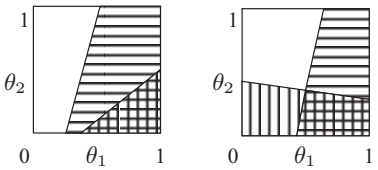



Abbildung 8: Implikation (links) and Überlappung (rechts) von Präferenzen

die Präferenz. Mehrere Präferenzen sind daher ausdrucksmächtiger, wenn deren Maxima in unterschiedlichen Ecken liegen. Die Präferenz $o_3 \geq o_4$ der Objekte aus Tabelle 1 bzgl. unserer Beispielanfrage führt zu einer Bevorzugung der Bedingung $s$ auf Kosten der Bedingung $p$. Die Gewichte streben also zu dem Extrem $\theta_s = 1$ und $\theta_p = 0$.

Jede nützliche Präferenz definiert durch die Forderung *diff* $> 0$ außerdem eine Region des Gewichtshyperwürfels, die nur die Präferenz erfüllende Gewichtswerte enthält. Wird mehr als eine Präferenz angegeben, dann entspricht die Region gültiger Gewichtswerte dem Schnitt der durch die Präferenzen definierten Bereiche. Abb. 8 verdeutlicht dies. Eine Implikation bedeutet hier, dass eine Präferenz in einer anderen (räumlich) enthalten ist. D.h. die Entfernung der enthaltenen Präferenz ändert also die Beschränkung der Gewichtswerte nicht. Hauptziel ist es, die minimale Anzahl von Präferenzen zu ermitteln, die einen kleinstmöglichen aber nicht leeren Schnitt aufweisen, um Gewichtswerte möglichst eindeutig zu definieren. Eine geringe Anzahl an Präferenzen bedeutet vor allem weniger Last für den Nutzer, was detailliert in Kap. 6 begründet wird, und eine schnellere Ausführung des Lernalgorithmus bewirkt (siehe Kap. 7).

### 4.2 Verschachtelte Gewichtung

Aufgrund der $\theta$-Abhängigkeit in geschachtelten, gewichteten Anfragen gestaltet sich das Finden der Maxima und Minima schwieriger als in ungeschachtelten Anfragen. In diesem Fall ist es nicht mehr möglich, die Gewichte isoliert zu testen. Für die Lösung dieses nichtlinearen Optimierungsproblems bieten sich u.a. Hill-Climbing-Algorithmen, wie der im Folgenden verwendete Algorithmus [NM65], an. Aufgrund der Eigenschaft von CQQL-Evaluierungen, dass diese als Summe von Produkten von atomaren Bedingungen und Gewichten eines Objekts (Exponenten sind jeweils 1) ausgedrückt werden können, läuft man jedoch nicht Gefahr in lokalen Maxima oder Minima stecken zu bleiben.

Abb. 9 zeigt anhand eines Beispiels, dass in bestimmten Situationen eine geschachtelt gewichtete Anfrage durch Einführung neuer Gewichte in eine ungeschachtelte Anfrage umgewandelt werden kann. Die Transformationen ergeben sich durch die Ersetzungsregel für die gewichtete Konjunktion sowie durch Gesetze der Booleschen Algebra. Diese Umwandlung ist immer dann möglich, wenn eine gewichtete Konjunktion auf einer gewichteten Konjunktion folgt. Dies gilt analog für zwei aufeinander folgende Disjunktionen.

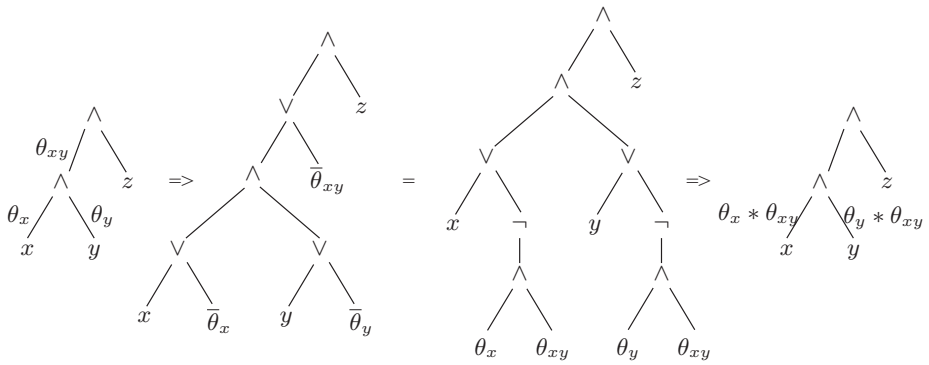



Abbildung 9: Transformation einer geschachtelten Anfrage in eine ungeschachtelte

# 5 Abbildungen zwischen Präferenzen und Gewichten

Die vorliegende Arbeit nutzt Präferenzen zur Nutzerkommunikation und Gewichte zur internen Repräsentation dieser. Mittels dieser Gewichte wird eine totale Ordnung von Ergebnisobjekten (Rank) erzeugt, welche dem Ähnlichkeitsempfinden des Nutzers bzgl. seiner Anfrage entsprechen soll. Im Folgenden wird die Abbildung von Präferenzen in Gewichte und von einem Rank in eine Menge von Präferenzen diskutiert.

## 5.1 Lernen von Gewichten aus Präferenzen

Die Eingabe des verwendeten Lernalgorithmus *prefsToWeight* bildet eine nutzerdefinierte gewichtete Anfrage $q^{\Theta}$ sowie eine konsistente Menge von Präferenzen $P$, die in Gewichte umgewandelt werden sollen (siehe Kap. 4), sowie eine Menge von Anfrageobjekten $O$. Außerdem wird angenommen, dass der Schnitt gültiger Präferenzregionen nicht leer ist.

Jede Präferenz bedeutet eine Einschränkung der Gewichtswerte $w$ durch $eval(q^{\Theta}, o_1, w) - eval(q^{\Theta}, o_2, w) \geq 0$. Mehrere Präferenzen werden wie folgt zusammengefasst:

$$\min_{(o_i \geq o_j) \in P} \left( eval(q^{\Theta}, o_i, w) - eval(q^{\Theta}, o_j, w) \right).$$

Damit die einzelnen Präferenzen gelten, wird gefordert, dass das Ergebnis der Zusammenfassung nicht kleiner als Null werden darf.

Das Lernen der Gewichte kann als ein Optimierungsproblem formuliert werden: das Finden der Gewichtswerte $w$, die obige Zielfunktion maximieren.

Betrachtet man die zu maximierende Funktion, so fällt auf, dass es sich um ein nichtlineares Optimierungsproblem handelt, da eine CQQL-Auswertung als Summe von Produkten atomarer Bedingungen und Gewichten (analog zur ausgezeichneten disjunktiven Normalform) ausgedrückt werden kann. Nichtlinearität führt zu einem schwer berechenbaren Problem. Abschwächend kann jedoch gesagt werden, dass nicht unbedingt das Maximum be-

rechnet werden muss, solange alle Präferenzen respektiert werden. Insofern sind gelernte Gewichte, welche Präferenzen genügen, nicht eindeutig festgelegt.

Nelder und Mead [NM65] schlagen für solche Probleme einen Hill-Climbing-Algorithmus, den *Downhill-Simplex-Algorithmus* vor, welcher das Maximum der Zielfunktion annähert. Um die Güte der Approximation zu verbessern, nutzen wir verschiedene, zufällig gewählte Startpunkte und verwenden das maximale Resultat der Durchläufe. Experimentelle Ergebnisse dieses Ansatzes werden in Kapitel 7 präsentiert.

## 5.2 Reduktion: Ableitungen von Präferenzen aus einem Rank

Die aus der Nutzerinteraktion gewonnene Gewichtungsfunktion $w$ erzeugt einen neuen Rank, welcher dem Anwender erneut präsentiert werden soll. Anstelle der Anzeige sämtlicher Ergebnisobjekte oder der Top-$k$-Objekte kann der Nutzer durch die Präsentation der charakteristischen Präferenzen $P'$, welche den Top-$k$-Rank erzeugen, unterstützt werden. Hierdurch wird das Verständnis des Nutzers für die aktuellen Gewichtswerte der Anfrage verbessert. Die abgeleiteten Präferenzen $P'$ unterscheiden sich in der Regel von den vom Anwender bestätigten Präferenzen, da letztere meist eine Änderung der ersten $k$ Objekte im neuen Rank bewirken, z.B. wenn ein Objekt durch den Nutzer als irrelevant bewertet wird. Diese aus dem Top-$k$-Rank entfernten Objekte werden durch neue ersetzt, welche die Grundlage für die Ableitung bilden. Bezüglich der Generierung des Top-$k$-Ranks sind beide Präferenzmengen jedoch äquivalent. Die abgeleiteten Präferenzen können nun in einer erneuten Nutzerinteraktion bei Bedarf modifiziert werden.

Folgende Anforderungen werden dabei für die abzuleitetende Präferenzmenge $P'$, für Gewichtungsfunktion $w$ und $k$ Ergebnisobjekte aufgestellt:

1. Die gegebene Gewichtungsfunktion $w$ sowie $P'$ erzeugen zwei Ranks $p(w)$ und $p(\textit{prefsToWeight}(P'))$, die $k$-äquivalent sind: $p(w) =_k p(\textit{prefsToWeight}(P'))$[8].

2. $P'$ ist minimal. Es kann also keine Präferenz entfernt werden kann, ohne dass die $k$-Äquivalenz verletzt wird:
   $\neg\exists(o_i, o_j) \in P' : p(\textit{prefsToWeight}(P' \setminus \{(o_i, o_j)\})) =_k p(w)$.

Zur Ableitung der Präferenzen aus einem gegebenen Rank, der durch $w$ erzeugt wurde, findet der folgende Algorithmus[9] Einsatz:

1. Anhand der ersten $k$ Objekte des Ranks werden paarweise Präferenzen $o_i \geq o_{i+1}$ für $i = 1, .., k-1$ abgeleitet. Die entstandene Präferenzmenge $P'$ ist damit bereits um Transitivität und Reflexivität reduziert. Nicht erfüllbare Präferenzen und leere Schnitte von Präferenzregionen treten nicht auf.

2. Nutzlose Präferenzen werden entfernt (siehe Kap. 4).

[8] D.h., ihre Top-$k$ Elemente sind gleich und treten in der gleichen Reihenfolge auf.
[9] Der Aufwand des Algorithmus ist quadratisch bezogen auf $k$ genutzte Objekte.

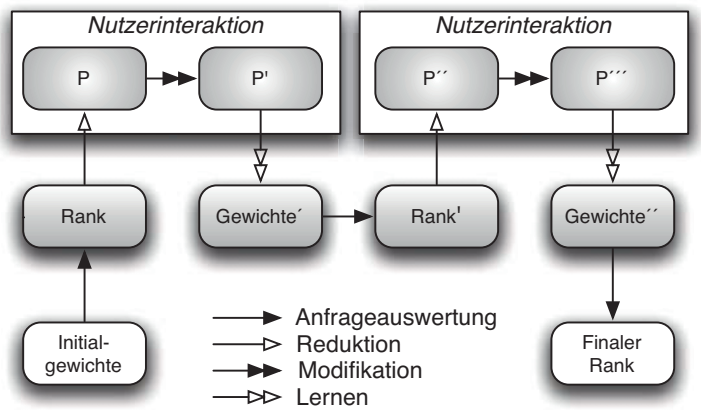



Abbildung 10: Verfeinerung von Gewichten durch Nutzerinteraktion mittels Präferenzen $P$

3. Erzeugung einer minimalen, $k$-äquivalenten Präferenzmenge $P'$:

   (a) Für jede Präferenz $p$ aus $P'$

      i. Teste, ob das Entfernen von $p$ aus $P'$ die $k$-Äquivalenz verletzt.

      ii. Bei Verletzung wird diese Präferenz wieder eingefügt.

   (b) Wiederhole Schritt (a), so lange, bis keine Präferenzen mehr ohne Verletzung der $k$-Äquivalenz entfernt werden können.

## 6 Konzeptioneller Relevance-Feedback-Prozess

Abb. 10 stellt den iterativen Relevance-Feedback-Prozess dar, welcher die vorgestellten Konzepte in sich vereint. Ziel diese Prozesses ist es, Gewichte durch nutzerseitige Interaktion mittels Präferenzen zu modifizieren und zu erlernen, um das subjekte Ähnlichkeitsempfinden bzgl. einer Nutzeranfrage abzubilden.

Während des Relevance-Feedbacks werden dem Nutzer Präferenzen präsentiert, die er entsprechend seines Empfindens verändern kann. Dabei kann aus den folgenden Alternativen gewählt werden:

*Bestätigung*: Eine Präferenz entspricht der Intention des Nutzers und muss nicht modifiziert werden.

*Umkehrung*: Eine Präferenz kann umgekehrt werden, wenn sie der beabsichtigten Bedeutung der Anfrage widerspricht.

*Entfernung*: Kann zwischen zwei Objekten keins bevorzugt werden, so kann die Präferenz entfernt werden.

*Erstellung*: Eine weitere Präferenz kann hinzugefügt werden, wenn diese die Intention des Nutzers besser ausdrückt.

Nach der Modifikation der Präferenzen durch den Nutzer werden diese auf Widersprüche und Nützlichkeit, wie in Kap. 4 und 5 beschrieben, hin untersucht. Um den Nutzer bei der Erstellung neuer Präferenzen zu unterstützen, ist es denkbar, dass ein Algorithmus Präferenzen vorschlägt. Gute Kandidaten sind hierbei solche, welche den Schnitt der Präferenzregionen halbieren oder aber die Gewichte in eine spezifische Richtung drängen.

Die Veränderung der Präferenzen durch den Nutzer führt i. d. R. zum jeweils neuen Rank $Rank'$ (siehe Abb. 10). Dies hat zur Folge, dass alte Top-$k$-Objekte verschwinden und dafür neue Objekte erscheinen. Diese neuen Top-$k$-Objekte dienen dann zur Ableitung neuer Präferenzen $P'$, welche vom Benutzer modifiziert werden können. Diese Schritte bewirken insgesamt ein Entwickeln der Präferenzen, d.h. $P$, $P'$ und $P''$ usw. unterscheiden sich. Greift der Nutzer nicht (weiter) ein, da er entweder zufrieden mit den abgeleiteten Präferenzen ist oder die Suche einfach abbricht, dann erreicht das System einen Fixpunkt. An diesem verändert sich $P^n$ nicht weiter. Ein Vergleich zwischen den verschiedenen Präferenzmengen kann dazu dienen, das Erreichen des Fixpunkts abzuschätzen.

Wesentlichen Einfluss auf die Nutzbarkeit der Verfeinerung der Ergebnismenge hat der Parameter $k$. Bei einem hohen Wert enthält einerseits die aus einem Top-$k$-Rank erzeugte Präferenzmenge viele Elemente, was dazu führt, dass der Nutzer eine Vielzahl dieser bewerten muss. Andererseits ermöglicht eine hohe Anzahl geprüfter Präferenzen ein schnelles und hochwertiges Lernen durch den Algorithmus ohne häufige Iterationsschritte. Demzufolge muss $k$ so gewählt werden, dass der Nutzer nicht überfordert wird, da er prinzipiell wenig Aufwand für die Interaktion mit dem System aufbringen möchte [SP05].

## 7 Experimente

Der diskutierte Ansatz wurde anhand des vorgestellten Beispiels 1.1 experimentell überprüft. Die Experimente wurden auf einem aktuellen Notebook[10] mit 456 Kameradatensätzen durchgeführt. Die Algorithmen sind in Java 5 implementiert und können auf Wunsch zugesandt werden. Grundlage der Experimente bilden jeweils 1000 Durchläufe.

Das erste Experiment analysiert, wieviele automatisch aus einem Rank abgeleitete Präferenzen tatsächlich nützlich sind (siehe Abs. 5.2), wobei alle Gewichte auf 1 gesetzt wurden. Abb. 11 zeigt, dass ca. 62% der Präferenz nützlich sind, so dass eine Reduktion also sinnvoll ist.

Abb. 12 zeigt die Laufzeitentwicklung des Lernalgorithmus. Diese hängt hauptsächlich von zwei Parametern ab: der Anzahl der Präferenzen und den Gewichtsvariablen innerhalb der Anfrage. Bemerkenswert ist hier, dass die Laufzeit für weniger als 10 Gewichtsvariablen und weniger als 40 Präferenzen unter einer Sekunde liegt. Eine höhere Anzahl der Parameter ist aufgrund des Nutzerverhaltens kaum zu erwarten [SP05], erhöht jedoch die Laufzeit nur in einem vernünftigen Rahmen.

[10] Intel Core 2 Duo 2.5 GHz, 4 GB RAM, Mac OS X 10.5.5, Java 5

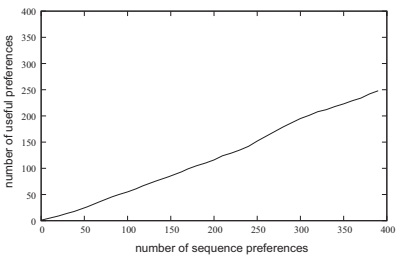



Abbildung 11: Anzahl nützlicher automatisch abgeleiteter Präferenzen

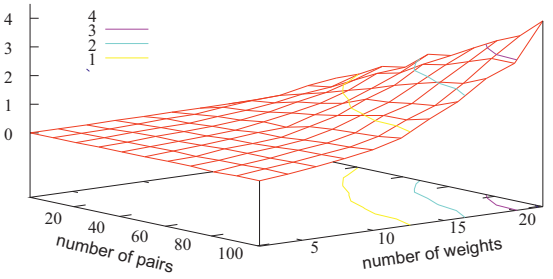



Abbildung 12: Laufzeit [s] in Abhängigkeit der Anzahl von Gewichten ($weights$) und Präferenzen ($pairs$)

Bezüglich der Nutzerinteraktion ist besonders interessant, wie schnell der Nutzer zu einem zufriedenstellenden Ergebnis bzgl. seiner Anfrage gelangt. Ziel des zweiten Experiments ist deshalb die Untersuchung, wieviele Präferenzen der Anwender überprüfen muss bis der Ziel-Rank erreicht ist. Wir gehen hier idealerweise von konsistenten, also widerspruchsfreien Präferenzen aus. Anhand einer zufälligen Gewichtung wurden die Top-$k$-Objekte berechnet und anschließend, ausgehend von einer zufälligen Initialgewichtung, überprüft, wieviele Präferenzen der Nutzer sehen oder korrigieren muss bis sein Ziel-Rank erreicht ist. Hierbei kam das Beispiel 1.1 in zwei Top-$k$-Varianten zum Einsatz. Einmal mit $k = 10$ und einmal $k = 20$.

Bei den Präferenzen wird bei diesem Experiment in gesehene und korrigierte unterschieden, wobei letztere durch den Nutzer umgekehrt wurden. Die Abbildungen zeigen jeweils zwei Kurven. Die obere gibt an, wieviele Zielobjekte bereits gesehen wurden und die untere welche Objekte im aktuellen Durchlauf unter den Top-$k$ liegen. Die Kurven konvergieren in Richtung von $k$, jedoch fällt auf, dass sie nicht streng steigen, was sich auf die Verwendung von Zufallszahlen im Algorithmus zurückführen lässt.

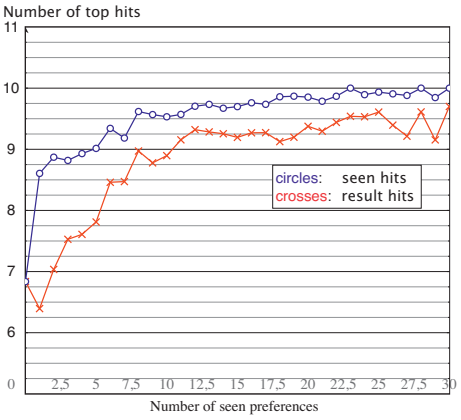



Abbildung 13: Top-10, gesehene Präferenzen

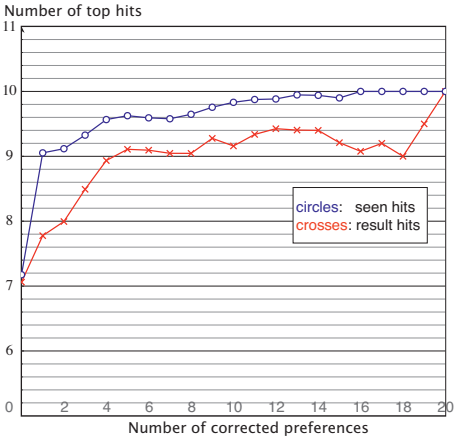



Abbildung 14: Top-10, korrigierte Präferenzen

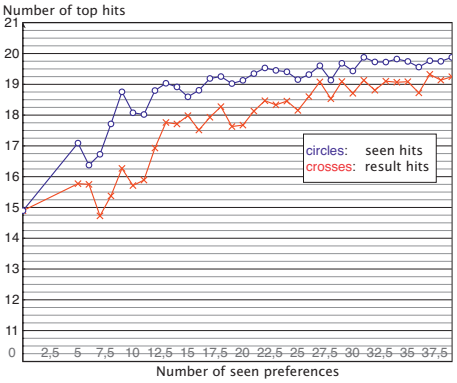



Abbildung 15: Top-20, gesehene Präferenzen

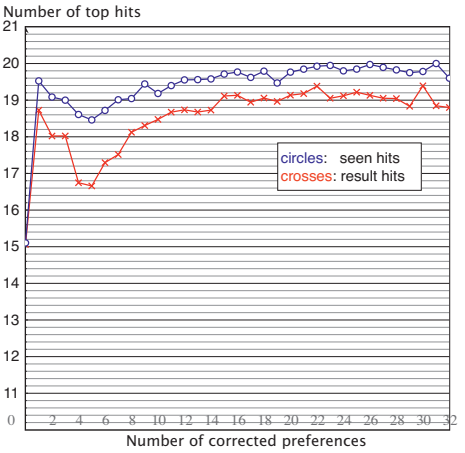



Abbildung 16: Top-20, korrigierte Präferenzen

Es fällt auf, dass in vielen Fällen bessere Resultate erzielt werden, wenn keine Präferenzen anstelle weniger definiert sind (siehe Abb. 13 - 16). Es kann folglich vermutet werden, dass die Angabe weniger Präferenzen die Semantik der Initialgewichte erst zerstört aber noch nicht ausreichend für eine neue Semantik ist. Offenbar muss erst eine kritische Masse an Präferenzen definiert werden, um die Suchqualität zu verbessern. Weniger überraschend ist die Erkenntnis, dass relativ zu einem höheren $k$ mehr Präferenzen benötigt werden, um die Suchqualität zu verbessern.

# 8 Ausblick

Aufbauend auf den vielversprechenden, experimentellen Ergebnissen muss das vorgestellte Verfahren erweitert werden. Im Schwerpunkt der Forschung steht dabei die Entwicklung einer Benutzeroberfläche, die den Anwender insbesondere bei der Verfeinerung seiner Anfrage unterstützt. Grundlage bildet dabei die Ausnutzung der Präferenzregionen und Gewichtsmaxima, um dem Nutzer sinnvolle Empfehlungen bei der Verfeinerung zu geben.

Das Finden einer Menge guter Präferenzen ist dabei mit dem Problem der Generierung von Entscheidungsbäumen verwandt, für das zahlreiche Algorithmen bekannt sind [BA97]. Gute Präferenzen, die dem Nutzer vorgeschlagen werden können, sind solche, welche den Schnitt bereits existierender Präferenzregionen halbieren. Anhand dieser Präferenzen kann der Nutzer seine Anfrage zielgerichtet steuern und so ein Verständnis für das konzeptionelle Modell des Systems gewinnen. Außerdem muss das Verhalten des Verfahrens bei wenig Präferenzen näher untersucht werden.

Zur Evaluierung des Lernens von Gewichten müssen weitere Algorithmen untersucht werden, die eine feinere Regulierung des Lernverfahrens ermöglichen. Denkbar sind hier genetische Algorithmen, deren Selektionskriterien sich an Nutzeranforderungen und Anwen-

dungskontexte anpassen.

Im Rahmen der Entwicklung eines integrierten Prototyps, welcher für Nutzertests unabdingbar ist, werden parallel geeignete Interface-Metaphern und Kommunikationsmechanismen analysiert, um dem Anwender ein vorhersehbar reagierendes, ihn unterstützendes Werkzeug zur Seite zu stellen. Hierbei werden auch Bereiche beachtet, welche die Anfrageformulierung nicht direkt tangieren, sondern einem ganzheitlichen Suchbegriff zuzuordnen sind. Denkbar sind hier unterstützende Recherchewerkzeuge, wie Ablagen, die gefundene Objekte aufnehmen, welche als Nebenprodukt der Hauptsuche anfallen und beispielsweise als Ausgangspunkt einer neuen Suche dienen können.

## Literatur


[ACDG03] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das und Aristides Gionis. Automated Ranking of Database Query Results. In *CIDR*, 2003.

[BA97] Leonard A. Breslow und David W. Aha. Simplifying decision trees: A survey. *Knowl. Eng. Rev.*, 12(1):1–40, 1997.

[BG93] Vicki Bruce und Patrick R. Green. *Visual Perception –physiology, psychology and ecology (2nd ed., reprinted)*. Lawrence Erlbaum Associates, Publishers, Hove and London, UK, 1993.

[BKS01] Stephan Börzsönyi, Donald Kossmann und Konrad Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering*, Seiten 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[BN36] G. Birkhoff und J. von Neumann. The Logic of Quantum Mechanics. *Annals of Mathematics*, 37:823–843, 1936.

[BP95] P. Bosc und O. Pivert. SQLf: A Relational Database Language for Fuzzy Querying. *IEEE Transactions on Fuzzy Systems*, 3(1):1–17, Februar 1995.

[Cho03] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[Cho07] Jan Chomicki. Database querying under changing preferences. *Ann. Math. Artif. Intell.*, 50(1-2):79–109, 2007.

[CMPT00] Paolo Ciaccia, Danilo Montesi, Wilma Penzo und Alberto Trombetta. Imprecision and User Preferences in Multimedia Queries: A Generic Algebraic Approach. In K.-D. Schewe und B. Thalheim, Hrsg., *FoIKS: Foundations of Information and Knowledge Systems, First International Symposium, FoIKS 2000, Burg, Germany, February 14-17, 2000*, Jgg. 1762 of *Lecture Notes in Computer Science*, Seiten 50–71. Springer, 2000.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan und Gerhard Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *CIDR*, Seiten 1–12, 2005.

[CW08] Claremont Workshop. The Claremont Database Research Self Assessment. Bericht, 2008.

[FW00] R. Fagin und E. L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the 28th Int. Conf. on Very Large Data Bases, VLDB'02, Hong Kong, China, August, 2002*, Seiten 311–322. Morgan Kaufmann Publishers, 2002.

[KN07] Aljoscha Klose und Andreas Nürnberger. On the Properties of Prototype-based Fuzzy Classifiers. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 37(4):817–835, 2007.

[KRR02] D. Kossmann, F. Ramsak und S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. of the 28th Int. Conf. on Very Large Data Bases, VLDB'02, Hong Kong, China, August, 2002*, Seiten 275–286. Morgan Kaufmann Publishers, 2002.

[Lee94] Joon Ho Lee. Properties of Extended Boolean Models in Information Retrieval. In SIGIR, Hrsg., *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, Seiten 182–190. Springer–Verlag New York, Inc., 1994.

[LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba und Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.

[NM65] J. A. Nelder und R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[RJ05] Lawrence A. Rowe und Ramesh Jain. ACM SIGMM retreat report on future directions in multimedia research. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(1):3–13, 2005.

[Roc71] J. J. Jr. Rocchio. Relevance Feedback in Information Retrieval. In G. Salton, Hrsg., *The SMART Retrieval System - Experiments in automatic Document Processing*, Kapitel 14, Seiten 313–323. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.

[SB88] Gerard Salton und Chris Buckley. Improving Retrieval Performance by Relevance Feedback. Bericht, Ithaca, NY, USA, 1988.

[Sch08] Ingo Schmitt. QQL: A DB&IR Query Language. *The VLDB Journal*, 17(1):39–56, 2008.

[Sel59] O. G. Selfridge. Pandemonium. A paradigm for learning. *The mechanics of thought processes*, 1959.

[SFW83] Gerard Salton, Edward A. Fox und Harry Wu. Extended Boolean Information Retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[SP05] Ben Shneiderman und Catherine Plaisant. *Designing the user interface: Strategies for effective human–computer interaction*. Pearson, Boston, 4. ed.. Auflage, 2005.

[SS03] N. Schulz und I. Schmitt. Relevanzwichtung in komplexen Ähnlichkeitsanfragen. In G. Weikum, H. Schöning und E. Rahm, Hrsg., *Datenbanksysteme in Business, Technologie und Web, BTW'03, 10. GI-Fachtagung, Leipzig, Februar 2003*, Lecture Notes in Informatics (LNI) Volume P-26, Seiten 187–196, Bonn, 2003. Gesellschaft für Informatik.

[SS04] I. Schmitt und N. Schulz. Similarity Relational Calculus and its Reduction to a Similarity Algebra. In Dietmar Seipel und J. M. Turull-Torres, Hrsg., *Third Intern. Symposium on Foundations of Information and Knowledge Systems (FoIKS'04), Austria, February 17-20*, Jgg. 2942 of *Lecture Notes in Computer Science*, Seiten 252–272. Springer-Verlag Berlin Heidelberg, 2004.

[SZN08] Ingo Schmitt, David Zellhöfer und Andreas Nürnberger. Towards Quantum Logic Based Multimedia Retrieval. *Annual Meeting of the North American Fuzzy Information Processing Society, 2008*, Seiten 1–6, 2008.

[Wei07] Gerhard Weikum. DB&IR: both sides now. In SIGMOD, Hrsg., *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Seiten 25–30. ACM, 2007.

[Zad88] Lofti A. Zadeh. Fuzzy Logic. *IEEE Computer*, 21(4):83–93, April 1988.

[Zie05] Martin Ziegler. Quantum Logic: Order Structures in Quantum Mechanics. Bericht, University Paderborn, Germany, 2005.

# Bringing BLINK Closer to the Full Power of SQL

Knut Stolze* Vijayshankar Raman† Richard Sidle† Oliver Draese*

**Abstract:** BLINK is a prototype of an in-memory based query processor that exploits heavily the underlying CPU infrastructure. It is very sensitive to the processor's caches and instruction set. In this paper, we describe how to close two major functional gaps in BLINK, which arise from real-world workloads. The manipulation of the data maintained by BLINK require specialized data structures. Another aspect being discussed are NULLs and how to avoid the three-valued logic during query processing.

## 1 Introduction

BLINK [RSQ+08] is a query processor that was designed to answer all queries in constant or near-constant time with respect to query complexity. This goal is accomplished by always running a highly efficient table scan over all the data. The data is stored denormalized in main memory so that no I/O operations can disrupt the query execution time. The data is highly compressed to provide for more raw data being stored in main memory and to reduce the amount of data flowing from memory to CPU during scans. At the same time, the compression scheme results in long runs of fixed-length values, which can be efficiently scanned with the CPU's vector operations without having to decode the data. A BLINK single system in a cluster shall store and process about 100 GB of uncompressed data in main memory.

A prototype demonstrates the feasibility and proves the constant query execution time. This has been achieved without requiring a performance tuning layer on which traditional DBMSs rely heavily. No indexes, materialized views, or a wide variety of tuning and configurations options are necessary. Still, BLINK has shown to improve the query execution time by one or more orders of magnitude compared to highly tuned (or even self-tuning) traditional relational database systems like MonetDB [BMK99].

In this paper, we present our ongoing efforts to extend the functionality of BLINK. First, we recapitulate in section 2 the primary techniques that were used in the query processor and highlight the current gaps in the functionality of the query engine. This serves as a motivation for the subsequent section 3, where we describe how data is organized physically in the main-memory structures. Those data structures need to be tailored to the envisioned usage of BLINK as a full function query engine for data warehousing that can also handle data modifications efficiently. Today's RDBMS all support SQL NULLs to represent unknown or not-applicable values. Section 4 describes our mechanisms how NULLs are represented in BLINK while still maintaining the highly efficient query processing.

*IBM Germany Research & Development, Böblingen, Germany
†IBM Almaden Research Center, San Jose, CA, USA

## 2 Background on BLINK

BLINK stores all data in main memory to avoid any I/O overhead. Since main memory is not in abundant supply as secondary or ternary storage (i. e. disk or tape), it is necessary to compress the data in memory.

We built BLINK on top of previous work on entropy compression of relations [RS06]. Values are encoded with variable length codes (shorter codes for more frequent values). Query-time scans have to parse each code in each tuple individually because the beginning of the next code depends on the length of the previous codes. The cost can be significant, especially because denormalized tables tend to have many columns. This overhead is a well-known problem, pointed out in [WKHM00, HRSD07]. For the same reason, most of other related work on fast querying over compressed data relies on fixed length codes. For example, Sybase IQ [MF04] and C-Store [SAB+05] use fixed length codes and leverage the homogeneity to do fast array-based operations.

### 2.1 Frequency Partitioning

BLINK uses a compression technique called *Frequency Partitioning* that still compresses close to entropy but produces long runs of fixed-length codes for efficient scanning. Our compression scheme amortizes the work of computing code lengths by grouping together tuples that have the same pattern of fieldcode lengths. To achieve this, we partition tuples coarsely by the occurrence frequency of their column values, and assign fixed-length codes within each partition. The encoding is dictionary-based. An encoded value is called *fieldcode*. The fieldcodes of a tuple are concatenated and form the *tuplecode*.

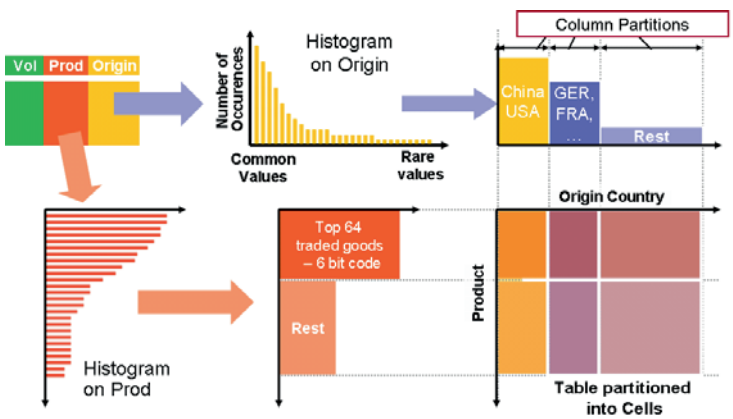



Figure 1: Illustration of Frequency Partitioning

Figure 1 illustrates this on a three-column table where the columns "Prod" and "Origin" are partitioned. We start by building separate histograms for both columns. Each his-

togram collects the occurrence frequencies over all distinct values. We use the histograms to form disjoint *column partitions* – C1a, C1b for column "Prod" and C2a, C2b, C2c for column "Origin", – according to their occurrence frequencies. Each combination of column partitions, e. g. (C1a, C2b), forms a partition of the table, called a *cell*. Separate dictionary of values for each partition of "Prod" and "Origin" are created and fixed length codes assigned. The dictionaries are used to encode the tuples of the table, so every tuple in a given cell is guaranteed to have the same pattern of fieldcode lengths and, thus, the same tuplecode length.

In general, say table $R$ has $n$ columns. Let $R^i$ be the domain of possible values for column $i$, so the domain of $R$ tuples is $\times_i R^i$. Each $R^i$ is partitioned into $p_i$ partitions $R^i_1 \dots R^i_{p_i}$, with $\cup_{1 \le j \le p_i} R^i_j = R^i$, such that values with similar frequency are clustered in the same partition. This partitioning of columns induces a partitioning of $R$ into cells. Each cell is labeled with an $id \in \times_i [1, p_i]$. Given a cell with id $(\theta_1, \dots, \theta_n)$, column $i$ has only $|R^i_{\theta_i}|$ values, and is given a fixed length code of $\lceil \lg |R^i_{\theta_i}| \rceil$ bits. The tuplecode is $\sum_i \lceil \lg |R^i_{\theta_i}| \rceil$ bits long. The codes are assigned to be order-preserving: higher values get higher codes.

### 2.2 SIMD Predicate Evaluation

At query time, it is necessary to scan all the data in BLINK and apply predicates to identify only matching tuples. Since BLINK uses order-preserving codes, we can apply equality and range predicates directly on the encoded data without any decoding. We evaluate all conjunctive equality and range predicates in parallel with a constant number of processor instructions, eliminating loops and branches. Due to the fixed-length tuplecodes in each cell, it is possible to apply bit masks on the tuples to extract only the fieldcodes to which a predicate is applied. The CPU's vector operations (single instruction multiple data (SIMD)) are exploited to operate on multiple tuplecodes in parallel [JRSS08].

### 2.3 Gaps in BLINK

The implementation of BLINK was designed to work on a single system only. Scaling it up beyond the system limits requires grid or cluster technologies to distribute data and workload across the then-available machines and to coordinate between those. Additionally, query processing has to be distributed as well. Each system produces only a partial result, and this has to be combined before further processing is applied on it. For example, a SQL query using an `ORDER BY` or `HAVING` clause can only be computed on the combined result set. The distribution of queries processing in a cluster is not further discussed.

A second gap in BLINK is that the system was tailored to query processing only. The aspect of maintaining the data has not yet been further investigated. Section 3 dives into the details of the data structures used to efficiently add or remove tuples to or from tables scanned by the query engine.

Thirdly, the data used in the performance measurements presented in [RSQ+08] did not include SQL NULLs. However, NULLs are often found in data warehouses, and many compression techniques actually take advantage of that. *Sparse* data cubes contain many NULLs in the cells of the cube, i. e. the values in the fully denormalized schema. Therefore, we explain in section 4 in more detail how NULLs will be handled in a very efficient manner in BLINK.

## 3 Data Structures for Data Modifications

A query processor supporting data modifications, i. e. `INSERT`, `UPDATE`, and `DELETE` operations has to ensure the integrity and consistency of the data. BLINK, being positioned for data warehouse applications, faces different requirements than OLTP systems. Data modifications do not trickle in. Data warehouses use batch ETL (extract, transform, load) that collected data changes from source systems, transform it (including aggregation and cleansing), and finally load the data. Thus, non-concurrent mass operations to roll-in/out batches of tuples are the sweet spot that need to be handled efficiently by BLINK as well.

### 3.1 Snapshot Semantics

A quick design decision was to avoid a lock manager – instead, snapshot semantics are used. Each tuple has an associated validity interval, stating when the tuple was added and when it was finally deleted. A reorganization can be run to eventually clean up deleted tuples at some point. The validity interval could be timestamp based, which is rather expensive storage-wise. A simple counting mechanism (called *epoch*) is sufficient. Each time a data modification is applied to the data managed by BLINK, this counter is incremented. Thus, a (denormalized) table $R$ is extended by two additional columns $e_s$ and $e_e$, where $e_s$ indicates the start epoch and $e_e$ the end epoch of the validity interval $[e_s, e_e)$.

When a data modification are initiated, the current epoch counter $ce$ for the system is determined and used for all processed tuples. For all new tuples, $(e_s, e_e)$ is set to $(ce+1, \infty)$. A tuple, thus prepared, is simply appended to the table. Deleting tuples is implemented with a specialized query, which does not only scan the data and returns the tuples found. It also modifies matching tuples and sets $e_e$ to $ce$ so that queries starting after the update operation completed exclude those tuples. The global current epoch counter is incremented at the end of the modification processing, at which point new queries will pick up the changed value for $ce$ and the modifications become visible.

Insertion and deletions can be done concurrently and lock-less to other queries. Again, $ce$ is determined for each query, and the query is augmented with two additional predicates:

$$\ldots \wedge e_s \leq ce \leq e_e$$

The predicate on $e_s$ excludes all tuples that may currently be inserted since $e_s$ is larger

than the query's $ce$ for those tuples. Likewise, the predicate on $e_e$ excludes all tuples that were deleted before the query started while still retaining the tuples that are currently in the process of being deleted.

### 3.2 Dictionary Based Coding

BLINK determines the optimal partitioning of the denormalized table into cells. The partitioning used a greedy algorithm that calculates the gain achieved in terms of code length if the number of partitions is increased for a column. The frequency histograms are exploited to determine the code length and find optimal splitting points, i. e. the points where a new column partitions should start. The column that contributes the largest gain gets a further column partition. This iterative approach is repeated until the maximum number of cells exceeds a given threshold.

The underlying assumption in that approach is that all data is already available when deriving the frequency histograms. Neither changes in the data distribution nor the addition of completely new values (for which no dictionary did yet exist) could be handled. In order to address those short-comings, we revisit the general partitioning approach and extend it.

**Regular Partition.** Originally, BLINK had only regular column partitions. All such partitions had an associated dictionary that contained the fieldcodes and their uncoded values. The ordering of fieldcodes in such a dictionary was determined by the ordering of the uncoded values. Those column partitions become *regular partitions*. The regular partitions are determined when the data is initially loaded into BLINK or when a reorganization is done. The cross product of column partitions $p_i$ over all columns $i$ determines all available regular cells, which results in $\prod^i p_i$ cells.

Encoding new values in such dictionaries only works in rare borderline situations if:

1. The dictionary has not yet reached its capacity. Dynamically growing dictionaries is not possible due to the fixed-length codes, which allows only the encoding of $2^n$ values where $n$ is the length of the field codes in the column partition.

2. The new value to be added is larger than the largest value in the dictionary. Otherwise, the order-preserving property of the dictionary would be violated.

Therefore, we introduce *extension partitions* and *catch-all partitions* for each column to address this issue, which are processed like regular partitions, except for fast SIMD scans.

**Extension Partition.** All values of a column that cannot be encoded in a regular partition qualify for the single extension partition, i. e. new distinct values added after the dictionaries for regular partitions have been built. An extension partition is still backed up by a dictionary. The dictionary is not necessarily order-preserving.[1] But the values are

[1] If the extension partition uses pure offset coding, its dictionary will be trivially order-preserving, of course.

still encoded, which means compression techniques (cf. section 3.3) can be applied. The dictionary for the extension partition is allocated with such a size that it can accommodate at least as many distinct values as are present in the regular partitions.

The extension partitions of all columns form a single *extension cell*. Extension partitions will never be combined with regular partitions to always guarantee efficient scan processing of range predicates. As soon as one value in a tuple has to be encoded in the extension partition of its column, the whole tuple will go to the extension cell.

The values in an extension partition are either purely offset-coded or pure dictionary-coded, but not a combination of both (cf. section 3.3). The main purpose of an extension partition is to provide a graceful degradation in compression. But once the extension partition is filled, any new values will be placed into the column's catch-all partition.

**Catch-All Partition.** Each column has a single catch-all partition that stores all its values in an uncoded format. Thus, no advantage of any compression can be achieved. It is used as a last resort to still be able to manage such data at all, even if no efficient evaluation will be possible. All values in such a partition are either identity coded or uncoded (cf. section 3.3), depending on the column length.

Comparable to extension cells, there is only a single *catch-all cell*, which is comprised of the combination of the catch-all partitions of all columns in the table. Once one value in some column $c_i$ cannot be encoded in regular cells and also not in the extension partition of $c_i$, it falls into the catch-all partition and forces the whole tuple to be placed there.

### 3.3 Partition Encoding Types

A value in a tuple can be encoded in different ways. The actual encoding depends on the type of the partition (regular, extension, or catch-all) and the dictionary associated with the respective partition. In all cases, it is first attempted to encode the value in a regular partition, preferring partitions with shorter code lengths.

**Offset Coding.** Per default, a dictionary uses offset coding. The fieldcode of a value is comprised of the combination of the dictionary index and an offset. The length of the fieldcode is the number of bits needed for the dictionary index plus the number of bits encoding the offset. Both of those numbers could be 0, resulting in pure dictionary coding or pure offset coding. A value $v$ encoded to fieldcode $fc$ can be reconstructed by:

$$v = decode(fc) = dictionary[fc.dictIndex] + fc.offset$$

The addition of the offset can take two different forms. In *prefix coding*, uses a bit-wise concatenation. This is used for floating point numbers, for example, that may share a common exponent but have a differing mantissa. Applying a mathematical addition to floating point numbers may give an incorrect result [Gol91]. A concatenation of bits reconstructs the exact same bit pattern of the original value.

For integer types, it makes sense to apply *minus-coding* where the offset is combined with the dictionary value using arithmetic addition. Thus, if the dictionary bits in the fieldcode identify a value of 100 in the dictionary and the offset bits are 32, the reconstructed value will be 132 – regardless of the actual bit pattern of that number.

Dictionaries using offset coding are always order preserving. Fast scans over the data in code space can be applied to cells that are associated to such a dictionary.

**Identity Coding.** Identity coding uses the uncoded value itself as fieldcode. It is very similar to uncoded values. However, there is still a dictionary associated with such a column partition (and it is trivially order-preserving). The query engine can treat such fieldcodes like purely offset coded values and no special-casing is needed in the code.

This encoding scheme can only be applied if the actual values are short enough to fit into processor registers, i. e. do not exceed the maximum length for fieldcodes. Otherwise, SIMD instructions could not be applied to do the fast scanning BLINK relies on heavily. Identity coding is especially useful for epoch columns as described in section 3.1, for which the actual values and their skew are not known in advance.

**Uncoded.** There are cases when values cannot be encoded at all or when it does not make sense to do so. Uncoded values require a dedicated storage layout.

Use cases are string columns with very high cardinality of distinct values. Such columns would have huge dictionaries with virtually no (or even a negative) compression ratios. Once BLINK determines those properties for a column, the column is marked as being uncoded and the values are stored as-is without being supported by a dictionary. Another use for not coding values are catch-all partitions where identity coding is not an option (due to the processor register limitations).

### 3.4 Banks and Tuplets

Internally, BLINK uses highly specialized storage structures for each cell. A cell is broken down into multiple *cell blocks*. A cell block defines a work unit for the query engine. Since multiple threads perform the scanning, the further break down of cells into cell blocks helps to balance the overall workload over all threads and facilitates scan sharing.

Cell blocks are partitioned into *banks*. Each bank contains a projection of a tuple, called a *tuplet*. A cell block contains a mixture of row store and column store, i. e. it is a hybrid of row-major and column-major order. Figure 2 illustrates the bank layout of tuples. The banks define a vertical partitioning. The arrows in the figure indicate the order in which the data is stored in the bank and in the cell block.

Columns are assigned to banks in a cell-specific manner. The deciding criteria are the data type of the column and whether it participates in a reference (foreign key) used to define the join conditions during denormalization. A column is assigned either to a *vertical bank* or a *horizontal bank*. A vertical bank is intended for columns on which typically predicates

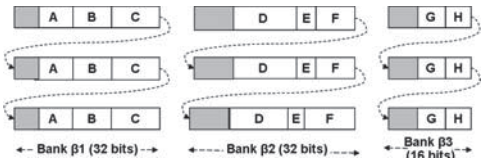



Figure 2: Bank layout

are evaluated while horizontal banks are used for *measure columns* on which aggregation is performed, typically.

Each vertical bank is a separate contiguous unit in the cell block, approximating a column store. Bin-packing is applied to minimize padding as much as possible. Vertical banks can vary in size. One constraint is that the bank width is at most four times the fieldcode length for each column in the bank. This is done to prevent having to scan too much unused data during predicate evaluation. Contrary, horizontal banks use a row store approach.

Uncoded banks, visualized in figure 3, deviate from that because the values may have varying length besides fixed-length values. In order to avoid sequential scans through the banks, offsets pointing directly to the uncoded value in the variable length bank are stored in the encoded portion of the tuplets.

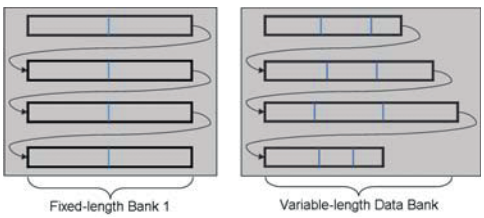



Figure 3: Layout for uncoded banks

## 4 Null Handling

NULLs were introduced in relational database systems as a method of representing missing data[ISO03]. For example, a table stores customer-related information like the first name, last name, and address of each customer. However, not all customers may have a middle name or middle initial – while others do.

NULLs are not considered to be values of the domain of the data, i. e. NULL is not a string or integer. The introduction of NULLs lead to three-valued logic for predicate evaluation. Only values in the same (or a compatible) domain can be compared with each other and give a well-defined result, namely the comparison is either `true` or `false`. Comparing a value with NULL is defined to be `unknown` [Mul07].

Todays relational database systems typically use internally a special marker for each value

in columns that may contain NULLs. For example, DB2 for z/OS [IBM07] prefixes the value of columns with a byte where `x'00'` means NOT NULL and `x'FF'` represents NULL. For varying length columns this null marker byte appears after the length of the value and it is included in the length. Using such an approach implies that all predicates specifically have to take care of NULLs by enhancing each predicate by adding a verification to test whether the marker is set or not, which causes branching in the evaluation code.

**Avoiding Three-Valued Logic.** In BLINK, we break with the notion to treat NULL as being excluded from the value's domain. Instead, the frequency of NULLs is explicitly counted in frequency histograms and a code in the column partition's dictionary (to which NULL belongs) is assigned as well. The dictionary entry for NULL (if present in the dictionary at all) will always be the first code, i. e. fieldcode 0. That way, it is possible to exclude or include NULL by simply adjusting the range predicates as table 1 shows for column $c_i$ and value $x$. No branching is necessary because regular range predicates can be applied.

| Predicate in Query | Predicate on fieldcodes |
|---|---|
| $c_i$ = $x$ | $fieldcode(c_i)$ BETWEEN $encode(x)$ AND $encode(x)$ |
| $c_i$ > $x$ | $fieldcode(c_i)$ BETWEEN $(encode(x)+1)$ AND $\infty$ |
| $c_i$ <= $x$ | $fieldcode(c_i)$ BETWEEN 1 AND $encode(x)$ |
| $c_i$ IS NULL | $fieldcode(c_i)$ BETWEEN 0 AND 0 |

Table 1: Handling of NULL in range predicates over fieldcodes

**Null Indicator Column.** We described in section 3.3 that not all values may be dictionary encoded but rather uncoded. For that case, it is necessary to still have the NULL marker. Another situation requiring such a marker occurs if the whole domain of column $c_i$ is consumed by values in the column and adding fieldcode 0 for NULL requires an additional bit. But increasing the number of bits is not always possible because the size of the encoded values may exceed the maximum bank width.

The NULL marker is not directly attached to each value. Instead, an internal *null indicator column* is added for each nullable column. Such a column is considered in the partitioning like any other column. Thus, it can be dictionary based and it will have at most two regular partitions. The partitions have 0 or 1 bit – 0 if all values in the column partition are the same, which is exploited if fieldcode 0 is used to represent NULL in the data column.

Predicate evaluation of cells that contain a non-dictionary encoded nullable column $c_i$ is implemented by augmenting the predicate for each such column $c_i$ to exclude tuples where the fieldcode in the corresponding null indicator column $c_j$ represents the NULL marker. It is assumed that NULL is represented by 1 in $c_j$.

$$fieldcode(c_i) \text{ BETWEEN } x \text{ AND } y \text{ AND } fieldcode(c_j) \text{ BETWEEN } 0 \text{ AND } 0$$

## 5 Summary

BLINK and its compression and scanning technology already came a long way to show that (near) constant query time processing is possible for data warehouse applications. We have shown that the gaps in BLINK's functionality can be closed. This concerns primarily the support for data modifications and NULLs. Using validity intervals, denoted by epoch columns allows the concurrent execution of DML and queries without requiring a locking mechanisms for all rows. Special attention had to be paid to the handling of new data that does not yet exist in the system. NULLs – despite not being a value of a column's domain – can be encoded like any other value and, thus, we can take advantage of SIMD instruction and avoid three-valued logic, except for some very rare cases for uncoded values.

## References


[BMK99] P. A. Boncz, S. Manegold, and M. L. Kersten. Database Architecture Optimized for the New Bottleneck: Memory Access. In *VLDB*, 1999.

[Gol91] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1), 1991.

[HRSD07] A. L. Holloway, V. Raman, G. Swart, and D. J. DeWitt. How to Barter Bits for Chronons. In *SIGMOD*, 2007.

[IBM07] IBM. *DB2 Version 9.1 for z/OS – Diagnostics Guide and Reference*, 2007.

[ISO03] ISO/IEC 9075-2:2003. *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, 2nd edition, 2003.

[JRSS08] R. Johnson, V. Raman, R. Sidle, and G. Swart. Rowwise Parallel Predicate Evaluation. In *VLDB*, 2008.

[MF04] R. MacNicol and B. French. Sybase IQ Multiplex – Designed for Analytics. In *VLDB*, 2004.

[Mul07] C. Mullins. Using Nulls in DB2, 2007. `http://www.db2portal.com/2006/04/using-nulls-in-db2.html`.

[RS06] V. Raman and G. Swart. Entropy Compression of Relations and Querying of Compressed Relations. In *VLDB*, 2006.

[RSQ+08] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-Time Query Processing. In *ICDE*, 2008.

[SAB+05] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A Column-oriented DBMS. In *VLDB*, 2005.

[WKHM00] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The Implementation and Performance of Compressed Databases. *SIGMOD Record*, 29(3):55 – 67, 2000.

# Towards Flash Disk Use in Databases— Keeping Performance While Saving Energy?

Theo Härder, Karsten Schmidt,
Yi Ou, and Sebastian Bächle
Databases and Information Systems Group
Department of Computer Science
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
{haerder,kschmidt,ou,baechle}@cs.uni-kl.de

**Abstract:** Green computing or energy saving when processing information is primarily considered a task of processor development. We, however, advocate that a holistic approach is necessary to reduce power consumption to a minimum. We explore the potential of NAND flash memory in comparison to magnetic disks for DBMS-based architectures. For this reason, we attempt to identify the IO performance of both storage types at the device level and file system interface, before we approach the most important question how DBMS applications can take advantage of the flash performance characteristics. Furthermore, based on an analysis of storage structures and related benchmark runs in our *XTC* system, we discuss how IO-intensive DBMS algorithms have to be adjusted to take most out of the flash technology which is entering the server area with dramatic pace. The ultimate goal is to substantially improve energy efficiency while comparable performance as in disk-based DB servers is maintained.

## 1 Introduction

Recently, green computing gained a lot of attention and visibility also triggered by public discussion concerning global warming due to increased CO2 emissions. It primarily addresses efforts to reduce power usage, heat transmission, and, in turn, cooling needs of hardware devices. Thus, research and development primarily focus on processor chips using extensive hardware controls. Thermal management, however, is a holistic challenge. It includes not only simultaneous optimizations in the materials, devices, circuits, cores, and chips areas, but also combined efforts regarding system architecture (e.g., energy-efficient storage) and system software (e.g., energy-optimal algorithms). Moreover, virtualization of all system resources enables an energy-aware system management, e.g., by launching a control center which monitors workload changes and may respond by allocating/deactivation entire server machines. Even supply chains are considered from this perspective to minimize their "carbon footprint"[1].

Up to now, research has hardly contributed to this important overall goal in the database management system (DBMS) domain. But, NAND flash memory (also called solid state

[1] "SAP optimizes CO2 balance of supply chains", Computer Zeitung, Sept. 2008

disk or flash, for short) has the potential to become the future store for permanent database data [4]; it promises—compared to magnetic disks (disk, for short)—breakthroughs in bandwidth (IOps), energy saving, reliability, and volumetric capacity [6].

So far, flash storage was considered ideal for keeping permanent data in embedded devices, because it is energy efficient, small, light-weight, noiseless, and shock resistant. So, it is used in personal digital assistants (PDAs), pocket PCs, or digital cameras and provides the great advantage of zero-energy needs, when idle or turned off. In these cases, flash use could be optimally configured to specific single-user workloads known in advance.

Because NAND flash memory is non-volatile, allows for sequential and random block reads/writes, and keeps its state even without energy supply, it can be compared to disks and can possibly take over the role of disks in server environments. However, not all aspects important for single-use devices count for DB servers and not all out-of-the-box DBMS algorithms can be directly applied and guarantee adequate performance when processing data stored on flash.

In this paper, we want to explore whether flash disks—from a performance point of view—can take over the role of or even replace magnetic disks in DB server environments. In Section 2, we summarize the potential of flash storage as a competitor of disks in such environments. While decades ago the device-related IO performance completely determined the cost of application IOs, nowadays a variety of built-in optimization mechanisms in the IO path of DBMS applications may relieve the real IO overhead and improve the related IO cost suchlike that the plain device characteristics may only play a minor role. For this reason, we evaluate in Section 3 the device-related IO performance of both competitors, where the influence of the OS file system is avoided as far as possible. In Section 4, more detailed IO performance tests are conducted at the file level to identify the influence of the OS and the file system. To figure out the discrepancy of flash and disk efficiency at the DBMS level in Section 5, we outline three native storage mappings of XML documents available in *XTC*, our native XML DBMS [12]. These storage mappings serve to discriminate the number of IO requests needed when processing XML workloads and help to reveal in Section 6 the DBMS-related IO performance for the storage devices compared. Finally, we throw a glance at the question of energy saving, before we conclude our findings in Section 7.

## 2 Flash Usage in DB Servers?

Usually, fixed-size *blocks* are mapped to external storage and are the units of IO (data transfer of a byte sequence) from an external device to the OS. Such blocks, in turn, are stored on disk as a consecutive sequence of smaller units, called segments typically consisting of 512 B. Storage representation is organized differently on flash. The entire storage space is divided into $m$ equal *flash blocks* typically much larger than disk blocks. A flash block normally contains $k$ (32–128) fixed-size *flash segments*, where a segment ranges between 512 B and 2 KB. Because of the NAND logic, an individual segment cannot be directly updated, but needs *erasure* (reset) of the flash block it is contained in (automatically done by the flash device), before it is rewritten. The flash segment is the smallest and the flash block the largest unit of read, whereas the flash segment is the unit of write; using chained

IO, the DBMS, however, can write $1<i\leq k$ flash segments into a block at a time. Note, whenever a flash segment is written in-place, *wear leveling* [1] automatically allocates a new flash block and moves to it all segments from the old flash block together with the updated segment (keeping an existing cluster property). Restricted *write endurance*, referring to the maximum possible erase cycles—between 100,000 (older references) and 5,000,000 (most recent references)—, would contradict general server usage. However, plenty of mapping tricks are provided [16] or it even seems to be a non-problem[2].

### 2.1 Read/Write Models

Disks are devices enabling fast sequential block reads and, at the same time, equally fast writes, whereas random block read/writes are much slower (requiring substantial "mechanical time"). The block size can be configured to the needs of the DBMS application with page sizes typically ranging between 4 KB and 64 KB. To hide the access gap between memory and disk, a DBMS manages a large DB cache in memory (RAM) where each cache frame keeps a DB page which, in turn, is mapped to a disk block. In most DBMSs, page propagation uses update-in-place applying WAL [8].

### 2.2 Flash vs Disk Performance

In the literature, there is a unanimous agreement on the following facts [4, 6, 15]: Disks are excellent *sequential* stores, whereas random access is only improving for larger page sizes. On the other hand, flash offers generally better sequential/random read (SR/RR) and comparable sequential write (SW), but terrible random write (RW) performance:

- SR and SW having a bandwidth of more than 60 MBps for page sizes $\leq$ 128 KB are superior, but remain comparable to those on fast disks.
- RR is spectacularly faster for typical DB page sizes (4–32 KB) by a factor of 10–15 ($\geq$3000 to $<$200 IOps).
- RW performs worst with $\sim$50 IOps, opening a huge advantage for RR, and is slower by a factor of 2–4 compared to disks (for the page size range considered).

Technology forecast is even more optimistic: J. Gray [7] claimed a flash read bandwidth of up to 5000 random reads per second or an equivalent of $\sim$ 200 $\mu$sec per 8KB page. Furthermore, an internal Microsoft report [6] predicts that about 1,100 random 8KB writes per second seem possible with some flash disk re-engineering.

This picture is complemented by the energy consumption: The power needed to drive a flash read/write is $\sim$ 0.9 Watt (in our case) and, hence, by a factor of $>$15 lower than for a disk. Using the figures for IOps, we can compute IOps/Watt as another indicator for energy-saving potential. Hence, 3,100 flash-reads or 55 flash-writes can be achieved per Watt, whereas a disk only reaches 13 operations per Watt ([6], [18]).

[2] A worst-case thought experiment [14] with repetitive sequential block overwrites on a flash indicates that 51 years are needed to reach the wear leveling limit.

For more than a decade, NAND flash chips doubled their densities each year and currently provide 64 Gbit. This growth will continue or accelerate such that 256 GB per chip are available in 2012 [13]. Because several of these chips can be packaged as a "disk", we should be prepared for flash drives with a capacity of several Terabytes in this near future. Of course, flash is still quite expensive. A GB of a flash device amounts to >3$ (but only <0.2$ for disk), but technology forecast predicts rapid market growth for it and, in turn, a dramatic decrease in the near future that it will roughly reach the disk[3] [5, 15].

## 3 Device-Related IO Performance

In contrast to impressive flash behavior, performance gains to be anticipated when integrating flash devices into a DBMS are not easy to obtain. After a long period of experiments under DBMS control—painful because of results which could not be interpreted at all—, we started from the scratch to explore step-by-step the influences of built-in optimization mechanisms in the IO paths of blocks read from or written to external storage.

### 3.1 Historical Note

40 years ago, timings of disk access from the file interface was more or less (very close to) those of raw-disk access, because all block transfers from the plain disk were directly done via a channel to some address space in RAM. There was no disk cache available and further abstractions (such as file buffers) achieved by the OS file system were not present. Therefore, it was rather simple to compute disk access times for various kinds of block transfers. Technology, device, and usage parameters, provided by the disk manufacturer, were sufficient to calculate for given access requests the expected block access times. For this purpose, the disk access time $t_d$ of a block of size $b$, where $x$ specified the number of cylinders to be crossed from the current access arm position to the cylinder carrying the block requested, was computed by

$$t_d(x,b) = t_{SIO} + t_s(x) + t_r + t_{tr}(b).$$

Here, $t_{SIO}$ is typically in the range of $\mu$sec and stands for the CPU overhead (OS supervisor call, device activation, etc.) until device-related activities can begin. The lion share of the access time was the disk seek time (access motion time) $t_s(x)$, which was specified by the manufacturer as a non-linear increasing function. Furthermore, the rotational delay and the data transfer were captured by the remaining variables.

To obtain a sufficiently accurate approximation, $t_{SIO}$ could be skipped in practical cases. Moreover, the calculation of $t_s(x)$ can be facilitated by approximating the course of the seek time using linearization in sections. The specific distances $x$ (for sequences of consecutive requests) could be derived from the file-to-cylinder allocation and the application requests. To enable access time calculation for a particular sequentially allocated file, certain device constants are given for each disk type: $t_{s_{min}}$ (for $x = 1$) and $t_{s_{max}}$ (for $x = N_{dev}$).

[3] We observed for a 64GB flash a dramatic reduction of $\sim$ 70% within 90 days down to $\sim$ 3.5 $/GB, whereas a 750 GB SATA only reached $\sim$ 7%/GB down to $\sim$ 0.15 $/GB.

$t_{s_{avg}}(x)$ assumes random access to blocks uniformly distributed over $N$ consecutive cylinders of the device for which $x = N/3$ is obtained as the average distance. The expected rotational delay $t_r$ was half a revolution ($t_{rev}/2$) of the disk platter. And, finally, the disk transfer rate $F_d$ was dependent on the track capacity $T_{cap}$ in KB (and, in turn, bit density) and revolution speed (e.g., $R$ = 7,200 rpm), which resulted in $t_{tr} = b/(T_{cap} * R) = b/F_d$ sec for a block size of $b$ in KB. An often used approximation formula [9] for $t_d$ was

$$t_d(x,b) \approx t_{s_{avg}}(x) + t_r + b/F_d.$$

Because the avg. travel distance of the disk arm is hard (or even impossible) to determine in a multi-user environment, the avg. seek value—specified as $t_{s_{avg}} = t_s(N_{dev}/3)$ by the manufacturer—was used instead. Hence,

$$t_d(b) \approx t_{s_{avg}} + t_{rev}/2 + b/F_d$$

where besides the application-specific variable $b$ only device characteristics occur.[4]

### 3.2 Device-related Experiments

Today, such simple access scenarios do not exist anymore for disks, because many optimization features were added along the IO path from disk to the application address space in RAM. A typical mechanism at the disk level is the existence of a (RAM) cache—in our case 8 MB—which cannot be controlled by the application. In fact, depending on the current workload, it automatically adjusts the sizes reserved for read and write caching.

Flash memory is now available for more than 20 years. So far, it was not considered as a serious competitor for the disk because of size and cost. However, these aspects have dramatically changed in recent years. Because flash consists of electronic memory, "mechanical movements" do not stretch the device access times, which are therefore much simpler to calculate. Furthermore, a flash is not equipped with a volatile cache, which would speed-up IO while significantly consuming more energy due to the permanent refresh cycles. However, the caching and prefetching options for the OS file system remain the same as for disks. Hence, if we could minimize the file caching effects as far as possible, we should be able to approximate raw-flash timings in a controlled measurement experiment. Because "mechanical time" is not present, when a block is read, flash access time $t_f(b)$ for a block of size $b$ ($b$ given in KB and independent of its location) can be approximated by

$$t_f(b) \approx t_{SIO} + b/F_f$$

where $t_{SIO}$ is roughly the same as for disks. But, it cannot be neglected here because of its relative magnitude compared to the data transfer time $b/F_f$ from the flash device.

To gain a deeper and more complete comparative picture of flash and magnetic disk, we performed dedicated throughput tests. While sequential read/write performance of flash

[4] For example, disk IBM 3330 rotating at 3,600 rpm was characterized by $t_{s_{avg}}$ = 30 ms, $t_{rev}$ = 16.7 ms and track capacity of 13 KB, resulting in $F_d$ = 690 KB/sec, which enabled the calculation of the avg. block access time with sufficient precision, e.g., $\sim$ 43.9 ms for a 4KB block.

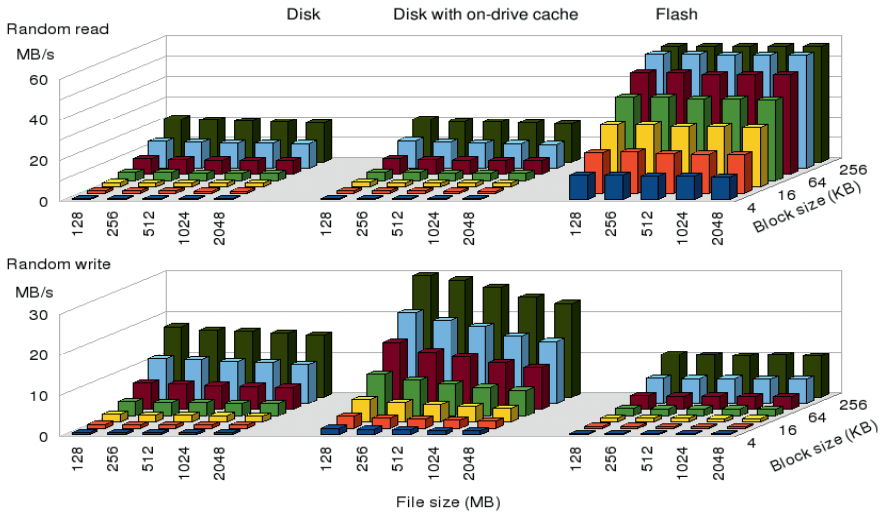



Figure 1: Flash and disk throughput for random reads/writes

and disk is comparable, we focused on random read/write operations at the device level where flash promises the biggest gain for reads, but also the strongest penalty for writes.

Figure 1 summarizes the throughput results (MB/s) of our tests at the LINUX file interface (using iozone[5]), where the OS caching effects were eliminated (to the extent possible) and no buffer pool was explicitly allocated. We varied the file size from 128 MB to 2 GB and the block size as the unit of access from 4 KB to 256 KB. All experiments started with a cold RAM and cold disk cache. In all cases, dividing the file size by the block size delivered a range of integers where *random numbers* for block reads/writes were taken from. The number of accesses were computed by *file size/block size* which guaranteed that (almost) the entire file content was fetched to or written from RAM. Our performance figures are gained for mid-range devices (SSD [21] and SATA 7,200 rpm). Although many design parameters can be optimized [6], the relative IO performance characteristics to be anticipated from both device types remain stable and are also valid for high-end devices. Therefore, the results contained in Figure 1 and the cross-comparisons can be considered indicative for the device-related throughput performance.

Referring to the measured results for flash *random read* in Figure 1, we can derive the values for the system-specific $t_{SIO}$ and the device-specific $F_f$ (65 MB/sec sequential read rate) to obtain an analytic approximation by

$$t_f(b) \approx (230 + b/0.065)\mu sec.$$

Hence, a 4KB block can be read from flash in $< 300$ $\mu$sec whereas read/write access to disk needs $t_d < 10$ ms for a 4KB block. Here this is due to the ratio of file size ($\leq 2$ GB) and disk size (80 GB) where frequent track-to-track seeks accounted only for 2 ms and an avg. latency of 4.2 ms.[6] As expected, a flash write with 4 KB costs $\geq 20$ ms.

[5] http://www.iozone.org

[6] An experiment with 80GB file size would consume $t_{s_{avg}} = 15$ ms for the disk and result in $t_d \approx 20$ ms.

While the overall IO performance of flash and plain disk confirm our statements in Section 2 and correspond to our expectations, closer inspection of Figure 1, however, reveals that, even for the device-related IO performance, cross-comparison and performance assessment in detail is not that easy. For example, the slight decrease of the read throughput for both device types with growing file size is not obvious. This effect is caused by the indirect addressing which provokes some additional block transfers for address information (see Section 4). Furthermore, larger files imply longer arm movements for the disk, however, in our case, to very limited degree.

The access behavior of disks with on-drive cache may substantially deviate from the plain case. Because we started with cold caches, *random read* could not profit. But, *sequential write* or *random write* throughput could be considerably improved, because blocks are buffered in the 8MB cache and asynchronously propagated to disk. Although we did not explicitly measure *sequential* IO performance, it is approached for *random read* ($\sim$ 65 MB/s) and *random write* ($\sim$ 10 MB/s) on flash using large block sizes (e.g., 256 KB). As per device specification, *sequential read/write* disk IO delivers $\sim$ 60 MB/s.

## 4 File-System-Related IO Performance

The file system embodies a major part of the IO path and presents a substantial deviation when application programs such as a DBMS invoke IO requests. Thus, a certain overhead and fuzziness—as far as interpretation of the IO cost is concerned—is induced by the optimization options of the OS kernel (i.e., IO clustering, prefetching, and kernel-related caches). To assess flash and disk access, we need to figure out how the additional abstraction level of the file system affects reachability and appearance of the underlying storage device for the program. Therefore, we try to disclose the cache hierarchy thread.

All our benchmarks and experiments were performed on an AMD Athlon X2 4450e (2.3 GHz, 1MB L2 cache) processor using 1 GB of main memory (RAM) and a separate disk for the operating system. As hard disk, we use a WD800AAJS (Western Digital) having an 8MB Cache, NCQ, and 7,200 rpm. The flash device is a DuraDrive AT series (SuperTalent) having a capacity of 32 GB. The operating system is a minimal Ubuntu 8.04 installation using kernel version 2.6.24 and the Java Virtual Machine is version 1.6.0_06.

Because all layers in the cache hierarchy have well-known interfaces, they may be separately substituted, but normally they are not elided. Figure 2 illustrates the OS- and device-controlled caches. As described in Section 3.2, disks benefit from integrated caches in case of sequential reads/writes or random writes, as opposed to flash devices which do not have such a cache. At the lower OS kernel interface, a *kernel cache* is allocated that allows for buffering entire blocks (as delivered by the device) using an LRU-based propagation strategy. It is automatically resized depending on available main memory. Cache filling is optimized for sequential accesses by prefetching. In the layer above, a virtual file system (VFS) maintains several caches for *inodes*, *directories*, and (application-formatted) *pages*. As these caches are solely controlled by the kernel, flushing them is the only functionality provided for the application. Typically, page cache and kernel cache hold actual, but not explicitly and currently requested data in RAM. In contrast, the *inode cache* and *directory cache* speed-up file resolution, block addressing, and navigation.

### 4.1 Caching Hierarchy

At the OS level, room for file caching may be reserved in such a way that all RAM currently not needed by the OS and the applications is used for caching block reads and writes. The file system may use this cache to prefetch blocks from disk in a way not visible and controllable by the applications. This kind of prefetching is often heuristically performed depending on the locality or sequentiality of current block accesses to disk. For example, Linux offers the following options:

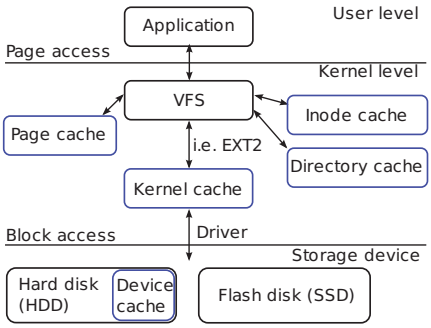


Figure 2: Caching hierarchy

- The first file access conservatively decides on prefetching. Only when the *first block* is read, sequential access is anticipated and some minimal prefetching takes place.
- Synchronous prefetching can be enabled to lower seek costs for larger reads.
- Read access pattern recognition may dynamically trigger synchronous and/or asynchronous prefetching.

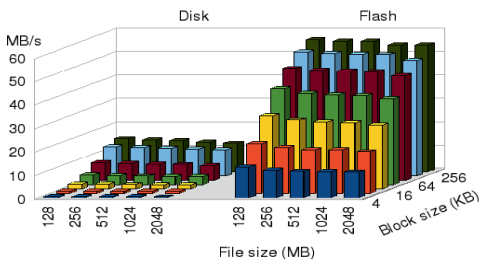
To enable greater flexibility, storage access optimization is further separated into raw-block caching and file-structure caching (for inodes, indirect block references, etc.). Moreover, some access options (direct, sync) are provided at the file system interface. However, not all OS versions (in our case Linux) observe such application directives, e.g., even if the raw-disk option is enforced, the OS does not respond to it. As a result, great indeterminism may be caused for all kinds of disk IO timings in real application environments.

### 4.2 File Structure Mapping

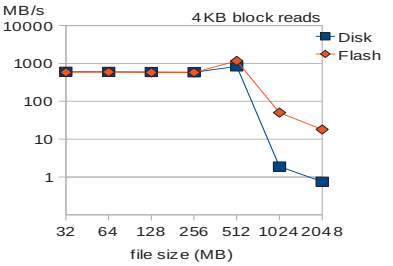
In Linux systems, the dominating file systems are EXT2 and its journal-enabled version EXT3, respectively. EXT2/3 divides the disk partitions in equal-length groups containing descriptors and bitmaps for inodes and block allocation. An inode is a special data structure describing a file's size, length, block allocation, etc. Because an inode only references 12 data blocks directly, the remaining data blocks for files larger than 12 blocks need to be referenced indirectly. This indirection of one, two, or even three levels can lead to an overhead of one-tenth of a percent for large files. Certain access patterns may suffer from such "long" block reads when several indirect block references have to be resolved before.

### 4.3 File/Device Caching Effects

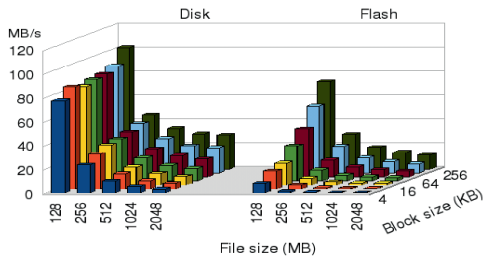
Prior work in [3] has shown that caching effects may closely interfere with the kernel's prefetching and replacement strategy. In all our experiments, we used the default LRU replacement and the default prefetching of the Linux kernel, because we want to compare



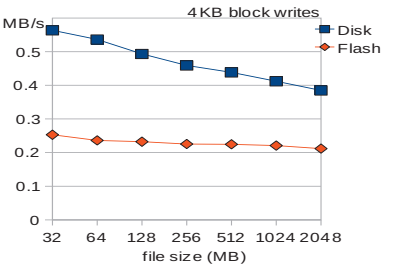

(a) Random read, cold buffer




(b) Hot buffer effect




(c) Random write, async




(d) Sync effect

Figure 3: File system benchmarks and their effects: cold vs. hot buffers for random reads (a vs. b) and unforced vs. forced random writes (c vs. d)

their effects for the different types of physical devices. Furthermore, we constrained some of our experiments to the default block size for the file system of 4 KB to limit the result dimension to a reasonable degree. Of course, when perfect knowledge of future access is available, the file-system block size can be adjusted to the minimal applications' page size, thereby gaining the most from caching and prefetching. However, real systems are faced with varying access patterns and the 4KB block size seems to be the lowest common unit.

Comprehensive read and write benchmarks, as shown in Figure 3, have revealed that the OS' page cache and the synchronous-write option may seriously affect the IO behavior of the underlying devices in different ways. Note that we illustrate our experiments in Figure 3a and Figure 3c only for file sizes of $\geq$128 MB, because the results for smaller files were dominated by caching effects that even cold-buffer experiments could not be explained in a plausible way. Figure 3a reveals the slight degradation as compared to that of Figure 1 due to the additional algorithmic overhead of the kernel's optimization efforts. In contrast, Figure 3c clearly shows the additional caching effects, when data is written asynchronously (default case). For smaller amounts of data (less than main memory available), most of the data is only written to the cache and not propagated to the physical device, leading to high throughput rates. Having more data to write than the available memory can cache, the asynchronous flush daemon (pdflush) is noticeably slowing down the throughput into the raw device's performance region (see Figure 1).

Another interesting effect appears when the IO behavior of hot and cold buffers is compared. As real systems do rarely restart, the buffers are more or less hot during runtime. Figure 3b shows the effect when operating on 4KB blocks: once the dataset does not

fit into main memory, the throughput drastically drops to the cold buffer rates. It further reveals that the throughput for small datasets is completely decoupled from the underlying storage, because the disk and flash throughput rates are equal up to a file size of 512 MB.

Because our primary application is database-based, the distinction between synchronous and asynchronous write is essential. The default file system behavior, shown in Figure 3c, does not enforce immediate write propagation at all. When using the *sync flag* or calling the system function *sync*—mandatory for some specific writes within transactions to guarantee the ACID paradigm—substantial throughput reduction has to be taken into account (compare Figure 3d and Figure 3c for 4KB block writes). Figure 3d shows for 4KB block writes that the throughput rates are independent of the file size. The slight decrease is explained by the file structure overhead discussed in Section 4.2 and the additional seek overhead for large files in case of a disk.

The simple and often constant throughput rates of the raw devices are markedly affected by the file system and the kernel's caching efforts. Thus, an application (i.e., our database) needs to be aware of that when using access paths or building cost models.

### 4.4 Non-influenceable Caches and Other Magics

Keeping instructions and data close to the CPU(s) in the processor caches can significantly change the anticipated IO behavior. Whenever physical access is not needed at all, certain OS-internal optimizations are effective to improve (or at least to influence) the read/write throughput. In some situations, where definitely no physical IO occurred anymore, read throughput reached an extraordinary level. In Figure 3b, it is up to three orders of magnitude ($\sim$ 600 MB/s) higher for file sizes less than the available main memory. A peak (of $\geq$ 1000 MB/s) occurs in this measurement series when processing a file of 512 MB, before the throughput dramatically decreases. For this "magic" behavior which is independent of the flash/disk distinction, we found the following explanations:

- The CPU has small, but really fast processor caches (so-called L1 and L2). They typically range from KBs to several MBs, enough to keep frequently referenced data (sometimes entire files) as close to the CPU as possible avoiding MMU and IO access. Unfortunately, parallel processes on multiple cores or operating on GB-sized data volumes cannot exploit it.
- In the identified peak situation, the kernel cache does not need to be accessed anymore, because the requested pages reside in the page cache and the processor caches. In particular, the processor caches have reached a state where at least the addressing information is steadily available, which may explain why the throughput is almost doubling, although already at a high level.
- Cached entries (pages, blocks) have a state (e.g., clean, locked, dirty) and can be accessed in isolation or shared among several processes. Different states involve different access control operations leading to more or less computational overhead.

There are further "peculiar" situations which affect pure throughput tests. However, dealing with large files or data volumes (i.e., much larger than the available main memory—typically several GB or TB nowadays), all of them are nearly smoothed to negligibility.

## 5 IO-Relevant DBMS Processing Concepts

The IO characteristics of flash indicate that disk-based DB performance cannot be obtained in an easy way, not to mention by simple replacement. Substantial bandwidth gains are possible for random reads, while random writes are problematic and have to be algorithmically addressed at the DBMS side. If frequent writes are necessary, the adjustment of (logical) DB pages to the device's block sizes may relieve the performance bottleneck. In mixed read/write workloads, write caching will play a major role to realize write avoidance. Further, the potential of sequential writes should be consequently exploited. Hence, DBMS processing should be adapted such that the drawbacks are avoided and, at the same time, the strengths of flash are used to full capacity.

### 5.1 Prime Optimization Principles

In several respects, performance and energy efficiency in DBMSs are not conflicting design goals and should be jointly approached, where appropriate. It is always a good idea to optimize at a logical or algorithmic level and not to purely rely on device-based improvements. When using external storage, the most important design principles are:

- Design compact and fine-grained storage structures to minimize IO for allocating, fetching, and rewriting DB objects.
- Use processing concepts which maximize memory-resident operations and minimize random read/write access to external storage—in particular, avoid random writes to flash to the extent possible.

We discuss these principles for native XML DBMSs and outline various XML storage structures, which help to reveal critical aspects of DBMS-related IO performance. In general, saving IO is the major key to performance improvements and, in turn, energy efficiency in DBMSs. Thus, both goals are approached by using *compact* storage formats, because *plain* XML documents contain substantial redundancy in the structure part, i.e., the inner nodes and paths of the document tree (see Figure 4a)[7]. However, the tree structure should be preserved to support fine-grained document processing. Despite IO minimization, storage mapping should be flexible enough to enable dynamic modifications.

### 5.2 Structure Encodings

Natively storing XML documents requires identification of the resulting tree nodes, i.e., the assignment of node labels. Because performance, flexibility, and effectivity of many XDBMS-internal operations critically depend on them, the appropriate label scheme design is a key issue for the system implementation. Early implementations, primarily optimizing static XML documents, mostly voted for *range-based labeling* schemes. They

[7] The Computer Science Index *dblp* currently contains almost $10^6$ instances for path class */bib/paper/author* and even more for element name *author*.

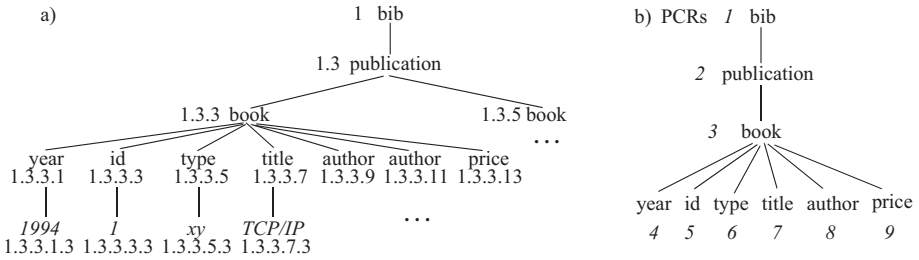



Figure 4: Labeled document (DOM tree) and its path synopsis

provide satisfactory query evaluation support: all axis relationships can be directly decided by comparing node labels. However, dynamic insertion of subtrees would cause a relabeling of the document (or a fraction of it) triggering a bulk IO operation. Moreover, they fail to provide the ancestor IDs when the ID of the context node is known, which is most important for setting intention locks when fine-grained multi-granularity locking is applied [12]. But, *prefix-based labeling*, derived from the concept of DeweyIDs, remains stable also in dynamic documents and enables all operations mentioned without document access—only by checking node labels [11]. Such labels as shown in Figure 4a support all optimization principles; we denote them SPLIDs (Stable Path Labeling IDentifiers).

A *path synopsis* represents equal path instances, i.e., those having the same sequence of element and/or attribute nodes, in an XML document only once as a so-called path class. PCRs (path class references) are added to the nodes of the path synopsis to identify the path classes, as illustrated in Figure 4b for a cut-out of the *dblp* document [17]. Typically, a small memory-resident data structure is sufficient to it during XML processing.

### 5.3 Structure Virtualization and Storage Mappings

Because of the typically huge repetition of element and attribute names, getting rid of the structure part in a lossless way helps to drastically save storage space and, in turn, document IO. As a consequence, log space and log IO may be greatly reduced, too. The combined use of SPLIDs as node labels and a path synopsis makes it possible to virtualize the entire structure part and to reconstruct it or selected paths completely on demand. When, e.g., SPLID=1.3.3.5.3 together with PCR=7 is delivered as a reference (e.g., from an index) for value *TCP/IP*, the entire path instance together with the individual labels of the ancestor nodes can be built: *bib/publication/book/title*.

B*-trees—made up by the document index and document container—and SPLIDs are the most valuable features of physical XML representation. B*-trees enable logarithmic access time under arbitrary scalability and their split mechanism takes care of storage management and dynamic reorganization. As illustrated in Figure 5, we provide an implementation based on B*-trees which cares about structural balancing and which maintains the nodes stored in variable-length format (SPLID+element/attribute (dark&white boxes) or SPLID+value (dark&grey boxes)) in document order. In the following, this XML storage mapping is denoted as *full* document representation.

As signified in Figure 4a, a SPLID is composed of so-called *divisions* and its length is dependent on the position of the corresponding node in the document. The value of a division identifying a node at the given level increases with the breadth of the document tree and, because it is inherited to the SPLIDs of all descendants, it affects the related descendent SPLIDs at all lower document levels. Although we use efficient Huffman encodings for the division values [11], they may reach substantial lengths in large documents. Note, the TPoX documents [19] consist of several million subtrees rooted at the second level in our 10GB experiments. Because the nodes in all storage mappings occur in document order (see Figure 5), their labels lend itself to prefix compression. Therefore, quite some storage space per document can be saved on external devices. On each page of the document container, only the first SPLID is completely stored. By storing the difference to the preceding SPLID and encoding the length of the SPLID prefix (in terms of divisions), which can be derived from the sequence of preceding node labels in a page, we obtain an effective compression mechanism (e.g., instead of storing 1.3.3.7.3, the encoded difference to the previous SPLID is enough). As a result, we gain the *prefix-compressed (pc)* storage mapping where in case of dense labels the avg. size needed for SPLID storage is reduced to $\sim$20–30 % [10].

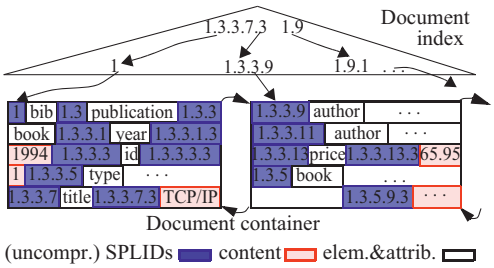



Figure 5: Storage mapping to a B*-tree

A virtualized structure only stores content (leaf) nodes in the container pages where each node is made up of a prefix-compressed SPLID, the PCR, and the text value. Hence, when storing or reconstructing the entire document or when accessing an individual root-to-leaf path, even more IO is saved as compared to the *pc* storage representation. We denote this kind of mapping as *elementless* XML storage mapping (*eless* for short).

In the following, we use all three storage mappings of XML documents in comparative measurements.[8] We applied the TPoX tools [19] to generate the various documents in the external text format, called the *plain* format, from which the differing storage mappings were created. To assess their storage effectivity, we cannot refer to absolute numbers, because the sizes of our sample documents vary from 10 MB to 10 GB. Therefore, we have normalized the space figures w.r.t. to *plain* (100%). Hence, the storage footprints of *full*, *pc*, and *eless* obtained quite some reduction and reached $\sim$95 %, $\sim$70 %, and $\sim$65 %, respectively; these figures directly characterize the IO saving when storing or reconstructing the entire document.

To later enable interpretion of query evaluation, we summarize in Table 1 the storage space needed for the TPoX-generated *plain* documents[9] when mapped to the *full* storage representation. For the corresponding footprints of *pc* and *eless*, we obtained roughly 75% and 70% of the pages numbers given. In particular, the growing sets of index pages and, in turn, the increasing heights of the document index $h_{docind}$, which together imply less buffer locality in our benchmarks, will affect the scalability results.

[8] Here, we do not consider content compression. It would further decrease IO overhead, but increase CPU time for additional compression/decompression tasks.

[9] We adjusted the TPoX generation parameters to roughly obtain *plain* documents of the given sizes.

Table 1: Footprints for *full* storage mapping of the TPoX documents.

| Documents | Number of index / container pages (4 KB) [index height] | | | |
|---|---|---|---|---|
| | 10 MB | 100 MB | 1 GB | 10 GB |
| customer | 7 / 967 [2] | 78 / 9,317 [2] | 964 / 121,026 [3] | 10,794 / 1,219,099 [3] |
| order | 7 / 871 [2] | 87 / 10,650 [2] | 728 / 101,641 [3] | 9,501 / 1,118,463 [3] |
| security | 7 / 982 [2] | 85 / 10,493 [2] | 220 / 40,385 [3] | 3,382 / 407,604 [3] |

# 6 DBMS-Related IO Performance

The flash characteristics highlighted in Section 2 promise quite some gain in performance and energy saving when considered in isolation. The interesting question is whether and to what degree their potential can be exploited in an XDBMS context? Furthermore, how can the optimization principles (see Section 5.1) assist our objectives?

A recent contribution [2] made us aware that "it is amazingly easy to get meaningless results when measuring a flash device because of the non-uniform nature of writes" even at the device level. For the interpretation of DBMS-related measurements, we have to cope with indeterminism provoked by uncontrollable OS caching and influence of DBMS buffer locality (which together fooled our initial complex measurements for quite some time). Therefore, we designed simple experiments which enabled meaningful result interpretation in the first place. Of course, they should reveal some (not all) important performance differences of flash- vs. disk-based DB systems.

To get an initial answer, we used *XTC* and implemented—by varying our options concerning external storage (disk and flash) and document representation (unoptimized (*full*), prefix-compressed (*pc*), and elementless, i.e., virtualized (*eless*))—six *configurations* of it. Furthermore, we designed a simple benchmark consisting of reader and writer transactions and ran it in single-user mode. Because we are solely interested in the relative performance behavior of the workload on the different system configurations, the processing details for the documents and transactions are only roughly sketched.

## 6.1 Workload Description

The performance analysis at the DB level is based on the well-known TPoX [22] benchmark to achieve a mixture of random/sequential access and a varying data distribution. Because most of the TPoX queries are quite complex and their detailed analysis would go beyond the scope of this paper, we chose the following queries supporting our objectives:

*RQ1* report an account summary
*RQ2* resolve a customer
*RQ3* return a security price value
*RQ4* return an order
*RQ5* search for a specific security
*WQ1* create a new account
*WQ2* place an order
*WQ3* update a price value

The *RQ** queries are read-only and the *WQ** queries include a share of updates or inserts. To reflect the indeterminism of real environments, all queries are supplied with

random parameters. By changing the weights of the queries, the share of the write load for the benchmarks is scaled from 10 % to 30 %.

Each benchmark run accesses the three XML documents *customer*, *order*, and *security* with almost uniform frequency. In all cases, a query selects some random target in a document with a given SPLID delivered from an element index—the only additional index used for these benchmarks. After having traversed the document index and located its context node, each query locally navigates in the document (see Figure 5) thereby accessing a number of records. Because the subtrees are small, these operations remain local (restricted to one or two container pages). Hence, the set of queries—each executed as a transaction—causes fine-grained and randomly distributed IOs on the XML database.

A major aspect of our analysis includes scalability properties. Each setup consisted of separate databases for the configurations *disk/full*, *disk/pc*, *disk/eless*, *flash/full*, *flash/pc*, and *flash/eless*. Choosing setups with 10 MB to 10 GB for the documents (in *plain* format), the system behavior is explored across three orders of magnitude. Therefore, the document footprint on a storage device (especially affecting the seek times on disk) and, in turn, the height of the document index considerably affect the length of a traversal path.

To gain some insight into the flash-disk dichotomy, we modeled two different processing situations where IO activities resp. complex computations in memory dominate the benchmark. Thus, we ran each benchmark in two versions. *IO-intensive* means that query evaluation query is the main transactional activity and that each transaction is started as soon as the preceding transaction committed. In contrast, in the *CPU-intensive* version we prolonged each transaction by 50 ms to account for complex main-memory computations.

### 6.2 Performance Measurements

There are literally infinite combinations of benchmarks, document configurations and other parameters. Here, we cannot provide area-wide and exhaustive empirical performance measurements. Instead, we intend to present some indicative results which focus on the cross-comparison of the differing configurations. For the different storage mappings and the considered devices, our measurement settings enable to highlight the influence of the OS- and device-related caches, to reveal some scalability aspects, and to figure out the flash benefits under different workload types.

The benchmarked workloads (consisting of 1K and 10K transactions[10]) were randomly assembled from the set of pre-defined queries and executed on the four DB sizes (setups). Within each setup consisting of 6 configurations, exactly the same queries were chosen, such that the differences in the IO behavior are only caused by the different storage mappings and the device type. To account for DB growth, only the query parameters were adjusted in the different setups to address document paths randomly spread over the entire document. While the IO activities on container pages remained rather unaffected by our scaling measures, the traversals through the document indexes were not. See Table 1 for the page numbers of the indexes, which determine their heights $h_{docind}$.

[10]Fewer transactions in a benchmark lead to lower *tps* rates, because they can—only to a lesser extent—take advantage of the increasing caching effects caused by index locality. But, on the other hand, their results are affected by singular and initialization effects. Therefore, such benchmark results are not reported here.

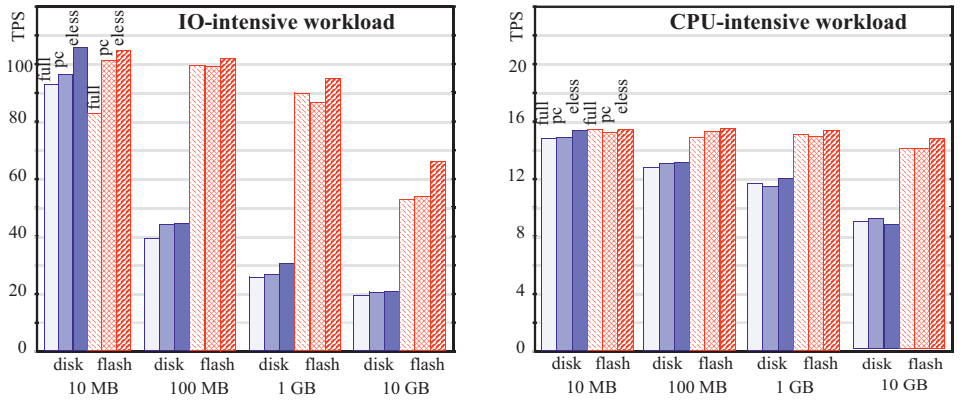



Figure 6: Reader transaction throughput (1,000 TAs)

As runtime parameters, we always used 4KB database pages, 4096 4KB buffer frames in *XTC*, and cold buffers.[11] Hence, the results of all read-only benchmarks were gained with the same workload (only adjusted to the DB sizes) in the same system environment.

Figure 6 summarizes the results of our empirical experiments for read-only benchmarks running 1K transactions, which behaved in the anticipated way, at least in principle. They confirm that there is a substantial rise in transaction performance (*tps* from disk to flash configurations (configs). The results of the disk/flash configs@10MB should be excluded from this comparison, because their differences are strongly overloaded with caching effects. In this case (see Table 1), all index pages are in the XTC buffer after $\sim 20$ queries, whereas obviously less index locality occurs for larger DB sizes and, for configs@10GB, index locality primarily occurs at higher index levels, because not all index pages reach the buffer during the benchmark runs. On the other hand, longer search paths have to be traversed, because $h_{docind}$ grows with the document size, and longer seeks are needed, because the storage footprint covers a larger disk area.

To illustrate the growing IO overhead, e.g. for *full* storage mapping, the 1K-transaction benchmark needed $\sim 1200$, $\sim 2400$, $\sim 3430$, and $\sim 4550$ page fetches for the 10MB, 100MB, 1GB, and 10GB sizes, respectively. Because the same set of transactions was executed and $\geq (h_{docind} + 1)$ logical page references were required in each of them, these numbers of XTC buffer faults reflect the locality achieved. As particularly visible for the IO-intensive workload at the *tps* level, scalability has suffered from the considerably growing sets of index pages and, in turn, the increasing index heights ($2 \leq h_{docind} \leq 3$ in Table 1). However, it affects to a much larger degree disk than flash configurations.

In contrast to the benchmarks with dominating IO-related activities, the CPU-intensive workloads only weakly exhibit the overall characteristics, because the IO influence with only $\sim 15\%$ of total time is almost leveled out. Of course, if hardly any IO is executed in a benchmark, flash cannot play its advantage off against disk.

[11] The effects of the kernel cache on transaction throughput could not be controlled. We observed that, when a 4KB page was requested, frequently a 64KB block was prefetched into the kernel cache.

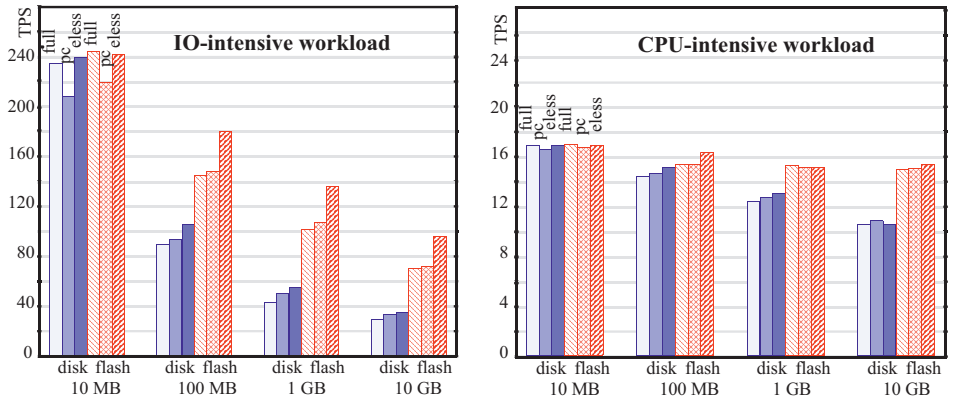


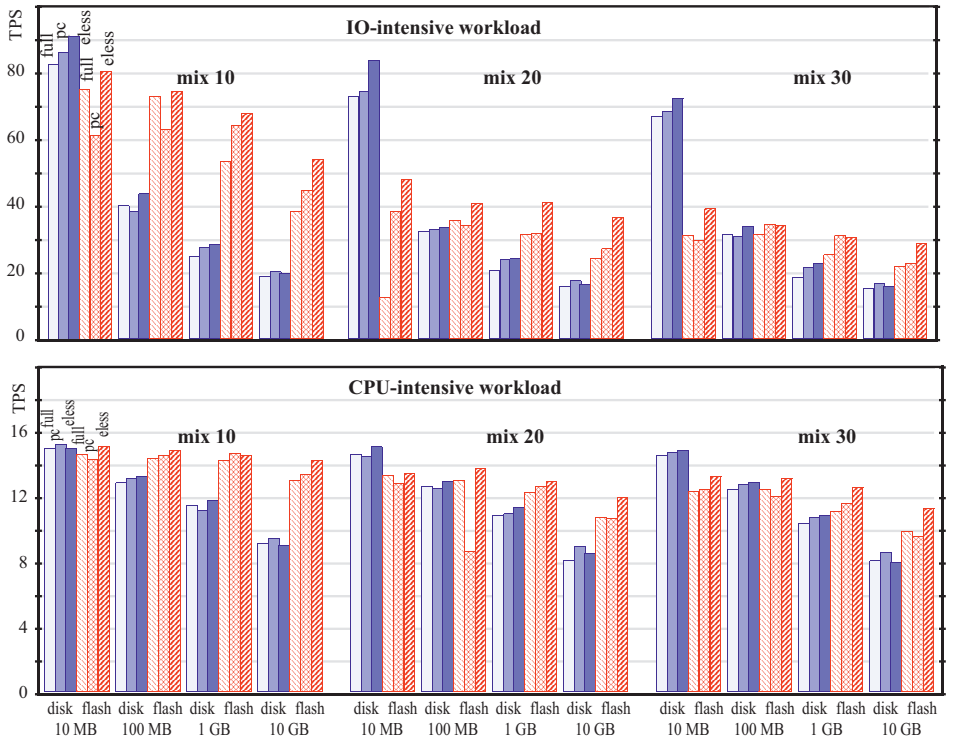

Figure 7: Reader transaction throughput (10,000 TAs)

Figure 7 summarizes the results for the benchmarks with 10K transactions, where the caching effects clearly increase on all setups and the IO-intensive *tps* rates more than double for the 10MB DB on flash as well on disk because of substantial locality on index and container pages. But also the larger DB sizes up to the configs@10GB are strongly influenced by index locality. While in the 1K-transaction, IO-intensive case the flash *tps* rates for configs@100MB and larger are $\sim 150\,\% - \sim 200\,\%$ higher than for disks, they are reduced in the respective 10K cases down to $\sim 75\,\% - \sim 150\,\%$—a definite example that caching may level out the flash performance superiority.

The result representation in Figure 6 and Figure 7 allows for some differential interpretations. What are the gross effects of *eless* and *pc* as compared to *full*? The *eless* mapping having the smallest storage footprint (only $\sim$ 70% of *full*) proved its superiority in terms of *tps* throughput among the disk cases and among the flash cases. The picture is not so clear with *pc* despite a footprint of $\sim$ 75% of *full*. This may be caused by the decompression efforts needed to reconstruct the SPLIDs in the structure part. But, in general, the built-in IO saving of a storage mapping pays off and is, in particular, visible in Figure 7 for the *eless* flash-configs@100MB and larger. While disk seeks take longer, flash access times are independent of DB sizes.

To gain some insight into application-level effects of the flash's "bad" write performance, we extended our benchmark with writer transactions. Figure 8 illustrates the results obtained for an increasing write share—again separated for the IO-intensive and CPU-intensive cases. As expected, the flash write performance continuously decreases with a growing share of writer transactions. In our experiments, the *tps* ratio steadily shrinks with an increasing writer share (from *10% write* to *30% write*). Comparing them to the read-only 1K transaction, IO-intensive case, the flash *tps* rates are clearly lower and reach $\sim 80\,\% - \sim 90\,\%$ for *mix 10* and only $\sim 30\,\% - \sim 35\,\%$ for *mix 30*. The performance gap to the disk closes with a growing writer share as well.

Our experiments definitely reveal the importance of the *eless* storage mapping—even more for write transactions. It enabled the highest *tps* rates in more or less all flash configurations and DB sizes.




Storage mappings: full, prefix-compressed (pc), virtualized (eless) structure

Figure 8: Transaction throughput for mixed workloads (1,000 TAs)

### 6.3 Energy Efficiency

So far, we have shown that at least IO-intensive (read-only) DB applications may substantially improve their transaction throughput when flash devices are used as DB store. To obtain some hints concerning the energy efficiency in our flash/disk comparison, we have started some initial measurements to figure out the overall energy consumption of our computer system (including processor, memory, cooling, and separate OS disk) when running with a flash or a disk as DB store. Table 2 summarizes the results of controlled 60-minute runs in *idle* mode and under the CPU-intensive and IO-intensive benchmarks on the 10GB DB (read-only, *pc* storage mapping).

The relative flash/disk power consumption in *idle* mode yields $P_f = \rho * P_d$ (with $\rho \sim 0.96$ in our case)[12]. To derive a formula for the same DB work, the runtimes for flash and disk can be set to $t_f = (tps_d/tps_f) * t_d$. Hence, we can compute the energy $W$ needed for the same DB load and obtain $W_f = P_f * t_f = \rho * (tps_d/tps_f) * t_d * W_d$. If we define the flash's energy efficiency (w.r.t. disk) as $E_f = W_d/W_f$, we are able to pinpoint the essence of our

[12] Although the power continuously varies between $P_{min}$ and $P_{max}$, we simplify our considerations by assuming constant consumption over time.

Table 2: Energy consumption of a computer system with flash and disk database.

| Device | Workload | Throughput tps | Power (Watt) min | Power (Watt) max | Energy KWh | No. of TAs per Joule |
|---|---|---|---|---|---|---|
| Flash | idle | 0 | 52.1 | 55.0 | 0.053 | 0 |
| | CPU-intensive | 12,530 | 49.5 | 84.6 | 0.071 | 0.176 |
| | IO-intensive | 72,240 | 48.8 | 92.8 | 0.085 | 0.850 |
| Disk | idle | 0 | 53.9 | 65.0 | 0.055 | 0 |
| | CPU-intensive | 10,470 | 53.7 | 87.9 | 0.063 | 0.166 |
| | IO-intensive | 22.810 | 53.7 | 94.7 | 0.062 | 0.368 |

findings summarized in Table 2. For the IO-intensive case, we yield a surprisingly large factor $E_f \leq 3.3$, whereas with $E_f > 1$ (in our case $E_f \simeq 1.25$) only moderate improvements can be gained for CPU-intensive cases.

In general, there are several possibilities to specify "efficiency", and a well-suited definition seems the (application-specific) *#TAs/Joule* ratio of a system (see Table 2). As shown in this paper, the plain IO characteristics cannot serve to choose a device for an application-level use case. Hence, the *#TAs/Joule* ratio should be addressed when thinking about efficiency. Here, flash devices have selectively shown their superiority over traditional disk storage.

## 7 Conclusions

The DB community will be confronted in the near future with the use of flash storage not only in embedded devices, e.g., PDAs, because it is energy efficient, small, light-weight, noiseless, and shock resistant. Expected breakthroughs in bandwidth (IO/s), energy saving, reliability, and volumetric capacity, will certainly shift this technology into high-end DB server environments. If data volumes in enterprises further grow (to the PB range), its zero-energy needs—when idle or turned off—will become a strong sales argument.

From this perspective, we started to explore the integration of flash storage into databases. We delivered a kind of "existence proof" for simple use cases where "flash beats the disk". There are definitely (read-only) scenarios where substantial performance gains and, at the same time, energy savings can be achieved. For write scenarios—as expected—, algorithmic adjustments are necessary in various DBMS components, in particular, in the areas of cache management and logging. But we could show for the flash that *performance can be kept or even improved while energy is saved.*

On the other hand, we are just at the beginning of flash-aware improvements in DBMSs. But, we believe that it is not sufficient to just develop new algorithms (e.g., for sort or join [20]) focusing on the device characteristics alone. While probably no general flash superiority is possible, there exist specific application areas which could profit from its use. Therefore, our future research will attempt to disclose these potentials and to bring flash technology to the core of database management.

## References


[1] Ban, A.: Wear leveling of static areas in flash memory. US patent, (6732221); Assigned to M-Systems (2004).

[2] Bouganim, L., Jonsson, B., and Bonnet, P.: uFLIP: Understanding Flash IO Patterns. Proc. Int. Conf. on Innovative Data Systems Research (CIDR), Asilomar 2009.

[3] Butt, A. R., Gniady, C., and Hu, Y.C.: The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms. Proc. SIGMETRICS Conf., 157–168 (2005).

[4] Graefe, G.: Database Servers Tailored to Improve Energy Efficiency. Proc. Int. Workshop on Software Engineering for Tailor-made Data Management, 24–28, Nantes, France (2008).

[5] Graefe, G.: The Five-minute Rule: 20 Years Later and How Flash Memory Changes the Rules. Proc. Third Int. Workshop on Data Management on New Hardware (DaMoN), Beijing, China (2007).

[6] Gray, J., and Fitzgerald, B.: FLASH Disk Opportunity for Server-Applications. http://research.microsoft.com/~Gray/papers/FlashDiskPublic.doc (Jan. 2007).

[7] Gray, J.: Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King, powerpoint talk, Microsoft, Dec. 2006.

[8] Gray, J. and Reuter, A.: Transaction Processing: Concepts and Techniques. M. Kaufmann Publisher (1993).

[9] Härder, T. and Rahm, E.: Database Systems – Concepts and Implementation Techniques (in German). 2nd edition, Springer (2001).

[10] Härder, T., Mathis, C., and Schmidt, K. Comparison of Complete and Elementless Native Storage of XML Documents. Proc. 11th Int. Database Engineering & Applications Symposium (IDEAS), 102–113, Banff, Canada, (2007).

[11] Härder, T., Haustein, M. P., Mathis, C., and Wagner, M.: Node Labeling Schemes for Dynamic XML Documents Reconsidered. Data&Knowledge Engineering 60:1, 126–149, Elsevier (2007).

[12] Haustein, M. P., and Härder, T. An Efficient Infrastructure for Native Transactional XML Processing. Data&Knowledge Engineering 61:3, 500–523, Elsevier (2007).

[13] Hwang, C.: Chip memory to keep doubling annually for 10 year. http://www.korea.net/News/news/LangView.asp?serial_no=20060526020&lang_no=5&part=106 (2006).

[14] Kerekes, Z. (editor of STORAGEsearch.com): SSD Myths and Legends – write endurance. http://www.storagesearch.com/ssdmyths-endurance.html (2007).

[15] Leventhal, A: Flash Storage Today. ACM Queue 6:4, 24–30 (2008).

[16] MFT: Managed Flash Technology. http://www.easyco.com/mft/index.htm (2008).

[17] Miklau, G.: XML Data Repository. http://www.cs.washington.edu/research/xmldatasets.

[18] Nath, S. and Kansal, A.: FlashDB: Dynamic Self-tuning Database for NAND Flash. ftp://ftp.research.microsoft.com/pub/tr/TR-2006-168.pdf (2007).

[19] Nicola, M., Kogan, I., and Schiefer, B.: An XML transaction processing benchmark. Proc. ACM SIGMOD Conf., 937–948, Beijing, China (2007).

[20] Shah, M. A., Harizopoulos, S., Wiener, J. L., and Graefe, G.: Fast Scans and Joins using Flash Drives. Proc. 4th Int. Workshop on Data Management on New Hardware(DaMoN), 17–24, Vancouver, Canada (2008).

[21] SSD White Paper. http://www.supertalent.com/datasheets/SSD_WHITEPAPER.pdf

[22] XML Database Benchmark: Transaction Processing over XML (TPoX). http://tpox.sourceforge.net/ (January 2007).

# High-dimensional indexing for multimedia features

Ira Assent* Stephan Günnemann+ Hardy Kremer+ Thomas Seidl+
*Department of Computer Science
Aalborg University, Denmark
ira@cs.aau.dk
+Data Mining and Data Exploration Group
RWTH Aachen University, Germany
{guennemann,kremer,seidl}@cs.rwth-aachen.de

**Abstract:** Efficient content-based similarity search in large multimedia databases requires efficient query processing algorithms for many practical applications. Especially in high-dimensional spaces, the huge number of features is a challenge to existing indexing structures. Due to increasing overlap with growing dimensionality, they eventually fail to deliver runtime improvements.

In this work, we propose an overlap-free approach to indexing to overcome these problems and allow efficient query processing even on high-dimensional feature vectors. Our method is inspired by separator splits e.g. in B-trees for one-dimensional data or for sequence data. We transform feature vectors such that overlap-free splits are ensured. Our algorithm then queries the database with substantially reduced number of disk accesses, while ensuring correctness and completeness of the result.

Our experiments on several real world multimedia databases demonstrate that we build compact and overlap-free directory information in our index that avoids large percentages of disk accesses, thus outperforming existing multidimensional indexing structures.

## 1 Introduction

There is tremendous growth in multimedia databases in application domains such as medicine, engineering, biology and entertainment. New technologies such as computer tomography or digital photography produce increasingly large volumes of data. Typical access to multimedia data is required in form of content-based similarity search, e.g. to support medical diagnosis by automatically retrieving similar magnetic resonance images. Similarity models are usually defined via distance functions evaluated with respect to content related features like color distribution or shape histograms.

Due to the massive amount of multimedia data, efficient algorithms for content-based similarity search are of foremost importance. Efficient retrieval is especially challenging for high resolution of features, i.e. high-dimensional feature vectors. In high-dimensional feature spaces, not only the one-to-one comparison of multimedia features is more costly. Most importantly yet, multidimensional indexing structures that are deployed to improve runtimes, degenerate. Degeneration is due to the fact that with increasing feature dimensionality, descriptor regions (e.g. minimum bounding rectangles in R-trees) overlap to a

growing extent. Consequently, most queries will require access to a large part of the directory and random access at the leaf level. Thus, eventually index-based search algorithms may perform even worse than a sequential scan of the entire database, meaning that they fail at their very goal of efficiency improvement.

In this work, we propose an indexing approach that avoids overlap even for high-dimensional feature histograms, thus allowing efficient runtimes for content-based similarity search. Our idea is inspired by the observation that for one-dimensional or sequential data types, overlap-free indexing approaches exist. Using the inherent order of dimensions, separators are directory entries that split the data into smaller and larger values without any overlap. We propose to derive a meaningful ordering of dimensions and discretization of values to generate an overlap-free separator split, while maintaining minimum bounding rectangles to enhance the pruning power.

In our method, query processing is efficiently possibly, as the number of paths that have to be followed for any given query is greatly decreased, and consequently, the number of disk accesses is substantially reduced. We provide a multistep filter-and-refine algorithm to ensure that discretization neither produces false dismissals nor introduces false hits.

We demonstrate in thorough experiments on several real world databases that our separator-based approach builds very compact index directories, and that the number of disk accesses is substantially reduced in comparison with competing approaches.

Summarizing, advantages of our method include:

- Compact and overlap-free indexing for multimedia databases
- Scalability to high-dimensional and massive data
- Efficient, complete, and correct query processing algorithm
- Substantially reduced number of disk accesses

This paper is structured as follows: we review related work on indexing of multimedia databases in Section 2 and study the challenges involved in overlap-free indexing for content-based similarity search in Section 3. Section 4 details our method, with Section 4.1 describing the indexing concept and its properties, whereas Section 4.2 specifies the index structure followed by the query processing algorithm in Section 4.3. Our experiments in Section 5 demonstrate great reduction in disk accesses for query processing, before we conclude our work in Section 6.

## 2 Related work

In the literature, different indexing structures for efficient query processing have been proposed [Sam06]. The general idea is to maintain compact directory entries that hierarchically narrow down the search to relevant parts of the database, thus decreasing the number of disk accesses and consequently the overall search time as well.

The original B-tree was developed for indexing of one-dimensional data or keys [BM70, BM72, BU77]. Directory entries are separators between smaller and larger values with respect to the order in this dimension. Moreover, as nodes reflect disk page size, short separators usually provide compact directory entries with large fanout, and thus small trees. During search, point queries will only require access to one path along the tree, and typically range queries read only small parts of the database as well. In String B-trees or the TS-tree, this idea was also used for sequential data where separators between sequences can be computed as well [FG99, AKAS08].

In multidimensional or spatial data, the R-tree family extends the idea of page-based balanced trees [Gut84, BKSS90, MNPT06]. Directory entries are minimum bounding rectangles of the respective subtrees. For low to medium dimensional data, these indexing structures provide substantial efficiency gains. With increasing dimensionality, however, overlap of minimum bounding rectangles eventually leads to degeneration. Typical queries overlap with larger parts of the directory and hardly any data can be pruned from the search. Consequently, indexing fails to deliver efficiency gains in high dimensional data, even compared with brute force sequential scan of the entire database.

This observation has led to index structures that work towards higher dimensionality. The X-tree extends the R-tree by introducing supernodes if a reasonable minimum bounding rectangle cannot be derived, and maintains a split dimension history to optimize the split choice in R*-trees [BKK96]. This approach alleviates problems with scalability, but cannot solve the problem entirely, as in high dimensional data overlap dominates the index eventually.

Other approaches have focused on improving the performance of the sequential scan directly. In the VA-file index, quantized representatives of the database are searched sequentially, and only potentially relevant representatives are refined by direct access to the respective multimedia objects [WSB98]. The IQ-tree and A-tree combine the idea of quantization with that of multidimensional index structures [BBJ$^+$00, SYUK00]. The IQ-tree approach uses different quantization at different levels of the tree and requires computation of the trade-off between finer resolution in quantization or splitting of nodes according to a cost model. The A-tree uses relative quantization with respect to the parent node and thus does not require costly recomputation. We compare our method against the A-tree in the experiments.

Another family of index structures is based on vantage points or other directory information based on relative distances to other objects in the database [Zez06]. Examples include MVP-tree, M-tree and the Slim-tree [BO97, CPZ97, TTSF99]. They are especially useful for metric databases where dimensional information is not necessarily available. In these cases, only other objects or their mutual distances can be used to create directories. They also suffer from poor performance when overlap is large [TTSF99].

In this work, we index multimedia data as in the TS-tree for sequential data. Our goal is to compute separators between nodes to guarantee overlap-freeness, yet maintain minimum bounding rectangles as in R*-trees.

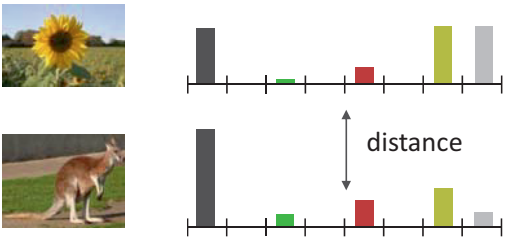



Figure 1: Feature vector based similarity

## 3 Multimedia indexing

Similarity models define the relevant characteristics of multimedia objects, for example color distribution in images, or pixel frequencies in shapes. Such features can be compared via distance functions which assign a degree of dissimilarity to any pair of histograms. An example is given in Figure 1. Two images and their respective color histograms, i.e. the relative frequencies of color pixels in each image, are depicted. To compare the images, distance functions compute a dissimilarity value on their histogram features.

We formally introduce the concept of feature vectors that can represent a variety of multimedia characteristics.

**Definition 1** ***Feature vector.***

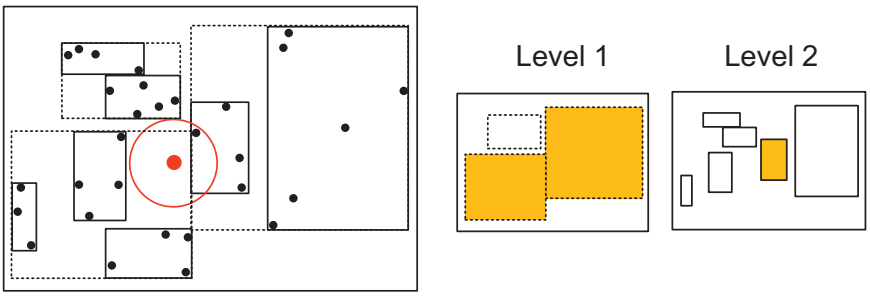
*For any multimedia object $o$ in database $DB$, a $d$-dimensional feature extraction $E$ is a mapping $E : DB \rightarrow R^d$ to a $d$-dimensional real valued feature vector $f = (f_1, \ldots, f_d)$ with bin entries corresponding to each of the dimensions $i \in \{1, \ldots, d\}$.*

An example that we will use throughout this work without loss of generality is (normalized) image color histograms. Then mapping $E$ is instantiated as the relative number of pixels corresponding to a certain color range in feature space:

$$f_i = \frac{|\{p_{xy} | p_{xy} \in b_i\}|}{|\{p_{xy}\}|}$$

where $p_{xy}$ denotes a pixel in position $x, y$, and $b_i$ denotes the corresponding bin per dimension $i$.

In multidimensional index structures, the main idea is to use the information contained in the individual dimensions. Hierarchical index structures create a small-size directory of the entire database based on those dimensions. This directory is used to direct the similarity search process to determine the result set. To do so, multidimensional features are grouped together according to their respective values. They are summarized by bounding geometries kept at the next higher levels in the directory. During retrieval, the query is compared to these bounding geometries to direct the search towards the result. In Figure 2(a), two-dimensional points are grouped in a hierarchy of minimum bounding rectangles. These rectangles are stored in a tree (Figure 3). Starting at the root, the query is




(a) R-Tree with isosurface for a 1-NN query

(b) page accesses

Figure 2: Minimum bounding rectangles geometrically

compared to the rectangles, thus choosing only paths down to the data that are relevant for the query. Comparison to rectangles using the Euclidean distance is straightforward. If the query $q = (q_1, \ldots, q_n)$ is smaller than the rectangle $\mathbf{MBR} = ((l_1, u_1), \ldots, (l_n, u_n))$ in that dimension, compute the distance to the lower boundary of the rectangle, if it is larger to the upper. Otherwise, the distance is zero:

$$dist_{MBR}(q, \mathbf{MBR}) = \sum_{i=1}^{n} \begin{cases} (q_i - l_i)^2 & q_i < l_i \\ (q_i - u_i)^2 & u_i < q_i \\ 0 & else \end{cases}$$

Figure 2(b) shows the page accesses for a Nearest Neighbor query for the query point in (a). Clearly, only a subset of the pages has to be accessed, resulting in a significant speed up.

As R*-trees grow bottom-up by splitting of minimum bounding rectangles if the corresponding node overflows, in high-dimensional spaces, many dimensions are not split at all or only once. This means that many of the MBRs extend across the entire unsplit dimension. Consequently, overlap is huge, and queries intersect with most of the minimum bounding rectangles. Figure 4 shows this problem in a low dimensional setting for illustrative purposes: if many MBRs overlap, the query $q$ at the center of the circular query range has to access most nodes in the tree, requiring access to the entire directory plus random read of the database. This worst case scenario is not typical for low dimensional spaces, but very common in high dimensional spaces.

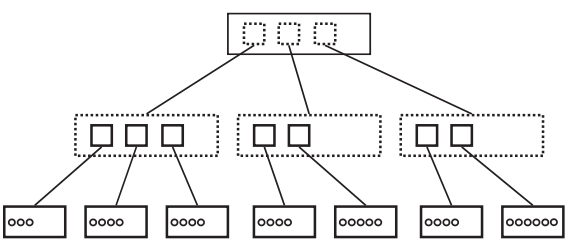
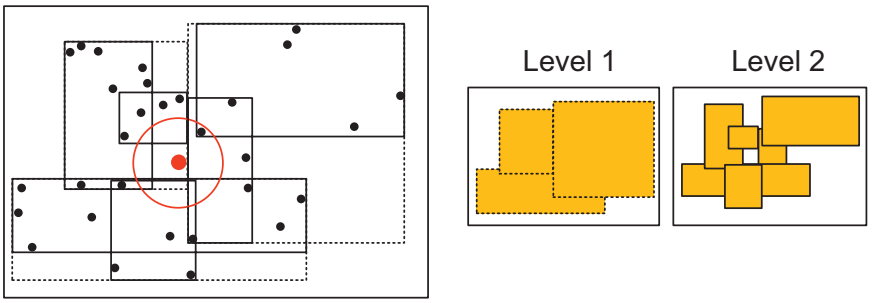


Figure 3: Hierarchy of minimum bounding rectangles




(a) R-Tree with isosurface for a 1-NN query

(b) page accesses

Figure 4: Overlap leads to index degeneration

To improve efficiency of multidimensional indexing, avoiding overlap is thus crucial.

# 4 Overlap-free indexing

We avoid overlap using a method successfully used in B-trees and TS-trees for one-dimensional and sequential data. The basic observation is that separator split leads to overlap-free descriptors. Consequently, the indexing concept relies on transformation and ordering of the feature vectors for this purpose. We show how this concept is used algorithmically in a lossless multistep filter-and-refine approach.

## 4.1 Indexing concept

Naively using a B-tree to index multimedia feature vectors would mean that only the first few dimensions are used for indexing [BM70, BM72, BU77]. Consequently, the first dimensions would dominate the query processing, and, even worse, only information on those first dimensions could be used before reaching leaf level. Thus, if there is no descriptor information on the remaining dimensions, we would see far too many disk accesses during query processing.

We therefore propose to proceed as in TS-trees for time series and include meta data on all dimensions, as well as principal components analysis to ensure that the first dimensions are the most relevant for indexing [AKAS08, Jol86]. For multimedia feature vectors, the additional meta data consists of minimum bounding rectangles, just as in R-trees or R*-trees.

**Definition 2** ***Separator.***

*A separator $S$ is the shortest string that is lexicographically larger than the left subtree $t_l$ and lexicographically smaller (or equal) than the right subtree $t_r$ : $t_l < S \leq t_r$.*

**Example. Separators according to default order.**

Assume a small database of only four feature vectors $f_1 = (0.1, 0.1, 0.1, 0.7)$, $f_2 = (0.1, 0.1, 0.2, 0.6)$, $f_3 = (0.1, 0.1, 0.3, 0.5)$, $f_4 = (0.1, 0.1, 0.4, 0.4)$. Using the order of dimensions as given in this example, a suitable separator to distinguish between the two smaller objects and the two larger objects would be $s = (0.1, 0.1, 0.3)$ with $s > f_1, f_2$ and $s \leq f_3, f_4$. A different method, e.g. based on the variances of the dimensions, could identify dimensions three and four as those with the most information, and would thus reorder the dimensions accordingly: $f_1' = (0.1, 0.7, 0.1, 0.1)$, $f_2' = (0.2, 0.6, 0.1, 0.1)$, $f_3' = (0.3, 0.5, 0.1, 0.1)$, $f_4' = (0.4, 0.4, 0.1, 0.1)$. Then, a separator to distinguish between the transformed feature vectors in the new space would be $s' = (0.3)$. This leads to separators that provide directory information on the most relevant dimensions in the database, thus providing good pruning power. Assuming that most queries follow the distribution in the database, queries might show values of $q = (0.1, 0.1, *, *)$, i.e. have a distance of zero to $s$ in the first two dimensions, leading to no pruning possibilities. Separators in the transformed space are moreover much shorter, and thus require less storage space which in turn leads to greater fanout.

Alternatively, the variance based reordering can be replaced by PCA (principal component analysis [Jol86]). Thus the statistical covariance in the dimensions is also considered by the transformation. In our algorithm and in the experimental section we use the PCA approach, while for illustrative purposes the variance based method is used.

Building separators on suitably ordered feature vectors still leaves one problem: on continuous valued vectors, separators would typically only be of length one, that is, be built just for the very first dimension. This is due to the fact that for real numbers, it is not very common to see the exact same values in many different objects. Consequently, splitting between different values does not require incorporating any further dimension and hence the objects are grouped only by the similarity in the first dimension. Therefore, discretization is necessary to ensure that separators include more dimensions.

**Example. Separators on continuous values.**

Given a database of five feature vectors $f_1 = (0.1, 0.1, 0.1, 0.7)$, $f_2 = (0.11, 0.29, 0.4, 0.2)$, $f_3 = (0.12, 0.38, 0.3, 0.3)$, $f_4 = (0.13, 0.07, 0.7, 0.1)$, $f_5 = (0.14, 0.1, 0.1, 0.7)$. One can see, that the two features $f_1$ and $f_5$ are very similar. But using the order of dimensions as given in this example, a suitable separator to distinguish between the two smaller objects and the three larger objects would be $s = (0.12)$ with $s > f_1, f_2$ and $s \leq f_3, f_4, f_5$. Thus, even though the objects $f_1$ and $f_5$ are very similar they are not grouped together, because they differ marginally in the first dimension. At the same time the separator length is only one, even though the first dimension provides low information. This fact cannot be uncovered by the separators, as real numbers that clearly distinguish between the values in the first dimension exist. By discretization of the objects, this effect is avoided. If only a certain number of ranges are used, separators will grow as more objects with the same discretized representation are indexed. For example, assuming a discretization with respect to the ranges $a = [0, 0.25)$, $b = [0.25, 0.5)$, $c = [0.5, 0.75)$, and $d = [0.75, 1]$, we get $\hat{f}_1 = (a, a, a, c)$, $\hat{f}_2 = (a, b, b, a)$, $\hat{f}_3 = (a, b, b, b)$, $\hat{f}_4 = (a, a, c, a)$, $\hat{f}_5 = (a, a, a, c)$. A

discretized separator would thus be $\hat{s} = (a, a, c)$ with $\hat{s} > \hat{f}_1, \hat{f}_5$ and $\hat{s} \leq \hat{f}_2, \hat{f}_3, \hat{f}_4$.

As we can see, discretization thus leads to automatic growth of separators, along with a mapping of similar feature vectors to the same discretized representation. Thus, discretization leads to a loss of information in feature vectors. To ensure correctness of the result, discretized representatives are used only in the index directory, whereas leafs store pointers to the original data. Upon reaching the leaf level, any potentially relevant feature vector in the original continuous-valued representation is used for comparison. Thus, during query processing, the query is compared against discretized ranges. Using a conservative distance computation, the actual distance is potentially underestimated. Refinement with the original features ensures that any false alarms are cleared out.

**Definition 3** ***Discretization.***

*Discretization of a feature vector $f$ to a partition $P = p_1, \ldots, p_m$ with $p_i = [l_{p_i}, u_{p_i})$ of adjacent ranges with $u_{p_{i-1}} = l_{p_i}$ maps each feature value $f_i$ to $p_j$ with $l_{p_j} \leq f_i < u_{p_j}$.*

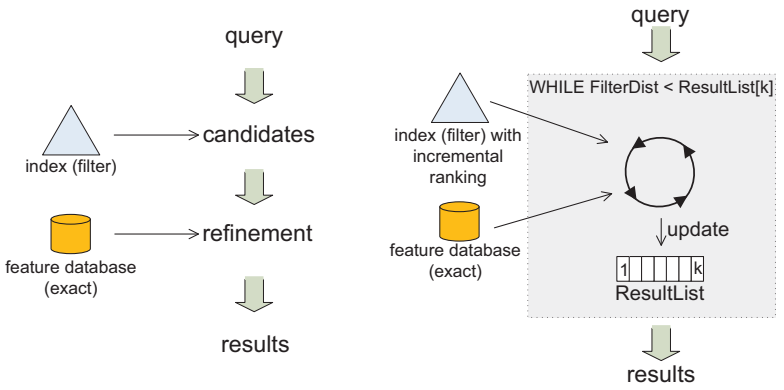
**Example. Query processing with discretized separators.**

In a database of five feature vectors $f_1 = (0.1, 0.1, 0.1, 0.7), f_2 = (0.11, 0.29, 0.4, 0.2)$, $f_3 = (0.12, 0.38, 0.3, 0.3), f_4 = (0.13, 0.07, 0.7, 0.1), f_5 = (0.14, 0.1, 0.1, 0.7)$. After variance based reordering and discretization, we get: $\hat{f}_1 = (c, a, a, a), \hat{f}_2 = (a, b, b, a)$, $\hat{f}_3 = (b, b, b, a), \hat{f}_4 = (a, c, a, a), \hat{f}_5 = (c, a, a, a)$. A suitable separator would be $s = (b)$ with $s > \hat{f}_2, \hat{f}_4$ and $s \leq \hat{f}_1, \hat{f}_3, \hat{f}_5$. Assuming a query $q = (0.1, 0.1, 0.1, 0.8)$ we want to find all features $f_i$ with a maximal (euclidean) distance of $\varepsilon = 0.1$ to $q$. Obviously the correct result set is $\{f_1\}$.
For query processing, $q$ is transformed to $\hat{q} = (d, a, a, a)$. Now we can calculate the distance between $\hat{q}$ and $s$, which results in a minimal value of $(d, a, a, a) - (b, *, *, *) = d - b = 0.75 - 0.5 = 0.25 > \varepsilon$. Therefore both features $\hat{f}_2, \hat{f}_4 < s$ can be pruned, because their distance could only increase. If we proceed with the distance computation to the remaining discretized features $\hat{f}_1, \hat{f}_3, \hat{f}_5$ the result set would be $\{\hat{f}_1, \hat{f}_5\}$.

Hence query processing on the discretized representation in the index structure alone would obviously result in incorrect results, as the exact value of the original feature vector is ignored. We therefore proceed in a multistep filter-and-refine fashion. Multistep algorithms, as e.g. in the GEMINI or KNOP frameworks speed up query processing by efficiently reducing the database to a small set of candidates via filtering [Fal96, SK98]. Only for those candidates the exact distance is used for refinement. The general idea is illustrated in Figure 5(a): the query is first evaluated with respect to the indexed representatives to obtain the set of candidates. To refine them, the full representation from the database is used.

To ensure that query processing via multistep filter-and-refine algorithms yields the desired efficiency gains, a number of quality criteria should be fulfilled. Summarized as *ICES* [AWS06], they refer to

- Index: to tap the full potential of database techniques for low runtimes, usage of indexing structures should be supported by the algorithm.




(a) GEMINI (b) KNOP

Figure 5: Multistep filter-and-refine frameworks

- Completeness: to ensure that efficiency is not at the cost of correctness of the result, the filter should not drop any true result.
- Efficiency: the filter step should be substantially more efficient than the refinement step, such that the overall algorithm is more efficient as well.
- Selectivity: the filter step should reduce the entire database to a small fraction of potential candidate objects, such that the costly refinement step has to be executed as little as possible.

Obviously, our approach offers index support. It is also complete, as we will prove below. Efficiency is guaranteed as computing the distance to the boundaries of the descriptors is efficiently possible. And finally, selectivity will be demonstrated in our experiments.

We deploy the KNOP multistep framework, as it minimizes the number of refinements in $kNN$ queries by keeping a $dist_{max}$ value of the current $k$th best nearest neighbor distance by immediate refinement of each potential candidate as it is demonstrated in Figure 5(b).

Formally, we ensure completeness of query processing via proof of the lower bounding property. Lower bounding property means that the distance computed between query and the index descriptors is never greater than the actual distance. Computing the distance between query and discretized features as the minimum of any feature vector within this range, fulfills the lower bounding property. As shown in [Fal96, SK98], lower bounding distance functions guarantee no false dismissals during multistep filter-and-refine processing according to the GEMINI or KNOP framework, respectively.

**Theorem 1** ***Lower bounding property of discretized feature vectors.***

*For any query feature vector $q$ and any feature vector $f$, the distance between $q$ and $f$ is lower bounded by the distance between $q$ and discretized feature vector $\hat{f}$:*

$$dist(q, \hat{f}) \leq dist(q, f)$$

**Proof 1**

*From the definition of the distance to upper and lower bounds, we immediately have*

$$dist(q,\hat{f}) = \sum_{i=1}^{n} \begin{cases} (q_i - l_i)^2 & q_i < l_i \\ (q_i - u_i)^2 & u_i < q_i \\ 0 & else \end{cases} \leq \sum_{i=1}^{n} \begin{cases} (q_i - f_i)^2 & q_i < l_i \\ (q_i - f_i)^2 & u_i < q_i \\ (q_i - f_i)^2 & else \end{cases} = dist(q,f)$$

□

### 4.2 The overlap-free tree

In this section we formalize the concepts presented in the previous section by introducing the overlap-free tree (OFT). With the discretized separator information at hand we can construct an overlap free index by adapting a B-Tree construction algorithm. The query processing for such an index was already shown in a previous example, where the separator information was used for pruning some feature vectors. To enhance the pruning power our index additionally stores MBRs in the index nodes. Thus pruning of subtrees is accessorily possible similar to the pruning methods in R-trees.

Formally, nodes in the index contain discretized representatives. As in TS-trees, we use finer partitioning to a degree $r$ of the leaf entries and the MBRs, and a rougher partitioning of the separators to ensure that more dimensions are used. Separators are simply added to the MBR information in R-trees or R*-trees, to rule out overlap during split, and identify relevant parts of the tree during query processing:

**Definition 4** ***Index node.***

*For branching factor $m$, rough discretization parameter $r$ and fine discretization parameter $f$ a node stores:*

- *$k$ entries ($m \leq k \leq 2m$) with MBRs and pointers to subtrees and additionally $k-1$ separators*
- *separators are discretized to the $r$ partitions and prefix compressed*
- *MBRs and leaf entries are discretized to the $f$ partitions*

The root node, of course, is allowed to contain fewer entries, i.e. between $2 \leq k \leq 2m$ if it is an inner node.

An example inner node is presented in Figure 6 (left). Each feature vector $\hat{f}$ covered by an entry/subtree, is located between the two neighboring separators (e.g. $\mathbf{Sep}_1 \leq \hat{f} < \mathbf{Sep}_2$) as well as within the MBR (e.g. $\mathbf{MBR}_2$). Figure 6 (right) illustrates this valid region for $\mathbf{MBR}_2$ and the two separators. In general we use a descriptor $\mathbf{D} = (\mathbf{MBR}, \mathbf{S_l}, \mathbf{S_r})$ to identify such regions.

During query processing, we could simply compute the distance only to the MBRs as in R-trees. However, as in B-trees or TS-trees, we may use the information given by

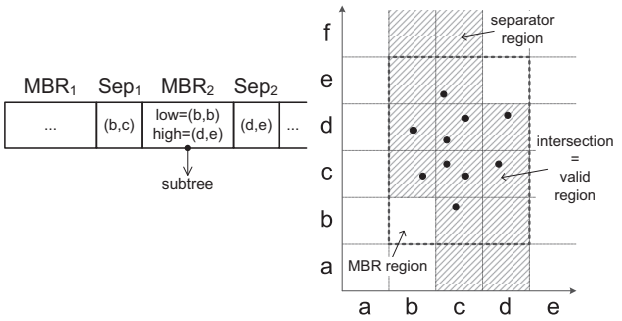



Figure 6: Example of an inner node (left) and the resulting regions in a 2d space (right)

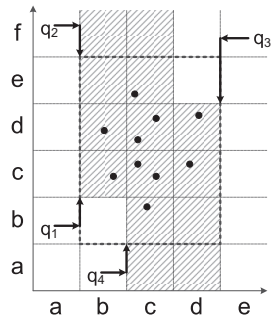



Figure 7: mindist to valid region for different query points

the separators not only for the split, but for query processing as well. Basically, we are interested in not only computing the minimum distance to the MBR or the separator, but instead to their intersection [AKAS08].

Figure 7 shows four different query points and their minimal distances to the valid region. As one can see, there are three different cases how the distances are determined. These cases mainly differ in the assignment of the query points to the three shaded "columns" (i.e. partitions in the first dimension) of the diagram.

- case A: If a query point is assigned to the first column, one has to ensure that the (virtual) point where the minimal distance could be realized, is lexicographically larger than the left separator (see $q_1$). Additionally we can add the distance to the MBR in the remaining dimensions, if the virtual point is still outside the MBR (see $q_2$).
- case B: The same argument holds if a query point is assigned to the last column (see $q_3$). First the lexicographical-smaller-relation to the right separator must be obtained and second the distance to the MBR (for the remaining dimensions) can be added.
- case C: In the last case a query point is assigned to a column in between (see $q_4$). The lexicographical relation to both separators is directly fulfilled, and one can add the distance to the MBR.

For a query $q$ we have to minimize over all three cases to get the minimal distance to the valid region. A running example with a (discretized) query $q = (a, a)$ and the descriptor from Figure 6 is illustrated in Figure 8.

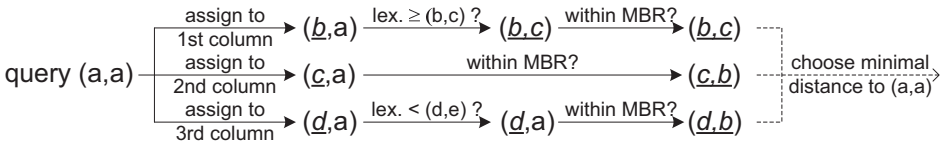



Figure 8: Schematic representation of the mindist computation

The rare situation (case D), where both separators starts with the same symbol ($\mathbf{S}_{\mathbf{l}1} = \mathbf{S}_{\mathbf{r}1}$), i.e. only one "column" exists, can be considered by calculating the minimal distance in this first dimension and go ahead in a recursive fashion. The complete minimal distance is formalized by:

**Definition 5** ***Minimum distance to MBRs and separators.***
*The* $mindist$ *between query* $q = (q_1, \ldots, q_n)$ *and a descriptor* $\mathbf{D} = (\mathbf{MBR}, \mathbf{S_l}, \mathbf{S_r})$ *of a subtree is defined as* $mindist(q, \mathbf{D}) := mindist(q, \mathbf{D}, 1)$*:*

$$mindist(q, \mathbf{D}, i) = \begin{cases} |q_i - \mathbf{S}_{\mathbf{l}i}| + mindist(q, \mathbf{D}, i+1) & \mathbf{S}_{\mathbf{l}i} = \mathbf{S}_{\mathbf{r}i} \quad (\textit{case D}) \\ mindist'(q, \mathbf{D}, i) & else \end{cases}$$

$$mindist'(q, \mathbf{D}, i) = \min \begin{cases} |q_i - \mathbf{S}_{\mathbf{l}i}| + mindist_{\geq}(i+1) & (\textit{case A}) \\ |q_i - \mathbf{S}_{\mathbf{r}i}| + mindist_{<}(i+1) & (\textit{case B}) \\ |q_i - \mathbf{S}| + dist_{MBR}(i+1), \forall \mathbf{S} : \mathbf{S}_{\mathbf{l}i} < \mathbf{S} < \mathbf{S}_{\mathbf{r}i} & (\textit{case C}) \end{cases}$$

*where:*

$$mindist_{\geq}(i) = \begin{cases} 0 & i > n \\ dist_{MBR}(i) & q_i > \mathbf{S}_{\mathbf{l}i} \wedge i \leq n \\ \min \begin{cases} |q_i - \mathbf{S}_{\mathbf{l}i}| + mindist_{\geq}(i+1) \\ |q_i - (\mathbf{S}_{\mathbf{l}i} + 1)| + dist_{MBR}(i+1) \end{cases} & else \end{cases}$$

$$mindist_{<}(i) = \begin{cases} \infty & i > n \\ dist_{MBR}(i) & q_i < \mathbf{S}_{\mathbf{r}i} \wedge i \leq n \\ \min \begin{cases} |q_i - \mathbf{S}_{\mathbf{r}i}| + mindist_{<}(i+1) \\ |q_i - (\mathbf{S}_{\mathbf{r}i} - 1)| + dist_{MBR}(i+1) \end{cases} & else \end{cases}$$

The functions $mindist_{\geq}(i)$ and $mindist_{<}(i)$ ensure in a recursive fashion, that the lexicographical order and the containment in the MBR is maintained, which is used by the two cases (A) and (B). In addition the expression $dist_{MBR}(i)$ is an abbreviation for the minimal distance from a point $q$ to a MBR, whereas both objects are restricted to the remaining dimensions $i \ldots n$, i.e.

$$dist_{MBR}(i) = dist_{MBR}(q_{|i \ldots n}, \mathbf{MBR}_{|i \ldots n}) = \sum_{j=i}^{n} \begin{cases} (q_j - l_j)^2 & q_j < l_j \\ (q_j - u_j)^2 & u_j < q_j \\ 0 & else \end{cases}$$

### 4.3 Algorithm

In this section, we give a general overview over our algorithm and discuss its inner workings. Algorithm 1 gives a pseudo-code description.

The algorithm starts by comparing the query against the information stored in the root node of the tree, i.e. the minimum bounding rectangles as well as the separators. Based

on the distance values computed, it decends the tree along the most promising path in terms of minimum distance until the first leaf node is refined. At this point, the actual data page containing the original feature vector representations is accessed. For all of these original feature vectors stored on this particular page, the algorithm sequentially compares the distance to the query. If their distances are better than the current set of $k$ nearest neighbors (which is obviously the case initially), they replace candidates in the result set that have a larger distance. $dist_{max}$, the variable that stores the distance of the $k$th nearest neighbor in the set, is updated accordingly. Query processing continues with the next most promising index or data page until the best filter distance is no longer smaller than the $k$th nearest neighbor. From the proof of the lower bounding property, we are now sure that the candidate set is exactly the true result set and return it to the user who issued the query.

**Algorithm 1** $k$NN queries in OFT

**input:** query $q$, result set size $k$
**output:** $k$ nearest neighbors

```
queue ⇐ List of (dist, Page) in ascending order by dist
queue.insert(ROOT)    // start with root page of OFT
resultArray ⇐ (dist, Objectfeatures) [k] = [(∞, NULL), ..., (∞, NULL)]
dist_max ⇐ ∞
while queue.hasNext and queue.nextDistance ≤ dist_max do
  p ⇐ queue.pollFirst
  if p.isDataPage then
    for each o in p do    // iterate over all feature vectors in the data page
      exactDist ⇐ Dist(q, o)
      if exactDist ≤ dist_max then
        resultArray [k] ⇐ (exactDist, o)
        resultArray.sort
        dist_max ⇐ resultArray [k] .dist
      end if
    end for
  else    // p is index page
    for each descriptor D in p do
      h = mindist(q, D)
      if h ≤ dist_max then
        queue.insert(h, D.childpage)
      end if
    end for
  end if
end while
return resultArray
```

The actual construction of the tree is given in Algorithm 2. We start by running a suitable feature extraction algorithm on the multimedia database that results in a database of feature vectors. For these feature vectors, we run principle components analysis to de-

rive an appropriate ordering of the dimensions. Depending on the dimensionality of the feature vectors, this step may also be used to reduce the overall dimensionality of the representation in the index. Please note that this reduction does not endager the lower bounding property, as the resulting distances may only decrease. Other dimensionality reduction techniques that fulfill this lower bounding property could be used as well. To prepare the feature vectors for separator construction, discretization to a partition of the value range is computed. The partition may be simply equi-width or pre-computed based on an assumption about the feature vector distribution, or even based on an analysis of the actual distribution. The feature vectors are simply mapped to their corresponding ranges as defined in Definition 3. After this step, the feature vectors have been pre-processed for indexing. The index is constructed in a bottom-up fashion, just like any B-tree, R-tree, R*-tree, etc. While the MBRs are constructed as in the R-tree family, the split itself is based entirely on the B-tree-style split to ensure that no overlap is incurred. Separators themselves are constructed like in B-trees as the shortest string that distinguishes between left and right subtree (cf. Definition 2).

**Algorithm 2** OFT construction

**input:** multimedia database $M$
  feature database $F \Leftarrow FeatureExtractor(M)$
  transformed database $T \Leftarrow PCA(F)$
  discretized database $D \Leftarrow Discretize(T)$
  **for each** $o$ in $D$ **do**
    Node $p \Leftarrow OFT.searchNode(o)$
    $p.insert(o)$
    $s \Leftarrow p.parentNode$
    **if** $p.isFull$ **then**
      $p_1, p_2 \Leftarrow p.SeparatorSplit$
      $s.computeMBRs(p_1, p_2)$
      $s.updateSeparators()$
      $s.propagateSplit()$
    **else**
      $s.computeMBRs(p)$
    **end if**
  **end for**

## 5 Experiments

We compare our approach with a recent index approach, the A-tree that combines MBRs with quantized virtual VBRs [SYUK00], and with the R*-tree [BKSS90]. The R*-tree uses floats (four bytes) per dimension to store the continuous values of the MBRs. For the A-tree, we use a discretization parameter of 8, which fared best in our preliminary experiments. All indexing structures are implemented using Java 1.6. We use implementation invariant performance measure like the number of pages accessed or the average capacity of the index pages as well as the number of refinements necessary. Experiments were

conducted on 2.33GHz Intel XEON machines running Windows Server 2008.

We thoroughly evaluate the performance of these approaches on several real world data sets.

- Data set 1 is a subset of the well known Corel database which consists of 59,870 photographs of different scenes.
- Data set 2 is a collection of 107,350 license free photos from the pixelio web page [Pix]. Due to constant updates by users of the web site, this database is very heterogeneous.
- Data set 3 comprises 103,642 images from several TV stations which were captured at an interval of at least 5 seconds.
- Data set 4 is the Hemera photo objects database containing 53,802 images [Hem]. Every object is represented multiple times in different positions.
- Data set 5 is the Aloi database [GBS05] which consists of 10,000 object images, each shown from 72 different directions; hence the database comprises 72,000 images.

We used color histograms in extended HLS color space as feature extraction method to obtain the feature vectors for all databases. The results of the conducted experiments are very similar for the different databases. Thus, we only show a selection of the outcomes.

### 5.1 Structural analysis

In our first experiment, we evaluate our overlap-free indexing structure in terms of its structural properties. We built R*-trees, A-trees, and OFT of different discretization parameter values (4, 16 and 64) and compared them for various dimensionalities on the Corel database. As we can see in Figure 9, the average number of entries per node and therefore also the fanout is substantially larger in our OFT approach. This is due to its discretization. The A-tree, while using quantized virtual bounding rectangles, does not show large capacity of nodes due to the additional overhead incurred from storing both the MBRs and the VBRs (quantized virtual bounding rectangles). In leaf nodes, the difference to the competing algorithms is generally smaller, as we maintain finer discretized representations at this level, which requires more storage overhead. This is done for the sake of reduction of the number of data page accesses. The better the information stored at leaf level, the fewer false hits require data page reads.

We have analyzed the effect of discretization on the number of page accesses in a very high dimensional setting. Figure 10 shows the results for the Corel database and the Pixelio database, respectively, on 60-dimensional color histograms and an index dimensionality of 16. Clearly, R*-trees do not scale to this index dimensionality. However, both the A-tree and our OFT approach fare much better and show reasonable numbers of page accesses for query processing. We can see that the best discretization strategy of our OFT algorithm is using between 8 and 16 equiwidth partitions per dimension. This strategy is slightly better than assigning the same number of partitions equidepth according to the normal

| *Tree* | *Type* | *Index Dimensionality* | | | | |
|---|---|---|---|---|---|---|
| | | *16* | *20* | *24* | *28* | *32* |
| **R*-Tree** | **Non-Leaf** | 22.78 | 18.44 | 16.40 | 14.05 | 12.16 |
| | **Leaf** | 46.08 | 37.74 | 31.74 | 28.01 | 24.36 |
| **A-Tree** | **Non-Leaf** | 24.59 | 19.76 | 16.51 | 13.64 | 12.50 |
| | **Leaf** | 110.66 | 89.09 | 77.15 | 65.50 | 57.01 |
| **OFT_s4** | **Non-Leaf** | 56.25 | 46.08 | 40.56 | 37.00 | 31.03 |
| | **Leaf** | 133.04 | 108.26 | 92.24 | 80.90 | 71.44 |
| **OFT_s16** | **Non-Leaf** | 55.50 | 47.63 | 40.26 | 38.50 | 37.38 |
| | **Leaf** | 134.84 | 114.25 | 99.12 | 86.39 | 76.26 |
| **OFT_s64** | **Non-Leaf** | 53.12 | 50.20 | 49.75 | 41.17 | 40.63 |
| | **Leaf** | 140.87 | 119.26 | 100.28 | 85.52 | 77.55 |

Figure 9: Capacity

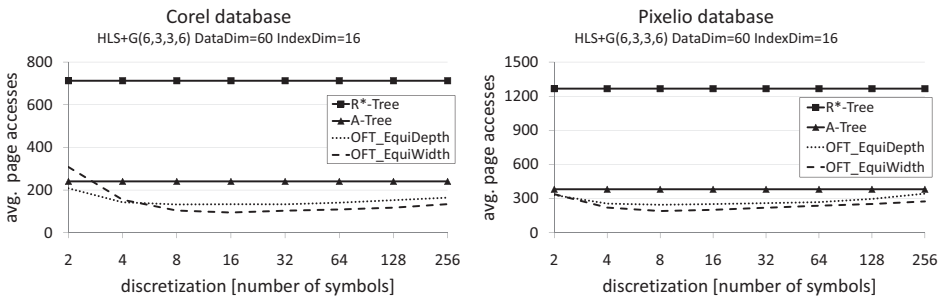



Figure 10: Effect of discretization

distribution. Apparently, the data does not follow a Gaussian pattern. Most importantly, our best technique saves more than 60% of the page accesses required by the A-tree.

## 5.2 Scalability

We further study the scalability of the OFT indexing structure compared to its competitors. Figure 11 illustrates the number of page accesses required for different index dimensionalities on the Hemera and Pixelio database. Starting from an original database of 60-dimensional feature vectors, we reduce them to different lower dimensional representations using PCA (principle components analysis, as discussed in Section 4.3). As we can see immediately from the graphs, dimensionality reduction is beneficial for all index structures as it reduces overlap in the competing approaches and reduces the storage usage in our OFT approach. Using a discretization to 16 equiwidth partitions clearly outperforms the competing techniques by at least an order of magnitude.

We further evaluate the performance on a data set of much higher feature vector dimensionality extracted on the Corel and TV databases. Figure 12 depicts the number of page accesses for different index dimensionalities starting from an original representation of

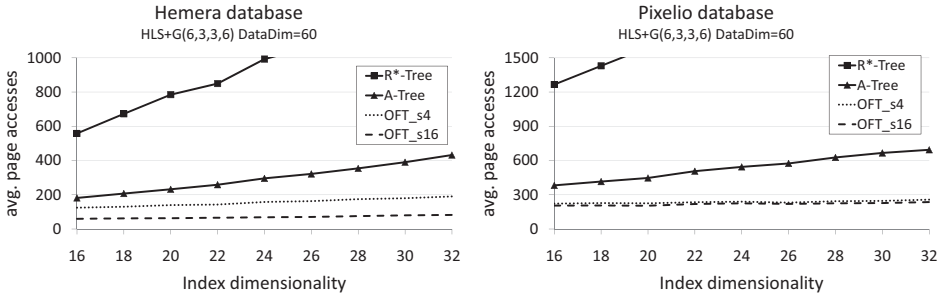



Figure 11: Variations of the index dimensionality

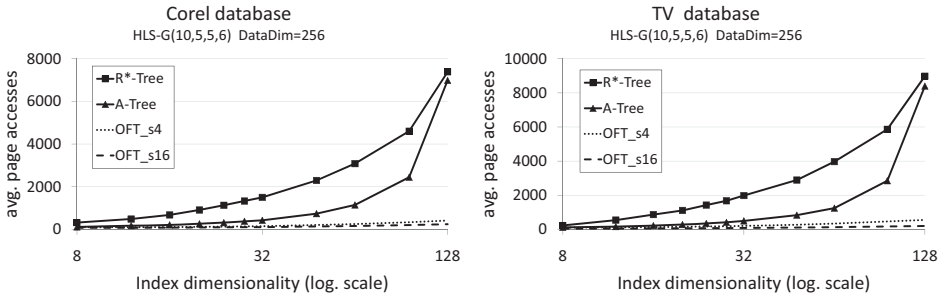



Figure 12: Variations of the index dimensionality in extreme cases

dimensionality 256. As we can see, R*-tree and A-tree suffer from increasing overlap in higher dimensionalities. Our OFT approach, however, scales extremely well. Even for this high dimensionality, the number of page accesses remains low.

To understand the effects of the difference in page accesses, we provide a more detailed experimental evaluation on the Corel, Pixelio, Hemera, Aloi and TV databases. Figure 13 therefore distinguishes between the number of index pages accessed and the number of actual data pages with the original representation. For all databases, we see that the number of index page accesses clearly dominates the number of data page accesses. OFT therefore outperforms A-trees and R*-trees mainly due to its informative index nodes that provide relevant information for pruning.

The effect of data dimensionality is analyzed in our next experiment. Figure 14 shows the results of fixed index dimensionality under varying original feature vector dimensionalities. Once again, we observe that R*-trees do not scale with the dimensionality of the data. Compared to the A-tree, we note that our best discretization scheme requires about one order of magnitude fewer page accesses. Moreover, the performance is very stable with increasing data dimensionality.

Finally, scalability with respect to database size is demonstrated. Figure 15 illustrates for a large-scale database, consisting of all 5 data sets, and a dimensionality of 16 and 32, respectively that R*-trees and A-trees do not scale very well with increasing database

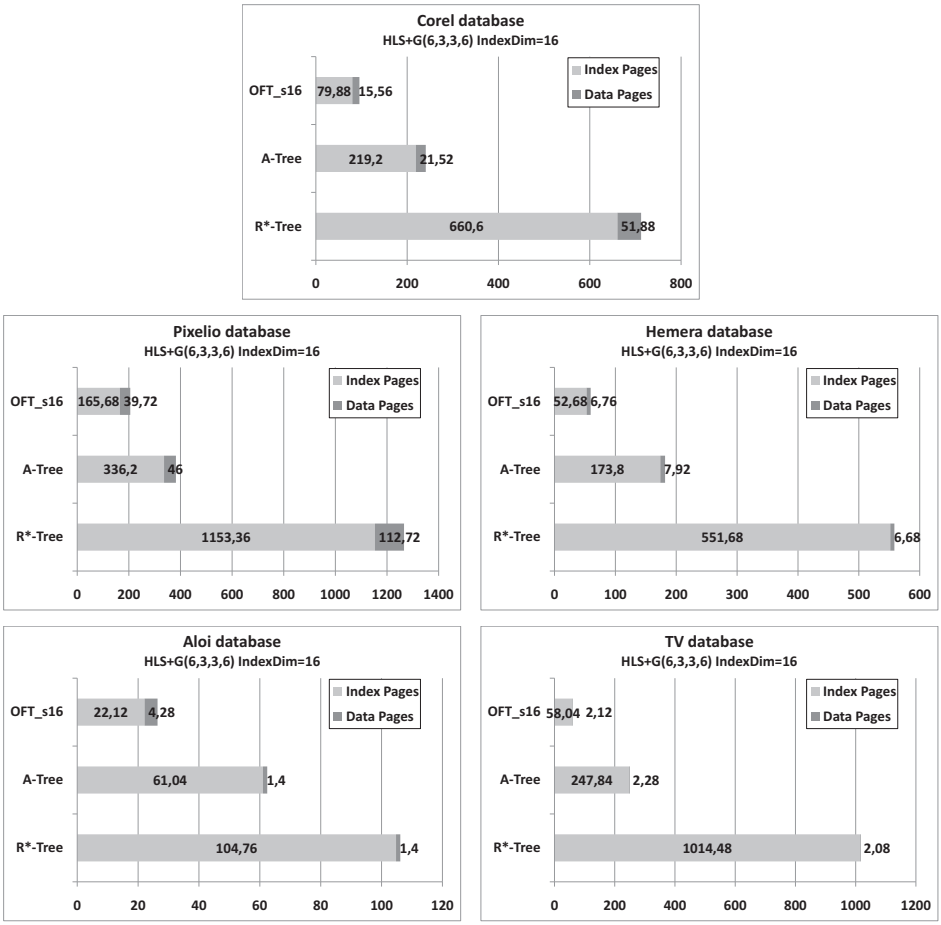



Figure 13: Index and data page accesses for data dimensionality = 60

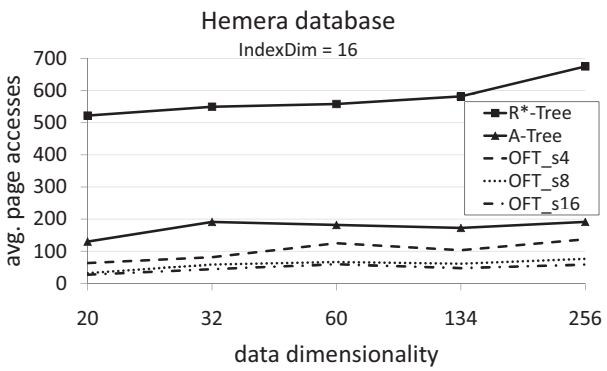
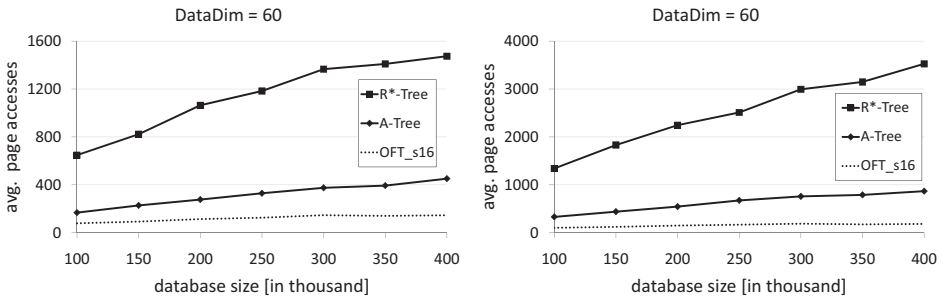



Figure 14: Influence of the data dimensionality




(a) index dimensionality = 16

(b) index dimensionality = 32

Figure 15: Scalability with respect to the dababase size

size. OFT, on the other hand side, shows remarkably good scalability with respect to the database size, clearly outperforming its competitors by at least one order of magnitude also in this respect.

## 6 Conclusion

Multimedia similarity search in large databases is a challenging task, especially for very high-dimensional feature vectors. In this work, we have proposed an approach that uses multimedia indexing structures to improve the overall query processing efficiency. A major obstacle for multimedia indexing structures is the fact that with increasing dimensionality, overlap leads to a degeneration of indexing structures. We therefore use the concept of separators to guarantee overlap-free splits. Separators require discretized values and ordering of dimensions, for which we presented a transformation that allows lossless query processing in an efficient multistep filter-and-refine algorithm. Our experiments demonstrate that our approach successfully scales multimedia similarity search to large and high-dimensional multimedia databases, clearly outperforming competing indexing structures.

## References

[AKAS08] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The TS-Tree: Efficient Time Series Search and Retrieval. In *EDBT*, 2008.

[AWS06] Ira Assent, Marc Wichterich, and Thomas Seidl. Adaptable Distance Functions for Similarity-based Multimedia Retrieval. *Datenbank-Spektrum Nr. 19*, pages 23–31, 2006.

[BBJ+00] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces. In *ICDE*, pages 577–588, 2000.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-Tree: An Index Structure for High-Dimensional Data. In *VLDB*, pages 28–39, 1996.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[BM70] Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indexes. In *Record of the 1970 ACM SIGFIDET Workshop on Data Description and Access,*, pages 107–141, 1970.

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1:173–189, 1972.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *SIGMOD*, pages 357–368, 1997.

[BU77] Rudolf Bayer and Karl Unterauer. Prefix B-trees. *ACM Transactions on Database Systems*, 2(1):11–26, 1977.

[CPZ97] Paulo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB, Athens, Greece*, pages 426–435, 1997.

[Fal96] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[FG99] Paolo Ferragina and Roberto Grossi. The String B-tree: A New Data Structure for String Search in External Memory and Its Applications. *Journal of the ACM*, 46(2):236–280, 1999.

[GBS05] J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders. The Amsterdam Library of Object Images. *Int. J. Comput. Vision*, 61(1):103–112, 2005.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57, 1984.

[Hem] Hemera Photo Objects. `http://www.hemera.com`.

[Jol86] Ian T. Joliffe. *Principal Component Analysis*. Springer, New York, 1986.

[MNPT06] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications*. Springer, London, 2006.

[Pix] License free photo database. `http://www.pixelio.de`.

[Sam06] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.

[SK98] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[SYUK00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *VLDB*, pages 516–526, 2000.

[TTSF99] C. Traina, A. Traina, B. Seeger, and C. Faloutsos. *Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes*. Springer, 1999.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*, pages 194–205, 1998.

[Zez06] P. Zezula. *Similarity Search: The Metric Space Approach*. Springer, 2006.

# Metrische Anpassung der Earth Mover's Distanz zur Ähnlichkeitssuche in Multimedia-Datenbanken

Christian Beecks, Marc Wichterich, Thomas Seidl
Data Management and Exploration, RWTH Aachen University
{beecks, wichterich, seidl}@informatik.rwth-aachen.de

**Abstract:** Die zunächst im Bereich Computer Vision vorgestellte Earth Mover's Distanz (EMD) hat aufgrund Ihrer hohen Flexibilität über eine Vielzahl von Anwendungen Einzug in die inhaltsbasierte Ähnlichkeitssuche in Multimedia-Datenbanken gehalten. Techniken zur effizienten Bearbeitung von EMD-Anfragen wurden in der jüngeren Vergangenheit vorgestellt und basieren zumeist auf einer mehrstufigen Bearbeitung mittels Indexierung oder Filterung der Datenbank. Eine Nutzung der Flexibilität der EMD zur nachträglichen Anpassung des Ähnlichkeitsmaßes wird jedoch nicht diskutiert. Der vorliegende Beitrag stellt ein mathematisches Rahmenwerk zur Anpassung des EMD-Maßes bei Erhalt der metrischen Eigenschaften vor.

## 1 Einleitung

Ständig wachsende Datenbanken multimedialen Inhalts erschweren es, einen umfassenden Überblick über die Daten zu erhalten. Auch die Anzahl solcher digitaler Datensammlungen steigt. Als Folge dessen ist es notwendig, große Mengen digitaler Daten übersichtlich und durchsuchbar zu strukturieren.

Der Wunsch nach ähnlichen Inhalten zu suchen wird durch Modelle der Ähnlichkeitssuche erfüllt. Sie unterstützen beispielsweise private Anwender bei der Suche nach ähnlichen Urlaubsbildern oder wissenschaftliche Anwender bei der Auswertung ihrer Arbeiten.

Ein flexibles, auf einem Optimierungsproblem basierendes Distanzmaß, welches an die vorhandenen Daten und an die subjektive Wahrnehmung der Ähnlichkeit angepasst werden kann, ist die sogenannte Earth Mover's Distanz (EMD) [RT01, AWS06b]. Für den effizienten Einsatz der EMD in Multimedia-Datenbanksystemen wurden in den letzten Jahren mehrere Techniken vorgestellt (siehe u. A. [LBS06, AWS06a, AWMS08, WAKS08]). Sie basieren zumeist auf einer mehrstufigen Anfragebearbeitung mittels Indexierung oder Filterung der Datenbank. Eine Nutzung der Flexibilität der EMD zur nachträglichen Anpassung des Ähnlichkeitsmaßes wird jedoch nicht diskutiert. Der vorliegende Beitrag[1] stellt ein mathematisches Rahmenwerk zur Anpassung des EMD-Maßes bei Erhalt der metrischen Eigenschaften vor.

[1] Die dem Beitrag zugrundeliegende Forschung ist Teil des DFG-Projekts "Schnelle inhaltsbasierte Suche in großen Multimedia-Datenbanksystemen mittels der Earth Mover's Distance" (SE 1039/1-2 und SE 1039/1-3).

## 2 Grundlagen und verwandte Arbeiten

In diesem Abschnitt werden für das weitere Verständnis notwendige Grundlagen und verwandte Arbeiten behandelt.

### 2.1 Ähnlichkeitssuche

Die Ähnlichkeitssuche ist ein Prozess, bei dem zu einem vorgegebenen Datenobjekt in einer Datenbank ähnliche Datenobjekte gesucht werden. Die Ähnlichkeit zwischen je zwei dieser Datenobjekte ergibt sich aus den Merkmalen der Datenobjekte unter Zuhilfenahme eines geeignet definierten Distanzmaßes.

Das Ergebnis einer Ähnlichkeitssuche in einer Datenbank kann beispielsweise die Menge aller Objekte $p$ der Datenbank sein, für welche die Distanz zu einem Anfrageobjekt $q$ kleiner gleich als eine gegebene Obergrenze $\varepsilon$ ist:

$$\{p \in DB | d(q,p) \leq \varepsilon\}$$

Für die Bearbeitung von Suchanfragen existieren effiziente Methoden. Häufig wird hierbei auf Indexstrukturen und Filter zurückgegriffen, welche in eine mehrstufige Anfragebearbeitung integriert werden [FRM94, KSF+98, SK98]. Einen Überblick über den gesamten Themenkomplex der Ähnlichkeitssuche gibt das Buch von Schmitt [Sch05].

### 2.2 Merkmale und deren Repräsentationen

Die für die inhaltsbasierte Ähnlichkeitssuche relevanten Merkmale eines Multimedia-Objektes können durch Histogramme in einer aggregierten Form erfasst werden. So lassen sich beispielsweise Bilder durch ihre Farben, Formen oder Texturen und das Layout dieser Merkmale beschreiben.

**Definition 1** *Sei $\mathcal{M}$ ein Merkmalsraum mit einer paarweise disjunkten Partitionierung $\mathcal{M} = \mathcal{M}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{M}_n$ und $p$ ein durch $p_1, \ldots, p_k \in \mathcal{M}$ beschriebenes Objekt. Ferner bezeichne $|\mathcal{M}_i^p| = \frac{1}{k} \cdot |\{p_j | p_j \in \mathcal{M}_i, 1 \leq j \leq k\}|$ den Anteil der einer Partition $\mathcal{M}_i$ angehörigen Elemente. Dann ist das Histogramm $h_p \in \mathcal{R}^n$ des Datenobjekts $p$ wie folgt definiert:*

$$h_p = (h_{p,1}, \ldots, h_{p,n}) \text{ mit } h_{p,i} = |\mathcal{M}_i^p|, 1 \leq i \leq n.$$

Ein Histogramm lässt sich hierbei als ein Punkt auf einer Hyperebene in einem $n$-dimensionalen Vektorraum auffassen. Die Summe der Histogrammeinträge ist jeweils 1. Im Falle von Farbhistogrammen sind die Merkmale $p_1, \ldots, p_k \in \mathcal{M}$ beispielsweise die Farbwerte von $k$ Pixeln eines Bilds, wobei $\mathcal{M}$ ein dreidimensionaler Farbraum ist (Bsp.: RGB, HSV).

## 2.3 Distanzmaße und die EMD im Speziellen

Neben den in Abschnitt 2.2 beschriebenen Merkmalen und deren Repräsentationen tragen Distanzmaße wesentlich zur Qualität der Ähnlichkeitssuche bei. Distanzmaße werden eingesetzt, um die Ähnlichkeit zwischen zwei Merkmalsrepräsentationen, im Folgenden Histogramme, zu messen. Ähnliche Repräsentationen der zu vergleichenden Objekte sollen hierbei eine kleine Distanz aufweisen.

**Definition 2** *Sei $d$ eine Distanzfunktion und x, y, z Elemente aus einem Raum $X$. Die Distanzfunktion $d : X \times X \mapsto \mathcal{R}^+$ ist eine Metrik, wenn sie für alle x, y, z die folgenden Eigenschaften erfüllt:*

- *Definitheit:* $d(x, y) = 0$ *gdw.* $x = y$
- *Symmetrie:* $d(x, y) = d(y, x)$
- *Dreiecksungleichung:* $d(x, y) \leq d(x, z) + d(z, y)$.

Metrische Distanzeigenschaften lassen sich für die EMD effizienzgewinnend ausnutzen [LBS06, AWS06a, AWMS08] und ermöglichen den Einsatz von Methoden zur distanzbasierten Indexierung (siehe [Sam06] Kapitel 4.5).

### Die Earth Mover's Distanz

Die Earth Mover's Distanz wurde von Rubner und Tomasi [RT01] im Bereich der Computer Vision vorgestellt. Das Anwendungsspektrum der Earth Mover's Distanz ist weit gefächert. Es existieren eine Vielzahl von Publikationen, welche die Earth Mover's Distanz als Ähnlichkeitsmaß für die Suche in Datenbanken einsetzen, z.B. auf Bildern [RT01, JLZZ04], auf Musikstücken [TGV$^+$03] oder auf Vektorfeldern in der Physik [LBH98].

Die Earth Mover's Distanz basiert auf dem Transportproblem der Linearen Optimierung, dessen Lösung durch eine hierfür optimierte Variante des Simplex-Algorithmus [HL90] berechenbar ist. Das Transportproblem beschreibt die Problematik, Produkte von Anbietern auf Verbraucher zu verteilen. Die Kosten, die für den Transport einer Produkteinheit von einem Anbieter zu einem Verbraucher anfallen, sind bekannt. Das Problem dieser Situation besteht darin, eine Verteilung der Produkte zwischen Anbietern und Verbrauchern mit minimalen Transportkosten zu finden. Übertragen auf die EMD, im Kontext von Histogrammen, berechnet diese die minimalen aufsummierten Kosten für die Transformation eines Histogramms in ein anderes Histogramm. Die Kosten für den "Transport" einer Einheit vom Histogrammeintrag $h_{p,i}$ nach $h_{q,j}$ werden hierbei durch eine Kostenmatrix $C = [c_{ij}]$ spezifiziert.

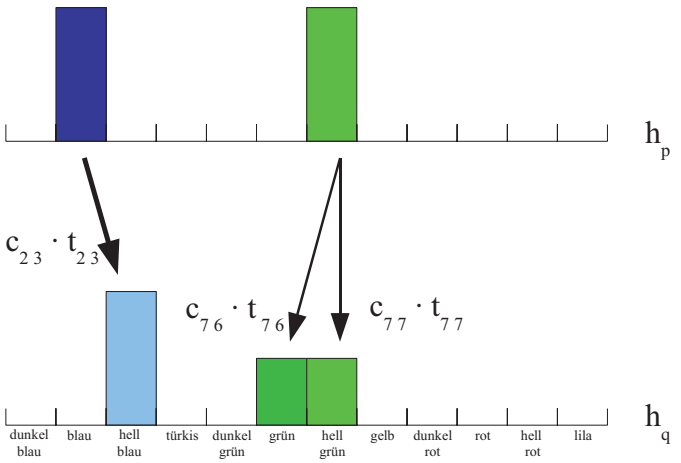



Abbildung 1: Die Earth Mover's Distanz berechnet den optimalen Transport $T = [t_{ij}]$ für die Transformation eines Histogramms $h_p$ zu einem Histogramm $h_q$ bei Kosten $C = [c_{ij}]$. Die Distanz beträgt hier $c_{23}t_{23} + c_{76}t_{76} + c_{77}t_{77}$.

**Definition 3** *Seien zwei Histogramme $h_q = (h_{q,1}, \ldots, h_{q,n})$ und $h_p = (h_{p,1}, \ldots, h_{p,n})$ und eine Kostenmatrix $C \in \mathcal{R}^{n \times n}$ gegeben. Die Earth Mover's Distanz $EMD_C(h_q, h_p)$ ist als Optimierungsproblem über alle zulässigen Transportmatrizen $T = [t_{ij}]$ definiert:*

$$EMD_C(h_q, h_p) = \min_{T \in \mathcal{R}^{n \times n}} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} t_{ij}$$

*unter den Einschränkungen*

$$t_{ij} \geq 0, \quad \forall 1 \leq i, j \leq n, \tag{1}$$

$$\sum_{j=1}^{n} t_{ij} \leq h_{q,i}, \quad \forall 1 \leq i \leq n, \tag{2}$$

$$\sum_{i=1}^{n} t_{ij} \leq h_{p,j}, \quad \forall 1 \leq j \leq n, \tag{3}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij} = 1. \tag{4}$$

Die Einschränkungen (1) bis (4) in der Definition 3 der Earth Mover's Distanz entsprechen der intuitiven Vorgehensweise bei der Transformation eines Histogramms in ein zweites Histogramm, wie in Abbildung 1 dargestellt. Die Gleichung (1) stellt sicher, dass alle auftretenden Transporte positiv sind. Die Gleichungen (2) und (3) limitieren die Transporte gemäß den Einträgen der Histogramme. Gleichung (4) formalisiert die Forderung, dass eine vollständige Transformation durchzuführen ist.

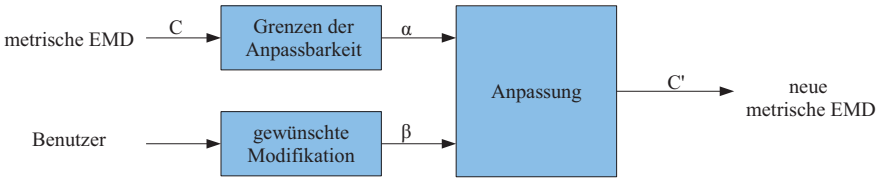



Abbildung 2: Schematischer Ablauf der metrischen Anpassung

# 3 Ein mathematisches Rahmenwerk zur Adaption der EMD

In diesem Abschnitt werden unsere Ansätze zur Anpassung der Earth Mover's Distanz auf Histogrammen unter Einhaltung der metrischen Eigenschaften beschrieben. Wir entwickeln ein Rahmenwerk welches die Anpassung der Einträge einer Kostenmatrix erlaubt. Abbildung 2 stellt die dazu notwendigen Schritte schematisch dar. Die Grenzen der Anpassbarkeit ergeben sich aus der Kostenmatrix der EMD und werden in Abschnitt 3.1 behandelt. Die eigentliche Anpassung, basierend auf der gewünschten Modifikation und den Grenzen der Anpassbarkeit, wird in Abschnitt 3.2 vorgestellt. Abschließend gibt Abschnitt 3.3 einen beispielhaften Ausblick auf mögliche Techniken zur Bestimmung der gewünschten Modifikation.

## 3.1 Grenzen der metrischen Anpassung

Die metrischen Eigenschaften des Distanzmaßes $EMD_C$ werden durch die verwendete Kostenmatrix $C$ impliziert. Der nachfolgende Satz zeigt, dass die Kostenmatrix $C$ dazu Eigenschaften erfüllen muss, welche analog zu denen aus Definition 2 sind.

**Satz 1** *Sei eine Kostenmatrix $C \in \mathcal{R}^{n \times n}$ gegeben. Die Earth Mover's Distanz $EMD_C$ ist metrisch gdw. folgende Eigenschaften für $1 \leq i, j, k \leq n$ erfüllt sind.*

- *Definitheit:* $c_{ij} = 0$ *gdw.* $i = j$
- *Symmetrie:* $c_{ij} = c_{ji}$
- *Dreiecksungleichung:* $c_{ij} \leq c_{ik} + c_{kj}$

**Beweisskizze.** Die Einträge $c_{ij}$ der Kostenmatrix $C$ implizieren ein Distanzmaß zwischen den Dimensionen $i$ und $j$ der zu vergleichenden Histogramme. Dieses erfüllt nach Definition 2 die Eigenschaften einer Metrik. Nach [RT01] ist somit auch $EMD_C$ eine Metrik. Ist eine der drei Eigenschaften für $C$ nicht erfüllt, so lässt sich jeweils ein Beispiel konstruieren, welches zeigt, dass dann auch $EMD_C$ nicht metrisch sein kann. □

Die Adaption der Earth Mover's Distanz erfolgt durch die Modifikation einzelner Einträge der Kostenmatrix. Die Modifikation einzelner Einträge kann durch Änderungsfaktoren realisiert werden, die für jeden Eintrag individuell bestimmbar sind. Eine Multiplikation eines Wertes der Kostenmatrix mit dem entsprechenden Änderungsfaktor bestimmt den resultierenden neuen Wert.

**Definition 4** *Sei eine Kostenmatrix $C \in \mathcal{R}^{n\times n}$ gegeben. Weiterhin seien Änderungsfaktoren $\alpha_{ij}$ gegeben. Dann sind die Einträge der neuen Kostenmatrix $C' \in \mathcal{R}^{n\times n}$ wie folgt definiert:*

$$c'_{ij} = c_{ij} \cdot \alpha_{ij}.$$

Die notwendigen Einschränkungen, welche bei der Veränderung der Werte $c_{ij}$ für $i \neq j$ gemäß der vorangegangenen Definition bestehen, ergeben sich aus Satz 1. Hierbei bedarf die Einhaltung der Dreiecksungleichung besonderer Betrachtung, da sie abhängig von den Verhältnissen der Werte untereinander ist. Das aus Satz 1 folgende Lemma zeigt die unteren und oberen Grenzen für die Anpassung eines Eintrags.

**Lemma 1** *Sei eine beliebige Kostenmatrix $C \in \mathcal{R}^{n\times n}$ einer metrischen EMD gegeben. Die maximal zulässige Verkleinerung bzw. Vergrößerung $\alpha_{ij}$ des Eintrags $c_{ij} (i \neq j)$ unter Einhaltung der Dreiecksungleichung ist durch die untere Grenze $0 \leq \alpha_{ij}^- \leq 1$ und die obere Grenze $1 \leq \alpha_{ij}^+ \in \mathcal{R}$ beschränkt:*

$$\alpha_{ij}^- = \frac{\max_{k\neq i,j}\{|c_{ik} - c_{jk}|\}}{c_{ij}} \quad \leq \quad \alpha_{ij} \quad \leq \quad \frac{\min_{k\neq i,j}\{c_{ik} + c_{jk}\}}{c_{ij}} = \alpha_{ij}^+.$$

Die Multiplikation des Änderungsfaktors $\alpha_{ij}$ in den zulässigen Grenzen $[\alpha_{ij}^-, \alpha_{ij}^+]$ mit dem Matrixeintrag $c_{ij}$ einer Kostenmatrix $C$ erhält also die Gültigkeit der Dreiecksungleichung. Für den Erhalt der Symmetrie muss jede Änderung an $c_{ij}$ analog auch für $c_{ji}$ durchgeführt werden. Weiterhin bedingt der Erhalt der Definitheit, dass $\alpha_{ij} > 0$ eingehalten wird. Der folgende Satz zeigt, dass eine Betrachtung von Werten $0 < \alpha_{ij} < 1$ jedoch gar nicht notwendig ist, da der Effekt einer jeden solchen Verkleinerung auch per Vergrößerung der übrigen Werte erzielt werden kann.

**Satz 2** *Sei eine beliebige Kostenmatrix $C \in \mathcal{R}^{n\times n}$ einer metrischen EMD gegeben. Eine Verkleinerung der Einträge $c_{ij}$ und $c_{ji}$ durch den Änderungsfaktor $\alpha > 0$ mit $\alpha_{ij}^- \leq \alpha \leq 1$ und $\alpha_{ji}^- \leq \alpha \leq 1$ ist äquivalent zu einer Vergrößerung der restlichen Einträge $c_{i'j'}$ durch den Änderungsfaktor $\frac{1}{\alpha} \geq 1$.*

**Beweis.** Bezeichne $C^- \in \mathcal{R}^{n\times n}$ die Kostenmatrix, die aus der Matrix $C$ entsteht, indem die Einträge $c_{ij}$ und $c_{ji}$ mit dem Faktor $\alpha$ multipliziert werden. Sei weiterhin $C^+ \in \mathcal{R}^{n\times n}$ diejenige Matrix, welche aus der Matrix $C$ durch Multiplikation aller Einträge $c_{i'j'}$ bis auf $c_{ij}$ und $c_{ji}$ mit dem Faktor $\frac{1}{\alpha}$ entsteht. Dann gilt gemäß der Konstruktion $\frac{1}{\alpha} \cdot C^- = C^+$. Die Matrix $C^-$ hält gemäß Lemma 1 die Dreiecksungleichung ein und ist per Konstruktion weiterhin symmetrisch und definit. Die Multiplikation von $C^-$ mit dem Faktor $\alpha > 0$

ändert nichts an den metrischen Eigenschaften der EMD. Somit ist $C^+$ bis auf Skalierung äquivalent zur Kostenmatrix $C^-$ und folglich auch $EMD_{C^+}$ metrisch. □

Die Vergrößerung mehrerer Einträge einer (ggf. skalierten) Kostenmatrix reicht somit aus, um den Effekt einer Verkleinerung eines einzelnen Eintrags zu erzielen. Jede beliebige metrische Modifikation einer Kostenmatrix der Earth Mover's Distanz ist folglich durch Vergrößerungen einzelner Werte und geeigneter Skalierung der Kostenmatrix möglich. Da bei Ähnlichkeitsanfragen vielfach nur das Verhältnis der Distanzen untereinander von Bedeutung ist, muss die Skalierung häufig nicht durchgeführt werden. In anderen Fällen lässt sich die Skalierung der Kostenmatrix auf die Skalierung der Eingabeparameter der Ähnlichkeitsanfrage zurückführen, da $EMD_{\alpha \cdot C} = \alpha \cdot EMD_C$ gilt.

### 3.2 Einfluss der gewünschten Anpassung

Bisher wurden die Grenzen der maximal zulässigen Veränderungen beschrieben. Die tatsächlich gewünschten Modifikationen werden im Folgenden auf die zulässigen Vergrößerungsintervalle $[1, \alpha^+_{ij}]$ abgebildet. Von der Bestimmung der gewünschten Modifikation soll hier zunächst abstrahiert werden. Einen Ausblick auf mögliche heuristische Verfahren gibt Abschnitt 3.3. Wir gehen hier davon aus, dass die Modifikationswünsche in Form von Einflussfaktoren $\beta_{ij} \in [0, 1]$ symmetrisch mit $\beta_{ij} = \beta_{ji}$ gegeben sind. Dabei soll sich die Relation untereinander möglichst gut in der Modifikation der Kostenmatrix widerspiegeln. Ein Wert von $\beta_{ij} = 0$ entspricht dem Wunsch, $\alpha_{ij}$ nicht zu vergrößern, während $\beta_{ij} = 1$ dem Wunsch nach möglichst starker Vergrößerung entspricht. Analog entspricht $\beta_{ij} = m \cdot \beta_{i'j'}$ dem Wunsch, $c_{ij}$ im Vergleich zu $c_{i'j'}$ prozentual $m$ mal so stark zu erhöhen.

Zunächst wählen wir vorläufige Vergrößerungen $\widetilde{\alpha}_{ij}$ unterhalb des Minimums $\alpha^+_{min}$ aller oberen Grenzen $\alpha^+_{ij}$, so dass den durch $\beta_{ij}$ spezifizierten Modifikationswünschen entsprochen wird.

**Definition 5** *Sei eine Kostenmatrix $C \in \mathcal{R}^{n \times n}$ und Einflussfaktoren $\beta_{ij}$ gegeben. Die vorläufige Vergrößerung $\widetilde{\alpha}_{ij} \in [1, \alpha^+_{min}]$ ist wie folgt definiert:*

$$\widetilde{\alpha}_{ij} = \beta_{ij} \cdot (\alpha^+_{min} - 1) + 1.$$

Alle $\beta_{ij}$ sind kleiner gleich 1, wodurch keines der $\widetilde{\alpha}_{ij}$ größer als $\alpha^+_{min}$ ist. Da sich aus der Definition der oberen Grenzen ergibt, dass die Vergrößerung eines Wertes die oberen Grenzen der anderen Vergrößerungen niemals verringert, können alle Anpassungen $\widetilde{\alpha}_{ij}$ durchgeführt werden und die Dreiecksungleichung gilt weiterhin. Ein Sonderfall tritt bei $\alpha^+_{min} = 1$ auf. Sind einzelne Einträge apriori gar nicht vergrößerbar, so kann den Modifikationswünschen nur teilweise entsprochen werden (durch Auslassung der entsprechenden Einträge bei der Bestimmung von $\alpha^+_{min}$ und bei der Anpassung selber).

Die in Definition 5 vorgestellten vorläufigen Änderungsfaktoren $\widetilde{\alpha}_{ij}$ bewirken die Vergrößerung der Einträge $c_{ij}$ der Kostenmatrix $C$ in einem minimalen, zulässigen Bereich. Hierbei wird jedoch die maximal zulässige Vergrößerung nicht notwendiger Weise er-

reicht. Durch einen einzelnen zusätzlichen Parameter $\lambda \in [0,1]$ als Einflussstärke lässt sich steuern, inwieweit die maximal zulässige Vergrößerung ausgeschöpft werden soll. Somit lassen sich dann schließlich die Änderungsfaktoren $\alpha_{ij}$ angeben:

**Definition 6** *Zu einer Kostenmatrix $C \in \mathcal{R}^{n \times n}$, durch Einflussfaktoren $\beta_{ij}$ bestimmte vorläufige Vergrößerungsfaktoren $\widetilde{\alpha}_{ij}$ und der Einflussstärke $\lambda$ sind die Änderungsfaktoren $\alpha_{ij}$ wie folgt definiert:*

$$\alpha_{ij} := \lambda \cdot \rho \cdot (\widetilde{\alpha}_{ij} - 1) + 1.$$

*Hierbei ist $\rho$ so zu wählen, dass $\alpha_{ij} \leq \alpha_{ij}^{+}$ für alle $1 \leq i, j \leq n$ gilt und dass die Gleichheit $\alpha_{ij} = \alpha_{ij}^{+}$ für mindestens ein Paar $i, j$ bei $\lambda = 1$ gilt.*

Nun ist nur noch $\rho$ passend zu bestimmen. Dazu zeigt der nächste Satz eine geeignete Wahl, welche alle geforderten Eigenschaften erfüllt.

**Satz 3** *Für $\alpha_{ij}$ gemäß Definition 6 und*

$$\rho = \min_{1 \leq i \neq j \leq n} \left\{ \frac{\alpha_{ij}^{+} - 1}{\widetilde{\alpha}_{ij} - 1} \right\}$$

*als der kleinsten prozentualen Vergrößerung der vorläufigen Änderungsfaktoren $\widetilde{\alpha}_{ij}$ gilt:*

$$\exists (i^*, j^*) : \left( \alpha_{i^* j^*} = \alpha_{i^* j^*}^{+} \wedge \forall (i', j') \neq (i^*, j^*) : \alpha_{i' j'} \leq \alpha_{i' j'}^{+} \right)$$

**Beweis.** Bezeichne $i^*, j^*$ ein Paar von Indizes, welche $\rho$ gemäß Satz 3 minimieren. Dann gilt für $\lambda = 1$:

$$\begin{aligned} \alpha_{i^* j^*} &= \lambda \cdot \rho \cdot (\widetilde{\alpha}_{i^* j^*} - 1) + 1 \\ &= \frac{\alpha_{i^* j^*}^{+} - 1}{\widetilde{\alpha}_{i^* j^*} - 1} \cdot (\widetilde{\alpha}_{i^* j^*} - 1) + 1 \\ &= \alpha_{i^* j^*}^{+} \end{aligned}$$

Sei nun $i', j'$ ein beliebiges Paar ungleich $i^*, j^*$ und $\lambda$ beliebig aus $[0,1]$. Dann gilt:

$$\begin{aligned} \alpha_{i' j'} &= \lambda \cdot \rho \cdot (\widetilde{\alpha}_{i' j'} - 1) + 1 \\ &\leq 1 \cdot \rho \cdot (\widetilde{\alpha}_{i' j'} - 1) + 1 \quad = \quad \frac{\alpha_{i^* j^*}^{+} - 1}{\widetilde{\alpha}_{i^* j^*} - 1} \cdot (\widetilde{\alpha}_{i' j'} - 1) + 1 \\ &\leq \frac{\alpha_{i' j'}^{+} - 1}{\widetilde{\alpha}_{i' j'} - 1} \cdot (\widetilde{\alpha}_{i' j'} - 1) + 1 \quad = \quad \alpha_{i' j'}^{+} \end{aligned}$$

□

Nachdem schließlich die zentrale Definition 6 die metrische Anpassung der Earth Mover's Distanz über die Wahl der Einflussfaktoren $\beta_{ij}$ ermöglicht, beschreiben wir nachfolgend kurz einige Ansätze zur Wahl dieser Einflussfaktoren.

### 3.3 Zur Wahl der Einflussfaktoren

Die metrische Anpassung der Earth Mover's Distanz basiert auf der Modifikation der Kostenmatrix mittels den Einflussfaktoren $\beta_{ij}$. Während die Modifikation in dem vorangegangenen Abschnitt theoretisch beschrieben wird, soll nachfolgend die Frage nach Heuristiken zur Bestimmung des Anpassungsbedarfs beispielhaft diskutieren werden.

Eine erste Möglichkeit besteht in einer direkten Bewertung der Ähnlichkeitswahrnehmung im Merkmalsraum durch den Benutzer. Ist der Merkmalsraum z.B. ein Farbraum, so steht jeder Histogrammeintrag für einen Farbbereich. Nun ist es möglich, dem Benutzer Paare von repräsentativen Farben zu präsentieren und nach der Ähnlichkeit auf einer fixen Skala zu fragen. Werden beiden Farben als unähnlicher eingeschätzt als durch die Kostenmatrix abgedeckt, so kann ein entsprechend großer Wert $\beta_{ij}$ gesetzt werden. Fällt es dem Benutzer schwer die Ähnlichkeit auf einer Skala anzugeben, so kann dies gegebenenfalls durch die Verwendung von mehr als zwei Farben pro Einschätzung vermieden werden. Hierbei muss der Benutzer nur entscheiden, ob eine gegebene Farbe ähnlicher oder unähnlicher zu einer zweiten im Vergleich zu einer dritten ist.

Statt eine Bewertung im, für den Benutzer eventuell abstrakten, Merkmalsraum vorzunehmen, kann dies über Techniken des Relevance Feedbacks auch direkt auf der Ebene der Datenbankobjekte geschehen. Hierbei können zum Beispiel statistische Eigenschaften, der durch den Benutzer als relevant eingeschätzten Objekte, in Form einer Kovarianzmatrix $K \in \mathcal{R}^{n \times n}$ gesammelt werden und Rückschlüsse von $k_{ij}$ auf $\beta_{ij}$ gezogen werden. Durch eine datenbezogene Anpassung läßt sich das vorgestellte Rahmenwerk ggf. für EMD-spezifisches "Distance Metric Learning" [YJ06] einsetzen.

Die konkrete Auswahl und die Untersuchung benutzerbasierter Heuristiken, welche sich in das hier vorgestellte Rahmenwerk integrieren, wollen wir in weiteren Arbeiten angehen.

## 4 Zusammenfassung und Ausblick

In unserem Beitrag haben wir ein mathematisches Rahmenwerk für die Anpassung der äußerst flexiblen Earth Mover's Distanz bei Erhalt der metrischen Eigenschaften vorgestellt. Zusammen mit den in der letzten Zeit vorgestellten Beschleunigungstechniken lassen sich nunmehr effiziente und effektive EMD-basierte Systeme zur Ähnlichkeitssuche in großen Datenbanken erstellen, welche an die Anwendungsanforderungen angepasst werden können.

In weiteren Arbeiten planen wir konkrete Strategien zur Wahl der Anpassung mittels des hier vorgestellten Rahmenwerkes zu entwickeln.

## Literatur


[AWMS08] I. Assent, M. Wichterich, T. Meisen und T. Seidl. Efficient Similarity Search Using the Earth Mover's Distance for Large Multimedia Databases. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2008.

[AWS06a] I. Assent, A. Wenning und T. Seidl. Approximation Techniques for Indexing the Earth Mover's Distance in Multimedia Databases. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2006.

[AWS06b] I. Assent, M. Wichterich und T. Seidl. Adaptable Distance Functions for Similarity-based Multimedia Retrieval. *Datenbank-Spektrum*, 6(19):23–31, 2006.

[FRM94] C. Faloutsos, M. Ranganathan und Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, 1994.

[HL90] F. S. Hillier und G. J. Lieberman. *Introduction to Mathematical Programming*. McGraw-Hill, 1990.

[JLZZ04] F. Jing, M. Li, H. Zhang und B. Zhang. An Efficient and Effective Region-based Image Retrieval Framework. *IEEE TIP*, 13(5):699–709, 2004.

[KSF+98] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel und Z. Protopapas. Fast and Effective Retrieval of Medical Tumor Shapes. *IEEE TKDE*, 10(6):889–904, 1998.

[LBH98] Y. Lavin, R. Batra und L. Hesselink. Feature Comparisons of Vector Fields using Earth Mover's Distance. In *Proc. IEEE Conf. on Visualization*, 1998.

[LBS06] V. Ljosa, A. Bhattacharya und A. K. Singh. Indexing Spatially Sensitive Distance Measures Using Multi-resolution Lower Bounds. In *Proc Int. Conf. on Extending Database Technology (EDBT)*, 2006.

[RT01] Y. Rubner und C. Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer Academic Publishers, 2001.

[Sam06] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[Sch05] I. Schmitt. *Ähnlichkeitssuche in Multimedia-Datenbanken - Retrieval, Suchalgorithmen und Anfragebehandlung*. Oldenbourg Verlag, 2005.

[SK98] T. Seidl und H.-P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, 1998.

[TGV+03] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering und R. van Oostrum. Using Transportation Distances for Measuring Melodic Similarity. In *Proc. Int. Conf. on Music Information Retrieval (ISMIR)*, 2003.

[WAKS08] M. Wichterich, I. Assent, P. Kranen und T. Seidl. Efficient EMD-based Similarity Search in Multimedia Databases via Flexible Dimensionality Reduction. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, 2008.

[YJ06] Liu Yang und Rong Jin. Distance Metric Learning: A Comprehensive Survey, 2006. `http://www.cse.msu.edu/~yangliu1/frame_survey_v2.pdf`.

# On the Need of Data Management in Automotive Systems

Sandro Schulze*, Mario Pukall, Gunter Saake, Tobias Hoppe, and Jana Dittmann
School of Computer Science
University of Magdeburg, Germany
{sanschul, mario.pukall, saake, tobias.hoppe, jana.dittmann}@iti.cs.uni-magdeburg.de

**Abstract:** In the last decade, automotive systems changed from traditional mechanical or mechatronical systems towards software intensive systems, because more and more functionality has been implemented by software. Currently, this trend is still ongoing. Due to this increased use of software, more and more data accumulates and thus, has to be handled. Since it was no subject up to now to manage this data with software separatly, we think that it is indispensable to establish a data management system in automotive systems. In this paper we point out the necessity of data management, supported by exemplary scenarios, in order to overcome disadvantages of current solutions. Further, we discuss main aspects of data management in automotive systems and how it could be realized with respect to the very special restrictions and requirements within such a system.

## 1 Introduction

An automotive system encompasses the hardware, i.e., sensors, actuators, and electronical control units *(ECU)*, and the several bus systems used for their connection and communication amongst each other in a modern car. Furthermore, the software implemented to fulfill more and more functionality in a car, belongs to such a system. Especially the latter increased rapidly in the last decade, making an automotive system a software-intensive IT system. More precisely, it is estimated that in 2010 approximately 1 GB of software is installed in automotive systems [PBKS07]. This evolution is accompanied by an increasing amount of data [CRTM98]. Additionally, typically mechanical components are substituted by electronical ones, e.g., considering the *X-By-Wire* technology or driver assistance systems like *ESP (Electronic Stability Program)*. All these mentioned aspects lead to a highly complex system and in near future, the complexity is expected to increase further due to new technologies like *Car-To-Car (C2C)* or *Car-To-Infrastructure (C2I)* [ZS07, S. 387]. Hence, it is crucial to guarantee reliability and safety of an automotive system while providing an efficient and flexible management of data.

In fact, the data in such a system is managed ad hoc by each ECU on its own, using internal data structures. Subsequently, problems which may occur (e.g., inconsistencies or concurrency problems) are solved locally, using mechanisms implemented directly on the hardware. This, in turn, not only increases the already high complexity of the overall sys-

*The author is funded by the EU through the EFRE Programme under Contract No. C(2007)5254

tem. Moreover, this approach decreases the flexibility, extensibility and maintainability of the system. Since a change of the overall system configuration (e.g., new functionality resulting in new or altered data) probably entails necessary changes of one ore more hardware implementations, this finally leads to increasing development costs. Thus, we conclude that a *data management system (DMS)* is inevitable to ensure the flexibility needed in automotive systems and beyond it, to decrease the development costs. Furthermore, such a DMS may be useful to ensure the reliability and safety of the system, its users and the respective environment.

In this paper, we introduce the idea of establishing a DMS in automotive systems and thus, to increase the already mentioned properties, e.g., efficiency, flexibility, or maintainability. Hence, we will discuss several motivating aspects for clarifying potential advantages of DMS in automotive systems. Furthermore, we examine the usefulness of integrating security mechanisms within such a DMS in order to address safety and reliability as well. Finally, we discuss how to address the particular conditions of an automotive system, regarding selected data management aspects.

## 2 Background

In the following we give a brief overview on automotive systems and the IT security in such systems.

### 2.1 Automotive Systems

An automotive system is a complex networked IT system, which is characterized by a frequent interaction of its components, in detail, dozens of ECUs, sensors and actuators. Each component can be seen as a self-contained embedded system, which leads to a highly heterogeneous character of the overall system. The heterogenity is tightened, since usually different ECUs are provided by different manufacturers. The communication between these components takes place via bus systems, primarily over CAN, but also LIN or MoST are used [ZS07, S. 36 ff.]. An examplary part of such an automotive system is depicted in Figure 1.

The different sensors ($S_i$) and actuators ($A_i$) are directly connected with the ECUs, which in turn, are connected via bus systems for communication. Furthermore, the ECUs are grouped into subbus systems according to their functionality. In our example, three subbus systems are depicted, namely *Comfort, Infotainment* and *Power Train* subbus system. The subbus systems differ slightly regarding conditions and constraints to be met. For instance, the power train system has hard real time constraints and thus, the transmission rate is higher than in other subbus systems. Apart from that, data can be exchanged between any arbitrary ECUs, regardless what subbus system they belong to. Furthermore, the exchanged data can be discrete as well as continuous, whereas the differences are due to the distribution of the data. The bus protocol distributes the continuous data in a time-triggered

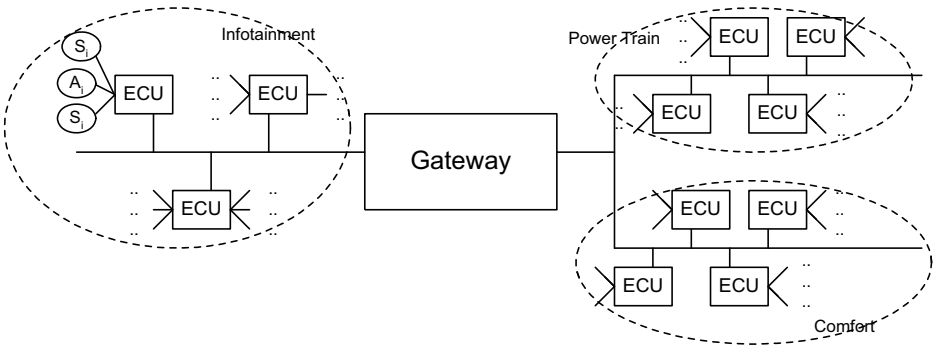



Figure 1: Exemplary Part of an Automotive System

way, i.e., the data is only valid for a certain time, while the discrete data is distributed in an event-triggered way (i.e., if new data is available, the ECUs are notified of this fact).

Usually, a micro controller commonly used in ECUs of current automotive systems (e.g., in off-the-shelf cars) has only a memory of 40-50 KB on average (distributed over RAM and EEPROM) and the computing power is less than 10 MHz. Additionally, the real time requirements within such a system are in dimensions of some milliseconds (with an upper limit of 10 ms), no matter which subbus system is considered. Thus, software development (in our case a DMS) is a challenging task for automotive systems.

### 2.2 IT Security in Automotive Systems

The IT security in automotive systems was neglected for a long time, but becomes more and more a focus of research due to the potential devastating consequences when it is violated. Per definition, security means reliability in terms of preserving security aspects of information (within a system) [HKLD07]. If all requirements are met to ensure these security aspects, namely *confidentiality, integrity, availability, non-repudiability* and *authenticity*, the system can be considered to be secure. In order to achieve a secure system, appropriate security mechanisms must be applied to the system.

An automotive system exhibits multiple opportunities for access from *outside* (e.g., by exploiting security vulnerabilities in wireless communication systems or by encompassing the use of manipulated media discs) or *inside* the car (e.g., directly by injected malicious code or additional devices which are physically attached either to explicit communication interfaces or implicitly to hooked-up bus wires). Thus, it is prone to malicious attacks, whereas the reasons can be diversified, e.g., tuning purposes or monetary interests. Examples for successful attacks can be found in [BD07, HD07, Paa08, HKD08b]. Although these attacks differ in the underlying approach as well as the target, they all have in common, that they aim at the manipulation of data. Subsequently, the mentioned security aspects have to be ensured for the *data* in automotive systems to keep the overall system secure. This data-centered character, and the fact, that violated security aspects can have

implications for the safety and reliability, make it worth to think about integrating security mechanisms into a data management system.

## 3 Exemplary Scenarios for Data Management in Automotive Systems

In this section we introduce exemplary scenarios for automotive systems, where a data management system is useful or even inevitable.

### 3.1 Uniform Data Structures

In current automotive systems, the data management is distributed as a hardware solution over the participating ECUs. Thus, each ECU is only responsible for the local data and its quality (e.g., consistency or availability) using internal data structures. This approach not only increases the already high complexity and heterogeneity of such systems, moreover, it leads to difficulties regarding the verification of the overall system. For instance, if data is distributed over several ECUs, each of them has to validate that the data is also the most recent one. Furthermore, the current decentralized hardware solution increases the I/O operations. Thus, a uniform data structure throughout the system, as provided by a data management system, is desirable, regarding an efficient data management. At first glance, this might be questionable because the current hardware solution offers a better performance (considering a single ECU) compared to a software solution like a DMS. However, the advantages of uniform data structures as provided by a DMS overcome the disadvantage of decreased performance. For instance, with uniform data structures, the data in an automotive system can be captured global, which facilitates validation (of data) or verification of the system. Furthermore, due to the structure, efficient data access can be achieved (e.g., using indexes) and thus, the performance can be increased. Moreover, uniform internal structures allows for uniform external representations of data as well.

### 3.2 Concurrency Control

Another reason for a DMS is the concurrency control, which ensures the integrity of the data and thus, the reliability of the system. In current automotive system the protocol of the bus system, mostly CAN, uses a kind of prioritization to control the (write) access rights for the bus, known as *bus arbitration* [ZS07, S.23 ff.]. In detail, an ECU needs the highest priority for a certain message (identified by a message ID) to put the data, contained in this message, on the bus. Additionally, on each ECU certain (local) mechanims are implemented to ensure that the current data is the most recent one. In Figure 2 a simplified concurrency scenario is depicted for only two ECUs. The data *x*, available on the field bus, is read by the $ECU_A$ at time $t_s$. After manipulating the data, the $ECU_A$ writes the data $x'$ at time $t_{s+n}$, so that $x'$ is now the valid data. At the same time (or even a subsecond

before) another ECU ($ECU_B$) reads the ”old” data $x$ for further usage, e.g., computation of other data. If $ECU_B$ does not notice that the data $x$ is out of date, the further computed data is not valid and thus, may endanger the reliability of the system.

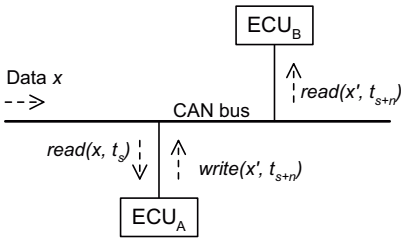



Figure 2: Example for Concurrent Access of Data

However, with the expected increase of complexity, due to additional electronical components or software, this approach is not appropriate anymore. Even today, it is difficult for a single ECU to ensure that the current data is up to date with its hard-coded mechanisms, since the data is scattered throughout the system at more than two ECUs, as supposed in our example.

Hence, a DMS may be useful to overcome this crucial situation by supporting a global view on the data. In the sense of concurrency control, *transaction management (TXM)* functionality could be used, to implement and coordinate simultaneous data access in automotive systems. By using respective strategies for assigning write access to ECUs, the efficiency and the reliability could be increased compared to the approach currently used.

### 3.3 Access Management

As already denoted in Section 2.2, a third person (e.g., an attacker) can participate in an automotive system by hooking up a device with the bus wire. Subsequently, the device can ”listen” to the data, i.e., an unauthorized access has been established. Additionally, this external device can provide manipulated data as well as demand for certain data of another ECU, which is a serious problem regarding safety and reliability of the system. Since no solutions preventing such intrusion exist, *access management* functionality of a DMS could be useful to tackle this problem. If such a management knows of all participating ECUs (or at least the respective ones for a certain data), it can control the read and write access to the data (or the message it is contained in) and preserve the system from contamination through falsified data. Additionally, security mechanisms, e.g., digital signatures or public key infrastructures, could be integrated to hinder the access to the data.

Of course, the current technology (e.g., CAN) does not support this approach. But future communication technologies, e.g., C2C, C2I or any other wireless communication, provide the respective opportunities to establish such functionality. Thus, access control should be considered seriously as an further advantage of data management in automotive systems.

### 3.4 Trip Recorder for Insurance Purposes

This scenario points out which additional facilities a data management system can provide in automotive systems. It covers a common situation in the day-to-day transportation - an

accident. In the case that there are no witnesses involved in such an accident, it is difficult to reproduce how the incident occured. Especially insurances are interested in the event reconstruction, even if this is only possible in parts. Thus, the information given by certain data could be employed for looking back at what happened exactly. Since not all data can contribute to the clarification, only those with a high degree of information should be considered. For instance, the data containing the speed or the steering angle provide useful information. Furthermore, the current state of the driver assistance systems (if present) or flags, if the light or the indicator signal was activated, are conceivable. However, if an accident occurs, an event has to be activated inducing the DMS to write the respective data into a protected partition of the persistent storage. After a further preparation of the data (e.g., information retrieval), it should be accessible to reproduce the accident.

## 4 Aspects of Data Management in Automotive Systems

In this section, we shed some light on implementation or design aspects of an automotive DMS. We identified three main aspects which have to be considered with care and we discuss each of them in the following.

### 4.1 Central vs. Distributed vs. Hybrid DMS

The first and maybe most important aspect, is the question where to establish the DMS within an automotive system physically. Three approaches are possible, namely *central*, *distributed* or *hybrid* DMS and we discuss their pros and cons in the following.

**Central DMS.** A central DMS means to implement the whole software system on a single ECU within the automotive system. Since the DMS has to communicate with all subbus systems and their respective ECUs, the central gateway is the only place where the central DMS can be enclosed. The major advantages of such a system are increased maintainability and extensibility. Updates or extensions (e.g., of functionality) only have to be provided for one piece of software which is situated at only one piece of hardware. Furthermore, strategies for concurrency control or other mechanisms can be handeld internally since no other DMS exists in the system and thus, optimization should be easier. Nevertheless, such a central solution comes along with significant disadvantages as well. Firstly, it is always a bottleneck regarding availability or reliability. If the data traffic is very high, the DMS may slow down and thus, propagation of data is delayed or even aborted. Moreover, if the gateway exhibits a failure or is not accessible for one or more subbus systems, this may have devastating consequences for the whole automotive systems since data is not available or out-of-date. Subsequently, the efficiency and reliability of the automotive system is decreased by such a solution. Secondly, since an ECU (even a gateway) has limited resources, a central DMS would not cope with the requirements of an automotive system (e.g., hard real time, computing power) for processing the data. We conclude, that a central DMS may be not appropriate at all for an automotive system.

**Distributed DMS.** Since this approach proposes to implement only one DMS for the whole automotive system as well, it differs from the central DMS by the fact, that the software components of the DMS are (physically) distributed over the whole system. This can be done by using the *AUTOSAR (AUTomotive Open System ARchitecture, www.autosar.org)* standardization, which has been established for automotive systems. This standard proposes a runtime environment and standardized interfaces, so that software components can be distributed over different hardware units [H+06]. With this approach we can overcome some of the disadvantages of the central DMS approach. First, distributing the DMS over all ECUs tackles the bottleneck problem which occurs in case that an ECU fails. Furthermore, some software components could be replicated on different ECUs (i.e., different from the ECU which contains the original component) and in case of failure, the replicated component can substitute the original one. Second, due to the distribution, we are not limited to the resources of only one ECU, which is also a problem of the central DMS. Beside these improvements (regarding the central DMS approach), the distributed DMS exhibits the already mentioned advantages like maintainability or extensibility as well. But in spite of all these advantages, even this approach comes along with some critical disadvantages. Because of the distribution of software components, the communication between the ECUs increases. This, in turn, leads to higher data traffic which could be critical at a certain point. Furthermore, the coordination between the particular components is quite complex, which may be enforced in the case that one ore more components (or their corresponding ECU respectively) fail. In summary, the distributed approach is yet more appropriate than the central DMS approach, although there is still room for optimizations.

**Hybrid DMS.** The last approach we want to introduce here is some kind of mixture of the previous two approaches. The idea is to establish one DMS for every subbus system. By this separation, we can adjust the several DMSs to the requirements, functionality and data traffic of the particular subbus system and thus, optimize the load balancing of the overall system. Within such a subbus system, the DMS can be distributed over an arbitrary number of ECUs and thus, benefit from the advantages of the distributed approach. Nevertheless, a single DMS has still a global view on the data of the whole automotive system regarding concurrency control or similar. For instance, if a certain piece of data is locked by the power train DMS, the DMS of another subbus system cannot write this data until it is unlocked. We think, that the hybrid approach can overcome all aforementioned disadvantages and thus, is the most appropriate solution for automotive systems.

### 4.2 Tailoring and Reusing Functionality

Once the decision for a certain kind of data management has been made, new problems regarding the implementation arise. As already mentioned, current solutions suffer from high development costs or poor maintainability. Furthermore, we have to consider the crucial requirements and constraints of an automotive system, e.g., limited memory, computing power or hard real-time requirements. To overcome these problems, the concepts of tailoring (software) and reusing (functionality) provide efficient solutions.

In the context of this paper, reusing functionality means that a DMS or parts of it can be reused instead of developing it from scratch. For instance, if we implement the hybrid DMS (cf. Section 4.1), it would be cumbersome and costly to develop the DMS for every particular subbus system from scratch. Rather, it would be eligible to develop one DMS, which could be adjusted to the requirements of the several subbus systems and thus, as much functionality as possible could be reused for every DMS. This not only leads to a noticeable decrease of development costs, but also improves the maintainability since changes to the DMS can be made at one central point (the original DMS). In addition, tailoring the DMS, i.e., the software contains only the functionality needed, can help to overcome the system requirements and constraints. For instance, if the DMS of the infotainment subbus system does not need SQL or index structures, this functionality is removed from the DMS. As a result, the DMS requires less system resources (e.g., memory, power consumption) and even the communication effort may be decreased. Further information about techniques and concepts for tailoring and reusing software can be found, e.g., in [K$^+$90, CE00, CN06].

### 4.3 Integrating IT Security Mechanisms

The third aspect we consider, covers the goals of increased safety and reliability in automotive systems. A central point for achieving these goals is to ensure the security of the systems (cf. Section 2.2). Hence, it is useful to integrate respective security mechanisms in the data management, because this is where data are handled first, before sending or after receiving. In Section 3.3, we already denoted how security can be integrated in the access management in order to ensure the integrity of the data. However, for achieving the goals, further DMS components have to be enhanced with respective security mechanisms, so that a holistic protection can be provided. Thus, appropriate components have to be identified and security mechanisms have to be adjusted so that they fit the needs of the software as well as the requirements of the underlying automotive system. As a result, we envision a kind of security layer within an automotive data management system.

## 5 Related Work

A lot of work has been done in the several fields, unified in this work. In the following we confine ourself to mention only those which are highly related to this paper.

**Automotive IT security:** In the automotive domain, holistic concepts for IT security are a very young research topic. While such concepts are absent so far, common IT security mechanisms for protecting single components, based on encryption or digital signatures, can already be found in today's cars. Some examples are central locking, keyless entry and immobiliser systems [LSS06], but also memory contents like firmware updates are protected against unauthorised manipulations. Because such existing mechanisms are not conceived to provide a holistic protection for the entire system, in the recent past research

about holistic concepts has begun. For example, the application of Trusted Computing technology in future automotive IT systems is increasingly investigated (see [BEWW07]). There are also approaches to employ PKI infrastructure and certification of automotive components to verify aspects like integrity and authenticity at every start of the car [BZ08]. In previous work, also the extension of such future automotive IT security concepts by Intrusion Detection approaches [HKD08a] has been discussed.

**Data management in embedded systems:** Tailor-made data management for embedded systems has been widely proposed in the recent years. First, there exist several approaches, dealing with whole DMS as well as with certain parts of it, e.g., the storage manager, with respect to embedded systems in general [LAS05, SRS+07, R+08]. Amongst others, they focus on motivations for tailoring a DMS and new concepts, techniques or paradigms for achieving the goal with a minimized effort. Second, some work on data management for automotive systems has been done as well. For instance, Nyström et. al. discuss a component-based approach for an efficient data management as well as general data management issues in automotive (sub)systems [NTN+02, NTNH04]. However, neither the former nor the latter adresses a holistic approach for data management in automotive systems, or even integrate IT security as important (non-functional) property.

## 6 Conclusions

Due to the high complexity in automotive systems, an efficient management of data is inevitable. In this paper, we suggested to establish a DMS for such systems to overcome the problems of current solutions. We presented different scenarios to point out the necessity of a DMS and beyond it, we proposed the idea of integrating IT security in such a DMS. Finally, we discussed three core aspects, which are crucial for the success of an automotive DMS and thus, have to be considered carefully during design and implementation.

## References


[BD07] A. Barisani and B. Daniele. Unusual Car Navigation Tricks: Injecting RDS-TMC Traffic Information Signals. In *Proc. of the CanSecWest Conf.*, 2007.

[BEWW07] A. Bogdanov, T. Eisenbarth, M. Wolf, and T. Wollinger. Trusted Computing for Automotive Systems. In *Proc. of the VDI/VW Gemeinschaftstagung Automotive Security*, pages 227–237, 2007.

[BZ08] D. Borchers and P.-M. Ziegler. Mit PKI gegen den Autoklau. Heise Newsticker, 2008. `http://www.heise.de/newsticker/meldung/104593`.

[CE00] K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley, 2000.

[CN06] P. Clements and L. Northrop. *Software Prodcut Lines: Practices and Patterns*. Addison Wesley, 2006.

[CRTM98] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a Revolution in On-Board Communications. Technical report, Volvo Technologies, 1998.

[H+06] Harald Heinecke et al. Achievements and exploitation of the AUTOSAR development partnership, 2006. http://www.autosar.org/download/AUTOSAR_Paper_Convergence_2006.pdf.

[HD07] T. Hoppe and J. Dittmann. Sniffing/Replay Attacks on CAN Buses: A Simulated Attack on the Electric Window Lift Classified using an adapted CERT Taxonomy. In *Proc. of the Workshop on Embedded Systems Security (WESS) at EMSOFT 2007*, 2007.

[HKD08a] T. Hoppe, S. Kiltz, and J. Dittmann. IDS als zukünftige Ergänzung automotiver IT-Sicherheit. In *Proc. of DACH Security*, 2008.

[HKD08b] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive CAN networks practical examples and selected short-term countermeasures. In *Proc. of the Int'l Conf. on Computer Safety, Reliability and Security (SAFECOMP)*, pages 235–248, 2008.

[HKLD07] T. Hoppe, S. Kiltz, A. Lang, and J. Dittmann. Exemplary Automotive Attack Scenarios: Trojan horses for Electronic Throttle Control System (ETC) and replay attacks on the power window system. In *Proc. of the 23. VDI/VW Gemeinschaftstagung Automotive Security*, pages 165–183, 2007.

[K+90] K.C. Kang et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[LAS05] T. Leich, S. Apel, and G. Saake. Using Step-Wise Refinement to Build a Flexible Lightweight Storage Manager. In *Proc. of the East-European Conf. on Advances in Databases and Information Systems (ADBIS)*, 2005.

[LSS06] Kerstin Lemke, Ahmad-Reza Sadeghi, and Christian Stüble. Anti-theft Protection: Electronic Immobilizers. In *Embedded Security in Cars*, 2006.

[NTN+02] D. Nyström, A. Tesanovic, C. Norström, J. Hansson, and N.-E. Bankestad. Data Management Issues in Vehicle Control Systems: A Case Study. In *Proc. of Euromicro Conf. on Real-Time Systems*, pages 249–256, 2002.

[NTNH04] D. Nyström, A. Tesanovic, C. Norström, and J. Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proc. of the Workshop on Soft. Eng. for Automotive Systems at the ICSE*, 2004.

[Paa08] Cristof Paar. Remote keyless entry system for cars and buildings is hacked, 2008. http://www.crypto.rub.de/imperia/md/content/projects/keeloq/keeloq_en.pdf.

[PBKS07] A. Pretschner, M. Broy, I.H. Krüger, and Th. Stauner. Software Engineering for Automotive Systems: A Roadmap. In *Proc. of Future of Soft. Eng. (FOSE) at the ICSE*, pages 55–71, 2007.

[R+08] M. Rosenmüller et al. FAME-DBMS: Tailor-made Data Management Solutions for Embedded Systems. In *Proc. of the Workshop on Soft. Eng. for Tailor-made Data Management (SETMDM)*, 2008.

[SRS+07] G. Saake, M. Rosenmüller, N. Siegmund, C. Kästner, and Thomas Leich. Downsizing Data Management for Embedded Systems. In *Keynote of the Int'l Conf. on Information Technology*, November 2007.

[ZS07] W. Zimmermann and R. Schmidgall. *Bussysteme in der Fahrzeugtechnik*. 3.Auflage. Vieweg + Teubner, 2007.

# Technische Grundlagen für eine laufzeitadaptierbare Transaktionsverwaltung

Florian Irmert, Christoph P. Neumann, Michael Daum,
Niko Pollner, Klaus Meyer-Wegener

Lehrstuhl für Informatik 6 (Datenmanagement), Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg
{florian.irmert, christoph.neumann, md, kmw}@informatik.uni-erlangen.de

**Abstract:**
Auf spezielle Anwendungen zugeschnitte Datenbankverwaltungssysteme (DBMS) erfordern in den meisten Fällen Individuallösungen, welche einen erhöhten Entwicklungs- und Wartungsaufwand mit sich bringen. Es fehlen modulare DBMS-Architekturen als konfigurierbare Standardlösung, die beliebig schlanke als auch umfangreiche Architekturen ermöglichen. Die Transaktionsverwaltung als querschneidender Belang durchdringt in herkömmlichen DBMS die Architektur und verhindert die Modularisierung. In dieser Arbeit wird die Transaktionsverwaltung aus einem DBMS herausgelöst und durch eine eigenständige, wiederverwendbare Komponente realisiert. Die Architektur der gekapselten und wiederverwendbaren Transaktionsverwaltung wird dokumentiert, sowie die aspektorientierte Verknüpfung zwischen reduziertem DBMS und separater Transaktionsverwaltung beschrieben. Als Grundvoraussetzung für zukünftige, sich selbstverwaltende DBMS kann die Transaktionsverwaltung zur Laufzeit dynamisch hinzugefügt und entfernt werden. Die prototypische Implementierung wird an Hand von SimpleDB in Verbindung mit einem dynamischen AOP Framework erläutert.

## 1 Einleitung

Existierende Datenbanksysteme sind in der Regel monolithische Systeme, die nicht für jedes Anwendungsgebiet optimal geeignet sind. Um Datenbanken bspw. in eingebetteten Systemen einzusetzen, ist ein möglichst geringes Speicherprofil von entscheidender Relevanz. Hierfür optimierte Speziallösungen haben jedoch einen erhöhten Entwicklungs- und Wartungsaufwand zur Folge. Eine nachhaltige Lösung muss darin bestehen, die Anpassungsfähigkeit und die schlanke Architektur einer möglichen Individualentwicklung mit dem umfassenden Funktionsangebot traditioneller Datenbanksysteme zu vereinen.

In eingebetteten Systemen ist oft auf Basis einer spezifischen Anwendungssemantik oder wegen einfachen und leicht zu synchronisierenden Anwendungen die Transaktionsverwaltung (TAV) ungenutzter Ballast. Ein schlankes Datenbankverwaltungssystem (DBMS) ohne TAV mit erhöhtem Durchsatz und besserem Antwortzeitverhalten ist für dieses Anwendungsgebiet optimal. Da nicht auszuschließen ist, dass das Anwendungsgebiet in seiner Komplexität mit der Zeit derart zunimmt, dass später eine TAV benötigt wird, soll die

Ergänzung des DBMS um den Transaktionsaspekt möglichst nahtlos stattfinden können.

Eine Herausforderung ist die Modifikation eines modularen Datenbanksystems zur Laufzeit. Gray postulierte „if every file system, every disk and every piece of smart dust has a database inside, database systems will have to be self-managing, self-organizing, and self-healing“ [Gra04]. Der Bedarf für dynamisches Austauschen und Aktualisieren von Modulen zur Laufzeit als relevante Forschungsaufgabe wurde bereits in den 70er und 80er Jahren erkannt [Fab76, SF89] und ist bis heute eine Herausforderung. In unserem Kontext bedeutet dies die Ergänzung des DBMS auf dem Endgerät um die TAV ohne eine erneute Auslieferung des DBMS und ohne einen damit verbundenen Neustart.

Das Projekt CoBRA DB (**Co**mponent **B**ased **R**untime **A**daptable **D**ata**B**ase) stellt eine moderne Referenzmodularisierung zur Verfügung [IDMW08]. Insbesondere die architektonische Kapselung der TAV stellt eine der größten Herausforderungen bei der Modularisierung eines DBMS dar, da die TAV ein rein konzeptionellen Systembaustein ist, der als querschneidender Belang viele Systembausteine eines DBMS durchdringt.

Ziel dieser Arbeit ist es, eine Referenzarchitektur und prototypische Implementierung vorzustellen, in der die TAV als streng eigenständiger Aspekt gekapselt ist, so dass sie auch in anderen DBS-Entwicklungen als Baustein eingesetzt werden kann. Zur Realisierung wird dynamische aspektorientierte Programmierung (d-AOP) [CS04] verwendet, eine AOP-Infrastruktur mit Unterstützung von Modifikationen zur Laufzeit. Die Modularisierung erfolgt im vorgestellten Prototypen auf Basis der SimpleDB [Sci07], die in die klassischen Schichten eines relationalen DBMS nach [Här05] gegliedert ist.

## 2 Verwandte Arbeiten

In diesem Kapitel wird dynamische AOP (d-AOP) kurz vorgestellt sowie andere Arbeiten im Bereich der modularen DBMS abgegrenzt.

### 2.1 Dynamische aspektorientierte Programmierung

Das Einweben von Code eines Aspekts, sog. Advices, auf Basis von Pointcuts kann grundsätzlich zu drei verschiedenen Zeitpunkten geschehen: Beim Übersetzungsvorgang (*compile time weaving*), zum Ladezeitpunkt der Klasse (*load time weaving*) oder zur Laufzeit (*runtime weaving*). Kann Aspektcode nach dem ersten Einweben verändert werden, wird von *dynamic AOP* gesprochen. In dieser Arbeit wird *JBoss AOP* verwendet, bei dem zur Ladezeit an denjenigen Stellen, an denen zur Laufzeit Advices ausgeführt werden können, sogenannte Hooks eingewebt werden [CS04].

### 2.2 Modulare Datenbankverwaltungssysteme

Vor- und Nachteile aspektorientierter Methoden in der Entwicklung von DBMS werden in [TSH04] diskutiert. Dabei liegt beim Refactoring des modularen DBMS *Berkeley DB* unter Verwendung von AOP der Schwerpunkt auf Wartbarkeit. *Berkeley DB* dient ebenfalls als Grundlage des in [Käs07] vorgestellten aspektorientierten Refactorings mittels *AspectJ*. Die Umgestaltung konzentriert sich nicht auf die TAV, sondern auf feingranularere Systemteile, da in erster Linie die Anwendbarkeit von AOP zur Implementierung von Features diskutiert wird.

In [PLKR07] wird die Entwicklung eines konfigurierbaren Transaktionsmanagements vorgestellt. Für die Umsetzung kamen *Aspectual Mixin Layers* zum Einsatz, eine Verbindung aus AOP und FOP (Feature Oriented Programming). Schwerpunkt bildet die Konfigurierung auf Basis identifizierter Merkmale innerhalb des Transaktionsmanagements. Im Rahmen des *COMET DBMS*-Projekts [NTNH03] erfolgt die Entwicklung eines komponentenbasierten DBMS unter Verwendung von aspektorientierter Programmierung.

Die vorliegende Arbeit unterscheidet sich von den oben genannten vor allem in zwei Bereichen: Zum einen konzentriert sie sich auf das Gebiet der Transaktionssicherung und gibt einen Architekturvorschlag für deren DBMS-unabhängige Implementierung. Zum anderen steht bei diesem Design die Möglichkeit des Hinzufügens und Entfernens des für die Transaktionssicherung zuständigen Codes zur Laufzeit im Vordergrund.

## 3 Anforderungen

Im Folgenden werden die grundsätzlichen Anforderungen an ein DBMS beschrieben, dessen TAV mit Hilfe von d-AOP angebunden werden soll. Des Weiteren wird begründet, dass Schlüsselwörter der Transaktionsunterstützung auch ohne Existenz der TAV vorbereitend durch die DBS-Nutzer und -Programmierer in SQL verwendet werden müssen.

### 3.1 TAV-Unabhängigkeit des DBMS

Es gibt mehrere Gründe, die eine Forderung nach Unabhängigkeit des DBMS von der TAV bedingen: 1) Die TAV soll nur bei Bedarf hinzugefügt werden und auf Wunsch auch wieder entfernt werden können. 2) Die Entwicklung und die Wartung des restlichen Systems werden durch direkte Berücksichtigung der TAV stark erschwert, da sich diese quer durch alle Schichten zieht. Bei einer nachträglichen Änderung an der TAV sind somit möglicherweise Änderungen an vielen Klassen notwendig, die keinen direkten Bezug zur Aufgabe der Transaktionssicherung haben.

### 3.2 DBMS-Unabhängigkeit der TAV

Auch in Bezug der Abhängigkeit der TAV vom DBMS ist aufgrund verschiedener Forderungen eine möglichst lose Kopplung anzustreben: 1) Änderungen am zugrundeliegenden System sollen nur möglichst wenige Änderungen an der TAV nach sich ziehen. 2) Der mögliche Einsatz einer einmal entwickelten TAV in verschiedenen Datenbanksystemen verringert sowohl den Entwicklungs- als auch Wartungsaufwand. 3) Bei einem konfigurierbaren System können die Implementierungen der einzelnen Schichten nach Bedarf ausgetauscht werden. Es ist deshalb anzustreben, die TAV so zu entwerfen, dass sie von einem solchen Austausch möglichst wenig betroffen ist.

Im Gegensatz zur TAV-Unabhängigkeit lässt sich DBMS-Unabhängigkeit nur bis zu einem gewissen Grad erreichen. Die Anbindung ist zwangsläufig spezifisch für ein bestimmtes System. Zur Durchführung seiner Aufgaben benötigt insbesondere das für Recovery zuständige Teilsystem Zugriff auf interne Strukturen der Datenbank. Die Wartbarkeit lässt sich hier mit Hilfe einer günstigen Architektur weitestgehend erhalten, indem der für die Anbindung bedeutsame Code in wenigen Klassen gekapselt und von der eigentlichen Logik der TAV getrennt wird.

### 3.3 Vorbereitende Verwendung transaktionaler Schlüsselwörter

Aus Sicht des DBS-Nutzers muss stets Isolation gewährleistet sein. Im Einbenutzerbetrieb gilt dies immer, für Mehrbenutzerbetrieb wird im Allgemeinen eine TAV benötigt. Mehrbenutzerbetrieb ohne TAV ist möglich, wenn die Anwendungen per Anwendungssemantik isoliert sind.

Für den DBS-Nutzer und -Programmierer beginnt mit jeder DBS-Verbindung eine logische Folge von Operationen; setzt er explizit SQL-Schlüsselwörter der Transaktionsunterstützung ein, beginnt für ihn anschließend eine neue logische Folge. Auf Systemseite wird dabei eine Transaktion beim Aufbau einer Verbindung und nach jedem *commit* und *abort* begonnen, entsprechend der *SQL*-Norm und dem Standardverhalten der *JDBC*-API. Ohne TAV werden mit Schlüsselwörtern der Transaktionsunterstützung nur logische Abschnitte markiert: Ein *commit* stellt ohne TAV insbesondere die Dauerhaftigkeit nicht notwendigerweise sicher, ein *abort* ist ohne TAV nicht möglich und dem Anwender muss ein Fehler mitgeteilt werden.

Im Allgemeinen kann die Datenbankzugriffsschicht auf Anwendungsseite zur Laufzeit nicht geändert werden. Da das DBMS die TAV zumindest potentiell in der Zukunft zur Verfügung stellt, müssen die Benutzer und Programmierer ihre logischen Operationsfolgen innerhalb von Sitzungen von Beginn an für den allgemeinen Mehrbenutzerbetrieb vorbereiten, indem sie *commit* verwenden. Verzichten sie darauf, muss ihnen klar sein, dass später ein DBMS mit TAV aus einer Sitzung eine einzige Transaktion ableiten wird. Besonders in eingebetteten Systemen laufen Anwendungen und ihre Sitzungen oftmals ohne Terminierung und würden in nicht-terminierenden Transaktionen über die Zeit Sperren akkumulieren, die niemals freigegeben werden. Als Vorbereitung einer dynamischen

Zuschaltung der TAV ist daher der Einsatz von *commit*-Anweisungen zur Strukturierung in logische Folgen notwendig.

## 4 Architektonische Kapselung der TAV

In der ursprünglichen SimpleDB wird die Ablaufverfolgung mit Hilfe eines Objekts vom Typ `Transaction` realisiert, das eine einzelne Transaktion repräsentiert. Diese Instanz wird, beginnend bei einer in der obersten Schicht eingehenden Anforderung, bei jedem Methodenaufruf als Parameter weitergereicht und durchläuft so alle Schichten, wodurch der Transaktionskontext überall verfügbar ist.

Zur Transaktionssicherung werden bei Zugriffen auf den Systempuffer, statt direkter Aufrufe an diesem, entsprechende Methoden des `Transaction`-Objekts verwendet, welche die zur Sicherstellung der Transaktionseigenschaften notwendigen Maßnahmen durchführen. Durch diese Technik entsteht eine feste Kopplung zwischen allen Klassen, die Zugriff auf den Systempuffer benötigen, und der TAV. Im Prototyp wurde zunächst jeglicher Transaktions-Code manuell aus der SimpleDB entfernt. Das Package-Diagramm in Abbildung 1 gibt einen Überblick über die Architektur der entwickelten TAV.

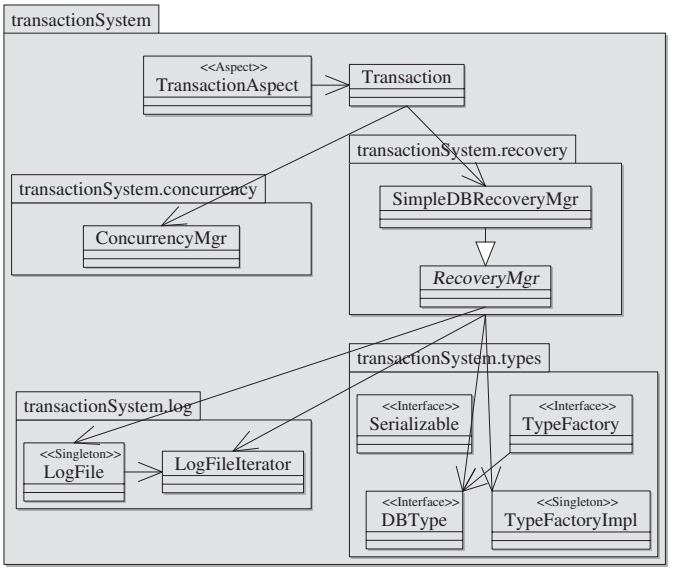



Abbildung 1: Überblick über die Architektur der TAV

### 4.1 Anbindung der TAV an das Grundsystem

Die äußere Schnittstelle der TAV, über die sie an die Vorgänge im zugrundeliegenden DBMS mittels Methoden wie z.B. `commit()`, `rollback()`, usw. angekoppelt wird, ist in der Klasse `Transaction` realisiert. Eine Instanz repräsentiert dabei genau eine Transaktion.

Die Klasse `TransactionAspect` sorgt für die Anbindung der Objekte vom Typ `Transaction`. Sie ist DBMS-abhängig und enthält Advices, die an allen relevanten Stellen des grundlegenden Systems die Methoden der betroffenen `Transaction`-Objekte aufrufen.

### 4.2 Innere Architektur der TAV

Die Klasse `LogFile` stellt die für das Recovery-System benötigten Funktionen zum Logging bereit. Sie bietet Methoden zum Schreiben von allgemeinen Daten in eine serielle Datei und unterstützt über einen entsprechenden Iterator das Auslesen. Da diese Klasse keine Annahmen über den Inhalt oder interne Strukturen macht, ist sie universell einsetzbar, unabhängig von der konkreten Ausprägung des Recovery-Mechanismus.

Die Klassen im Package `concurrency` stellen den Concurrency-Manager der TAV dar. Als Schnittstelle zur Klasse `Transaction` dient `ConcurrencyMgr`, von dem jeweils eine Instanz für eine Transaktion zuständig ist. Die Implementierung ist unabhängig von dem zugrundeliegenden DBMS. Das Sperrverfahren kann als Teilmodul ausgetauscht werden.

Im Package `recovery` befinden sich die für das Recovery Management zuständigen Klassen. Da der Recovery-Manager auf die Speicherstrukturen des DBMS zugreifen muss, um Logging und Recovery durchführen zu können, wurde das *Template Method Pattern* eingesetzt, um eine Unabhängigkeit der grundlegenden Recovery-Prozedur zu erreichen. Die Recovery-Strategie ist in der prototypischen Realisierung eine reine Undo-Recovery und kann als Teilmodul durch eine Redo- und Undo-Recovery ausgetauscht werden.

Um eine generische Repräsentation von Daten zu ermöglichen, unabhängig von den im zugrundeliegenden System hierfür eingesetzten Typen, wurde eine Hierarchie spezieller Klassen im Package `types` entworfen. `Serializable` stellt eine einfache Schnittstelle für Klassen dar, die eine Serialisierung und Deserialisierung unterstützt. Klassen, die einen Datentyp des zugrundeliegenden DBMS repräsentieren, implementieren die Schnittstelle `DBType`, die neben den Methoden von `Serializable` noch die Methode `getTypeId()` besitzt. Über diese kann eine Identifikationsnummer abgerufen werden, die für einen bestimmten Datentyp eindeutig ist. In der Logdatei können so die Identifikationsnummer des Datentyps und danach die Daten in serialisierter Form gespeichert werden.

### 4.3 Verfolgung der Transaktionen

Dynamische AOP bietet keine Möglichkeit, den Kontrollfluss innerhalb einer Anwendung zu überwachen. Die Verfügbarkeit von Kontrollflussinformation ist aber bei der Realisierung einer TAV unabdingbar, da die Aufrufe in den einzelnen Komponenten der jeweiligen Transaktion zugeordnet werden müssen.

In der ursprünglichen SimpleDB wurde der Transaktionskontext im `Transaction`-Objekt durch alle Schnittstellen gereicht, was bei einer echten Trennung der TAV vom DBMS eliminiert werden muss. Die grundlegende Problematik, einen transaktionalen Kontext zu bestimmen ohne eine enge Kopplung der TAV mit dem DBMS zu verursachen, wurde bereits in [Käs07] diskutiert, jedoch ohne eine für die Zwecke der vorliegenden Arbeit zufriedenstellende Lösung aufzuzeigen.

Eine Möglichkeit zur Verfolgung des Kontrollflusses stellen die Threads des Systems dar, da diese den Kontrollfluss repräsentieren. Die Ermittlung des transaktionalen Kontexts durch die TAV kann mit Hilfe von Tabellen realisiert werden, welche die Zuordnungen zwischen JDBC-Connection, SQL-Statement, Transaktion und Thread verwalten. Die Tabellen werden durch die TAV vollkommen autonom verwaltet, die notwendigen Advices werden in das DBMS eingebracht.

Der Mechanismus der Transaktionskontextermittlung wird in Abbildung 2 dargestellt. Grundlage des Beispiels ist die Ausführung einer SQL Update-Anweisung. Die mit *AOP* gekennzeichneten Aufrufe sind aspektorientierte Aufrufe von Advices der TAV, die durch entsprechende Pointcutdefinitionen ausgelöst werden. Die Sequenz beginnt mit dem Aufruf von `createStatement()` durch den Client, mit dem Ziel eine Referenz auf ein `RemoteStatementImpl`-Objekt zu erhalten. Das `RemoteConnectionImpl`-Objekt repräsentiert die bereits bestehende Verbindung.

An der mit *1* bezeichneten Stelle wird vor der Rückkehr des `createStatement`-Aufrufs die Zuordnung der neuen `RemoteStatementImpl`-Instanz zum `RemoteConnectionImpl`-Objekt in einer *statement-connection* Tabelle abgelegt. Um ein Update durchzuführen, ruft der Client im Beispiel die Methode `executeUpdate()` auf.

Bei Stelle *2* wird über den in *1* erzeugten Tabelleneintrag die zugehörige Verbindung ermittelt und anhand einer weiteren *connection-transaction* Tabelle geprüft, ob für diese Verbindung gerade eine Transaktion läuft. Im Rahmen dieses Beispiels enthält die Tabelle noch keinen Eintrag. Daher wird an Stelle *2* innerhalb der TAV ein neues `Transaction`-Objekt erzeugt und somit eine neue Transaktion gestartet. Anschließend wird die *connection-transaction* Tabelle um die Zuordnung zwischen Verbindung und neuer Transaktion ergänzt. Außerdem erfolgt ein Eintrag in einer *thread-transaction* Tabelle, der dem aktuellen Thread die ermittelte bzw. neu erzeugte Transaktion zuordnet.

Die drei verwendeten Verwaltungstabellen haben eine unterschiedliche Intention: Die *thread-transaction* Tabelle liefert direkt und effizient bei allen folgenden Satzoperationen des ausgeführten SQL-Statements den transaktionalen Kontext. Die *connection-transaction* Tabelle und *statement-connection* Tabelle werden bei neu eintreffenden SQL-Statements benötigt, um einen neuen Thread einer bereits laufenden Transaktion zuzuordnen.

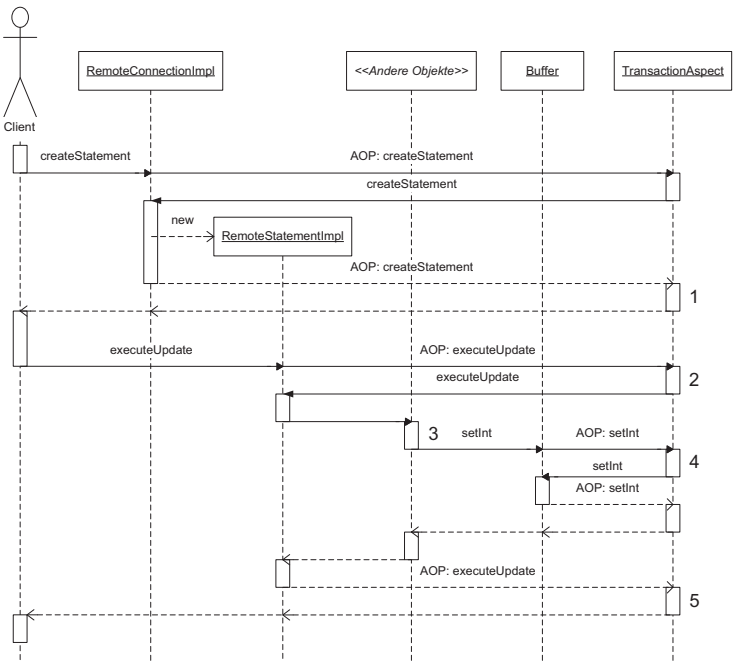



Abbildung 2: Sequenzdiagramm der Transaktionsverfolgung (leicht vereinfacht)

Nach dem Anlegen der Verwaltungsinformation veranlasst der Advice die Ausführung der eigentlichen `executeUpdate()`-Methode. Sie wird in ihrem Verlauf Satzoperationen aufrufen (Stelle *3*), von denen letztendlich ein Aufruf an den Systempuffer (`Buffer`) abgesetzt wird, der Maßnahmen durch die TAV erfordert. Im Diagramm ist dies beispielhaft für `setInt()` dargestellt: Bei Stelle *4* wird über die *thread-transaction* Tabelle das für diesen Aufruf passende `Transaction`-Objekt ermittelt. Die TAV veranlasst über dieses Objekt die notwendigen Maßnahmen wie Sperranforderung und Logging.

An der Stelle *5* schließlich wird vor der Rückkehr des `executeUpdate()`-Aufrufs an den Client die Zuordnung von Thread zu Transaktion wieder aus der entsprechenden Tabelle entfernt.

## 5 Zusammenfassung

In dieser Arbeit wurde die prototypische Implementierung einer mittels d-AOP angebundenen TAV vorgestellt. Als Grundlage diente ein um die TAV reduziertes schlankes DBMS auf Basis der SimpleDB, deren Funktionsumfang vollständig erhalten wurde. Zielumgebungen sind eingebettete Systeme, in denen einerseits ein schlankes Speicherprofil der Softwarebausteine besondere Relevanz hat, aber andererseits die Anpassbarkeit an eine

zukünftige Anwendungsumgebung gefordert wird.

Die TAV wurde als autonomes und DBMS-unabhängiges Modul realisiert, das mit Hilfe eines Transaktionsaspekts an das DBMS angeschlossen ist. Die beschriebene Architektur optimiert die DBMS-Unabhängigkeit der TAV, indem die DBMS-abhängigen Anteile ausschließlich in Form von Pointcuts und Advices repräsentiert sind, als Anschlussstellen an das DBMS. Abhängigkeiten der TAV-Subsysteme für Synchronisation, Logging und Recovery zum zugrunde liegenden DBMS wurden auf ein Minimum reduziert. Der Einsatz eines d-AOP Frameworks ermöglicht es, das Hinzu- und Abschalten der TAV zur Laufzeit vorzunehmen.

Das übergeordnete Ziel ist es, eine modulare DBMS-Architektur und einen „DBS-Baukasten“ zu gewinnen, der umfassende Anpassungsfähigkeit an beliebige Anwendungsumgebungen unterstützt und dabei sowohl beliebig schlanke Architekturen als auch umfangreiche Architekturen ermöglicht. Diesem Ziel sind wir um den autonomen Baustein der Transaktionsverwaltung nähergekommen. Zukünftige Aufgabe ist die exakte Definition der transaktionalen Semantik sowie die Realisierung einer Zustandsmigration bei Adaption der TAV zur Laufzeit.

## Literatur


[CS04] R. Chitchyan und I. Sommerville. Comparing Dynamic AO Systems. In *Dynamic Aspects Workshop (DAW 2004)*, Lancaster, England, März 2004.

[Fab76] R. S. Fabry. How to design a system in which modules can be changed on the fly. In *2nd International Conference on Software Engineering (ICSE 1976:)*, Seiten 470–476, Los Alamitos, CA, USA, Oktober 1976. IEEE Computer Society Press.

[Gra04] Jim Gray. The next database revolution. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, Seiten 1–4, New York, NY, USA, Juni 2004. ACM.

[Här05] Theo Härder. DBMS Architecture - Still an Open Problem. In Gottfried Vossen, Frank Leymann, Peter C. Lockemann und Wolffried Stucky, Hrsg., *11. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW 2005)*, Jgg. 65 of *LNI*, Seiten 2–28. GI, März 2005.

[IDMW08] Florian Irmert, Michael Daum und Klaus Meyer-Wegener. A New Approach to Modular Database Systems. In *Software Engineering for Tailor-made Data Management (SETMDM 2008)*, Seiten 41–45, März 2008.

[Käs07] Christian Kästner. Aspect-Oriented Refactoring of Berkeley DB. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, School of Computer Science, Department of Technical and Business Information Systems, Februar 2007.

[NTNH03] Dag Nyström, Aleksandra Tesanovic, Christer Norström und Jörgen Hansson. The COMET Database Management System. MRTC report ISSN 1404-3041 ISRN MDH-MRTC-98/2003-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, April 2003.

[PLKR07] Mario Pukall, Thomas Leich, Martin Kuhlemann und Marko Rosenmueller. Highly configurable transaction management for embedded systems. In *6th workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS 2007)*, Seite 8, New York, NY, USA, März 2007. ACM.

[Sci07] Edward Sciore. SimpleDB: A Simple Java-Based Multiuser System for Teaching Database Internals. In *38th ACM Technical Symposium on Computer Science Education (SIGCSE 2007)*, Seiten 561–565, New York, NY, USA, März 2007. ACM.

[SF89] M. E. Segal und O. Frieder. Dynamic program updating: a software maintenance technique for minimizing software downtime. *Journal of Software Maintenance: Research and Practice*, 1(1):59–79, September 1989.

[TSH04] Aleksandra Tešanović, Ke Sheng und Jörgen Hansson. Application-Tailored Database Systems: A Case of Aspects in an Embedded Database. In *8th International Database Engineering and Applications Symposium (IDEAS 2004)*, Seiten 291–301, Washington, DC, USA, Juli 2004. IEEE Computer Society.

# Towards web supported identification of top affiliations from scholarly papers

David Aumueller

University of Leipzig
david@informatik.uni-leipzig.de

**Abstract:** Frequent successful publications by specific institutions are indicators for identifying outstanding centres of research. This institution data are present in scholarly papers as the authors' affilations – often in very heterogeneous variants for the same institution across publications. Thus, matching is needed to identify the denoted real world institutions and locations. We introduce an approximate string metric that handles acronyms and abbreviations. Our URL overlap similarity measure is based on comparing the result sets of web searches. Evaluations on affiliation strings of a conference prove better results than soft tf/idf, trigram, and levenshtein. Incorporating the aligned affiliations we present top institutions and countries for the last 10 years of SIGMOD.

## 1 Introduction

What are the most important research institutions in a specific field? Where do publications to conferences and journals come from? We want to answer such questions by analysing scholarly papers. Affiliations are stated in various forms denoting the same real world institution. Thus, to be able to aggregate papers by single real word entities, the different variants need to be matched. Affiliation variants include acronyms, abbreviations and multiple long forms, making it hard for common approximate string metrics such as edit-distance and trigram. We propose a web-based approach and **introduce the URL overlap similarity metric**. Each string is queried against a web search engine to collect the result set containing URLs to relevant pages. Basically we argue, the more URLs overlap across two result sets, the more likely the two query strings can be treated as synonyms. We assign countries to identified affiliations to depict their locations on a map, e.g. with more publications highlighted in different colour (more in shades from blue to red as in figure 1).

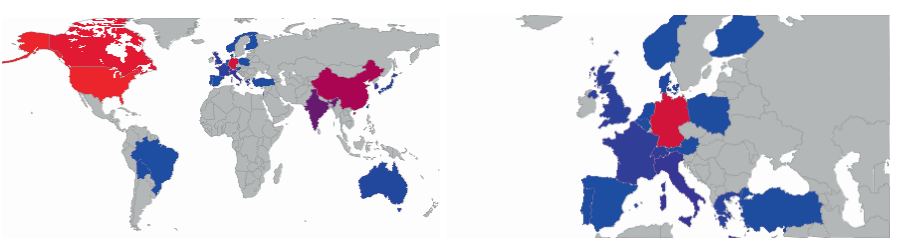


Figure 1: Maps of the world and Europe coloured by no. of 10 years' SIGMOD papers

Other motivating use of this data is the incorporation of an institution dimension as well as a geographic dimension in citation analyses, cf. [RT05], and online bibliographies such as collected using Caravela [AR07].

The next section outlines the general approach to identify outstanding institutions, in section 3 we present and evaluate our URL overlap metric for matching heterogeneous strings. Section 4 lists results of initial analyses of a conference series building on the data of section 3. We close by presenting related work and a summary with future work.

## 2 General approach to identify outstanding institutions

Authors of scholarly publications state their affiliations in the papers. As the ACM digital library website has these affiliations listed to each author we extracted the affiliation strings thereof (but are developing heuristics to extract such data from PDF fulltexts as well). The strings often contain departments and other details. We did no thorough pre-processing but left the strings as is except for cutting off the major variants of departments (suffixed by a colon) via a regular expression.

The next step is to determine affiliation strings denoting the same (real world) institution. We use our new approximate string measure (presented in the next section) to establish the mapping of corresponding affiliation strings. Groups of affiliation strings need to be clustered, e.g. by putting all strings into one group whose matches have a higher similarity score than a given threshold.

To analyse the data by geographical aspects, countries and possibly cities or latitude/longitude need to be linked to the institutions. We currently experiment with a large database of city names to establish such a mapping by matching city mentions. To disambiguate city names we iteratively decrease specificity: "*city, state, USA*", "*city, state*", "*city, country*", and "*city*" are used as patterns. Context in the fulltext of the papers may hint to determine the country as well, e.g. email addresses. For the results presented in section 4 we checked and completed the country information manually.

## 3 Approximate matching of affiliation string variants

Affiliation matching is a challenging task due to the many variants affiliations are stated not only on scholarly papers. The heterogeneity mostly derives from abbreviations of varying degree, ranging from ambiguous acronyms to self-explanatory abbreviations such as '*Univ.*' for '*University*'. One of the more challenging examples is the "*University of California at Santa Cruz, California*" of which the shortest form reads "*UCSC*", a medium short form "*UC Santa Cruz, Santa Cruz, CA*", and lots more are around. Common string metrics for matching two string variants include Levenshtein edit-distance, n-gram comparison, and soft tf/idf. For an overview and comparison of string metrics see e.g. [CRF03]. Applied to affiliations these work well within limits. These metrics do not help in mapping acronyms to long forms, i.e. they do miss a lot of correspondences such as stated above. In Wikipedia, mappings between varying

affiliation names are established for at least the well-known ones via numerous redirects pointing to the according article. One could extract and use this data, but scholarly papers and other sources contain more affiliations and variants than covered in Wikipedia.

### 3.1 The URL overlap string similarity metric

To overcome these drawbacks we propose a web-based approach that does not primarily take into account the overlap in the syntactic representation of the strings but compares the overlap of the search engine results queried for these strings. Here we concentrate especially on the overlap of identical URLs in the result set.

Hypothesis: The more URLs overlap the more similar the concepts behind the strings.

We use the complete URL for quantifying URL overlap and calculating similarity. We considered taking only parts of the URLs into account, e.g. domain or hostname similar to [Ta08], but web sites like Wikipedia are contained within the search result of many affiliation strings leading to identical hostnames that are not discriminating the search terms. Normalizing the URL overlap within the range 0..1 could be simply achieved by dividing the URL overlap by the maximum number φ of retrieved URLs, i.e. $sim_1$ = overlap / φ. We also take the ranks of the first overlapping hit into account and add that to $sim_1$ with a weighting factor (for examples of correspondences see table 1, next page):

$sim_{URLoverlap} = [ \alpha \cdot ( overlap / \varphi ) + \beta / ( 1 + \delta ) ] / ( \alpha + \beta )$, whereby …

overlap: number of overlapping URLs,
α, β: weighting factors, e.g. 2 and 1,
δ: distance between min ranks of overlapping URL, i.e. min(rank of URL u in result set for query str1) – min(rank of URL u in result set for query str2),
φ: number of retrieved search results per query, usually 10, 50, or 100.

We experimented with a URL overlap similarity metric that integrates also the ranks of all overlapping URLs with larger weights for higher ranks but could not detect more discriminating power.

### 3.2 Experiments and observations

To test our approach we collected over 4000 unique affiliation strings from database conferences covered in the ACM digital library. For each string we issue a web search query and store the maximum of returned results per one call into a relational database. With using the current BOSS service provided by Yahoo there is no restriction on the number of calls per timeslot. One call returns up to 50 hits, though. Each query result contains a projected number of overall hits and each hit entry contains (among others) rank, URL, size, date, and a snippet. To create the whole mapping of correspondences determined via URL overlap similarity a single SQL query suffices. A self-join on the web search result table as r and s using r.URL=s.URL aggregated by r.string and s.string

produces as group count the number of overlapping (identical) URLs. One could further specify a threshold in the having clause of the aggregation to limit the size of the resulting mapping. As the approach is relatively restrictive in nature we did not use a threshold here:

```
insert into mapping
select r.id, s.id, r.string, s.string, count(*) as overlap,
(2*overlap/50) + 1/1+abs(min(r.rank)-min(s.rank)))/3 as sim,
from result r, result s
where r.url=s.url and r.id<>.s.id
group by r.id, s.id
```

We experimented with levenshtein, trigram, soft tf/idf and URL overlap. We run soft tf/idf in the combination of a Jaro Winkler distance for the tokenized affiliation strings. The tf/idf-part takes care of scoring more frequent tokens lesser, which serves well for discriminating e.g. the lot of "*University of ...*" strings. Each set misses different potentially correct correspondences which would speak for a combination of similarity metrics.

| | | | |
|---|---|---|---|
| URL overlap | | | |
| 11, sim=0.48 | Carnegie-Mellon University, Pittsburgh, Pa. | CMU, Pittsburgh, PA | true |
| 10, sim=0.47 | University of California, San Diego | UCSD | true |
| 22, sim=0.46 | University of Illinois, Urbana-Champaign, IL | UIUC, Urbana, IL | true |
| 8, sim=0.44 | Massachusetts Institute of Technology, Cambridge, Mass. | MIT, Cambridge, MA | true |
| 5, sim=0.40 | AT&T Labs-Research, New Jersey, NJ, USA | Bell Labs Research | true |
| 6, sim=0.22 | Dresden University of Technology, Dresden, Germany | TU Dresden | true |
| 6, sim=0.11 | Max-Planck Institute for Informatics, Saarbrücken, Germany | Saarland University, Saarbrücken, Germany | false |
| 1, sim=0.02 | University of Illinois at Chicago, Chicago, Illinois | University of Chicago | false |
| Levenshtein | | | |
| $sim_l$=0.69 | Università di Torino, Torino, Italy | U. of Torino, Torino, Italy | true |
| $sim_l$=0.70 | Michigan State University, Lansing, MI | Wichita State University, Wichita, KS | false |
| Trigram | | | |
| $sim_t$=0.50 | Microsoft Corporation, Redmond, WA | Microsoft Research, Redmond, WA | true |
| $sim_t$=0.61 | University of Illinois at Chicago | University of Chicago | false |
| Soft tf/idf | | | |
| $sim_s$=0.77 | State Univ. of New York, Stony Brook | Stony Brook University | true |
| $sim_s$=0.37 | Tsinghua University, Beijing, China | Microsoft Research Asia, Beijing, China | false |

Table 1: Examples of true and false positives found by various sim metrics

The distance (difference) between the highest ranks of an overlapping URL supplies a further parameter in the score calculation, as hinted by the false[1] positive result of "*University of Illinois at Chicago, Chicago, Illinois*" ($rank_{highest}$=44) and "*University of Chicago*" ($rank_{highest}$=1), i.e. hit 44 for string 1 is the best whose URL is also in the result set for string 2, whereas already the URL of the first hit for string 2 is also within the result set for string 1. Considering the correct "TU Dresden" and the false "Saarland" exemplary correspondences of table 1, both have 6 URLs overlapping but the distances between minimum rank values are diametric and thus add more discriminating power to the measure, e.g. resulting in 0.22 for the correct match vs. 0.11 for the false match.

The choice of web search engine also influences the results [Sp06]. Some, e.g. Google, have synonym dictionaries in place that may prove useful here, but may be biased towards the English language. Also the available spell checkers and the resulting "did you mean"-links could be followed to sort out typos, e.g. the strings "*Univerisity*" and "*MicrosoftWay*" did not return useful results via the used Yahoo API, whereas in the end user interface such typos are corrected automatically. Besides, newly established institutions may be unknown or underrepresented in the web search results. Thus, no. of returned hits could be taken into account or maybe generally limited to a lower number than the used 50. Furthermore, the time querying a web service is costly and the web service acts as a blackbox that can only be adjusted within the available parameters.

Often, for manual labelling, the decision whether a match is a true or false positive is not an either/or-decision. Correspondences identified as false positives from a syntactic perspective could as well hint to an institution merger or name change as e.g. Bell Labs formerly known as AT&T. Generally, background knowledge is needed to decide whether two strings denote the same or different real world entities.

### 3.3 Evaluation

For evaluation purposes we will hand-proof precision and recall of a subset. For this we determine the true positives, i.e. correctly identified correspondences, as well as the false positives, i.e. false correspondences, of each tested similarity metric. Based on the cardinalities of these sets, we thus can compute precision and recall:

$$\text{precision} = |\text{true positives}| / ( |\text{true positives}| + |\text{false positives}| )$$
$$\text{recall} = |\text{true positives}| / |\text{real correspondences}|$$

With precision the reliability of the found correspondences are judged whereas by recall the share of real correspondences that is found is nominated. To combine both measures in a single one f-measure can be calculated (see [DMR02] for a discussion):

$$F_\beta\text{-measure} = (1 + \beta^2) \cdot (\text{precision} \cdot \text{recall}) / (\beta^2 \cdot \text{precision} + \text{recall}), \text{ with } \beta \geq 0.$$

To quantify recall (and with precision also combined f-measure) the complete correct mapping is needed which we manually established for affiliation strings from the

[1] Not affiliated according to http://en.wikipedia.org/wiki/University_of_Chicago_(disambiguation)

publications of the SIGMOD 2007 conference as extracted from the ACM website. Here, 140 publications yield 150 different affiliations strings. We manually determined the perfect (symmetric) mapping having a size of 268 by determining clusters of affiliation strings and summing up the possible match correspondences. Only one third of the strings have correspondences, i.e. few institutions are represented by many affiliation variants whereas the majority of institutions only appear once. The following charts show lines for precision, recall, and $f_1$-measure (precision and recall evenly weighted).

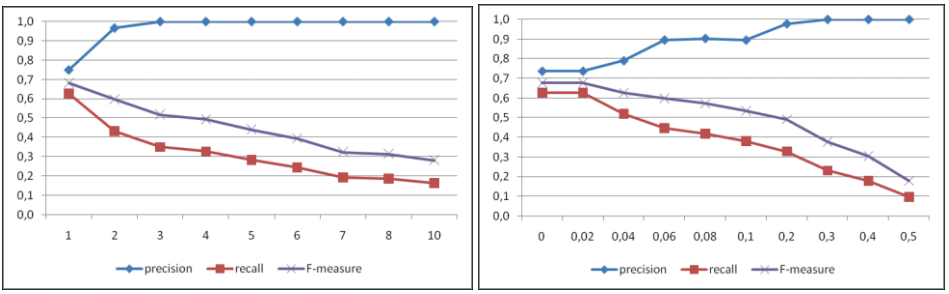



Figure 2: URL overlap similarity with at least a) x overlapping URLs and b) $sim_{URLoverlap}$

Concluding, according to this evaluation URL overlap (68% f-measure, figure 2) performs best for matching affiliation strings, followed by the soft tf/idf approach (55% at threshold 0.5, figure 3). We also tried trigram (47% f-measure) which still performed better than levenshtein (32%).

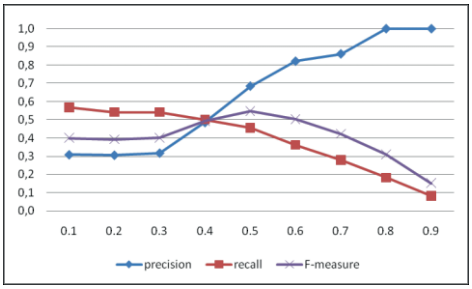



Figure 3: Soft tf/idf similarity with threshold t

Generally, the combination of URL overlap with classic string matching algorithms needs further testing and evaluations, as e.g. trigram or soft tf/idf combined with URL overlap to get acronyms/abbreviations could perform well in this context. Also left out in these experiments are manually established synonym tables for common abbreviations and blocking strategies as e.g. comparing academic affiliations and others separately. The danger here though lies in leaving out possible true correspondences, e.g. creating a subset of affiliations containing "Uni%" the "U of" matches would be lost. Further, more sophisticated pre-processing, e.g. not only separating off department but also institution name and location information, could boost similarity scores. Considering the fulltext of scholarly publications, institutions and locations could be derived from the authors' email addresses. Generally, to further evaluate the approach experiments with other data sets are needed, e.g. conferences and journals are variably named and referenced.

## 4 Results for a series of conferences

For identifying top institutions and top countries we examine ten years of SIGMOD publications as presented on the ACM web site. The collected data (table 2) consists of 1,026 papers (research, industrial, demo) with an average of 3.5 authors per paper across 200 different institutions in 1,044 different strings (993 distinct with department cut off).

| year | papers | affils | insts | year | papers | affils | insts |
|---|---|---|---|---|---|---|---|
| **1999** | 85 | 111 | 54 | **2004** | 118 | 145 | 64 |
| **2000** | 84 | 134 | 64 | **2005** | 116 | 146 | 78 |
| **2001** | 84 | 132 | 60 | **2006** | 99 | 140 | 79 |
| **2002** | 82 | 125 | 57 | **2007** | 140 | 150 | 79 |
| **2003** | 86 | 104 | 51 | **2008** | 132 | 173 | 90 |

Table 2: The examined data, ten years of SIGMOD

For further analysis we grouped affiliation strings with high similarity together, completed and corrected these clusters manually. In the ten years SIGMOD set there are e.g. 80 variants of IBM affiliations (with departments already cut off), in 2007 e.g., we have the following 10 specimens. For our analysis we denoted "IBM" as institution across all instances but assigned different countries accordingly.

```
IBM Almaden Research Center, San Jose, CA
IBM T.J. Watson Research Center, Hawthorne, NY
IBM T.J. Watson Research, Hawthorne, NY
IBM Silicon Valley Lab, San Jose, CA
IBM Toronto Lab, Toronto, Canada
IBM, Beijing, China
IBM, San Jose, CA
IBM Almaden Research Lab, San Jose, CA
IBM India Research Lab, New Delhi, India
IBM Toronto Lab, Markham, ON, Canada
```

Other years offer further variants, e.g.:

```
IBM, Markham, On, Canada
IBM Watson
IBM T.J. Watson Research Center, Yorktown Heights, NY
```

### 4.1 Publications by institution

We list top institutions in table 3 (next page). Apart from the Hong Kong University of Science and Technology (HKUST) and the National University of Singapore (NUS) the top 10 is dominated by US institutions. A further non-US candidate in this list of over 20 papers in 10 years of SIGMOD is the Indian Institute of Technology (IIT) Bombay. The top German institutions are the University of Munich with 11 positioned papers, the MPI with 7, the TU Dresden as well as the TU Munich with 6 each, followed by the University of the Saarland (5), RWTH Aachen, U Mannheim, U Marburg (4 each), HU Berlin, U Halle, U Konstanz, and U Leipzig (3 each).

| | '99 | '00 | '01 | '02 | '03 | '04 | '05 | '06 | '07 | '08 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| IBM | 6 | 11 | 11 | 19 | 11 | 15 | 14 | 10 | 18 | 20 | **135** |
| Microsoft | 7 | 6 | 8 | 5 | 8 | 15 | 14 | 10 | 19 | 11 | **103** |
| Bell Labs+AT&T | 14 | 11 | 10 | 16 | 12 | 13 | 5 | 8 | 5 | 6 | **100** |
| Stanford Uni | 5 | 7 | 5 | 2 | 10 | 6 | 4 | 1 | 2 | 1 | **43** |
| U of Illinois | 4 | 1 | 1 | 3 | 4 | 9 | 5 | 6 | 4 | 6 | **43** |
| U of Wisconsin | 6 | 3 | 1 | 6 | 2 | 10 | 3 | 5 | 4 | 1 | **41** |
| NUS | | 3 | 2 | 1 | 4 | 3 | 9 | 3 | 3 | 11 | **39** |
| UC Berkeley | 1 | 3 | 3 | 6 | 8 | 4 | 2 | 4 | 3 | 1 | **35** |
| Oracle | 2 | 1 | 4 | 5 | | 2 | 4 | 2 | 3 | 5 | **28** |
| HKUST | 2 | | 1 | 3 | 3 | 2 | 3 | 3 | 5 | 5 | **27** |
| U of Washington | 2 | 2 | 4 | 2 | 2 | 5 | 6 | | 2 | 2 | **27** |
| Cornell Uni | 3 | | 2 | 4 | 3 | 3 | 1 | 3 | 5 | 2 | **26** |
| U of Toronto | | | 3 | 1 | 1 | 2 | 4 | 4 | 4 | 6 | **25** |
| IIT Bombay | 4 | 4 | 4 | 1 | 1 | 4 | 1 | 4 | | 1 | **24** |
| Carnegie Mellon | | 3 | 1 | 2 | 5 | 3 | 4 | 2 | 1 | 2 | **23** |
| U of Michigan | 1 | 1 | 1 | 2 | 2 | 4 | 1 | 4 | 4 | 2 | **22** |
| U of Maryland | 3 | 3 | 1 | 2 | 2 | 3 | | 2 | 2 | 3 | **21** |

Table 3: Top institutions per papers in ten years (20+ papers)

### 4.2 Publications by country

For the following numbers (table 4) we have identified a country to each affiliation string variant, i.e. institutions may have multiple countries assigned. Breaking it down to states and cities is left to future analyses.

| | **'99** | **'00** | **'01** | **'02** | **'03** | **'04** | **'05** | **'06** | **'07** | **'08** | **papers** | **insts** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| US | 65 | 64 | 68 | 64 | 69 | 92 | 89 | 79 | 98 | 84 | **772** | 584 |
| CA | 5 | 6 | 8 | 6 | 8 | 11 | 12 | 6 | 11 | 12 | **85** | 48 |
| DE | 8 | 8 | 4 | 6 | 3 | 6 | 7 | 8 | 13 | 7 | **70** | 67 |
| CN | 2 | | 3 | 4 | 5 | 4 | 5 | 7 | 15 | 13 | **58** | 42 |
| SG | | 3 | 2 | 1 | 4 | 3 | 9 | 5 | 4 | 12 | **43** | 19 |
| IN | 4 | 6 | 5 | 3 | 2 | 6 | 2 | 4 | 2 | 4 | **38** | 38 |
| IT | 2 | 4 | 2 | 1 | | 3 | 4 | | 6 | 3 | **25** | 31 |
| KR | 2 | 3 | 2 | 3 | 3 | | | 2 | 4 | 2 | **21** | 20 |
| FR | 2 | 4 | 2 | | 2 | 3 | 1 | 2 | 2 | 2 | **20** | 27 |
| CH | 1 | | 1 | 1 | 2 | 1 | 1 | 5 | 4 | 4 | **20** | 14 |

Table 4: Papers per year and country, as well as distinct institutions per country (countries with 20+ papers only)

In tables 4 and 5 countries are denoted by their ISO 3166 country code. Figure 1 on page 1 already illustrated these numbers visually on a map, as rendered by the Google charts API.

Having multiple authors from different institutions or countries on a single paper can be interpreted as co-operations. Table 5 lists the numbers of papers in the 10 year SIGMOD set authored by researchers from different institutions within the same or across two and more countries. All listed countries have papers co-authored with US institutions. Apart from these, only few countries have papers together with authors of other countries' institutions (notably exception being China and Singapore).

| | US | GB | SG | NL | KR | IT | IN | IL | HK | FR | DE | CN | CH | CA | AU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AU | 2 | | | | | | | | | | | | | 2 | *3* |
| CA | 44 | | 4 | | | 2 | 5 | | | | 3 | 3 | | *12* | |
| CH | 6 | | | | | | | | | | | | *2* | | |
| CN | 18 | 2 | 12 | | | | | | 2 | | | *10* | | | |
| DE | 22 | 2 | | | | 2 | | | | | *13* | | | | |
| DK | 4 | | | | | | | | | | | | | | |
| FI | 2 | | | | | | | | | | | | | | |
| FR | 10 | | | | | | | 2 | | *6* | | | | | |
| GR | 6 | | | | | | | | | | | | | | |
| IL | 3 | | | | | | | *2* | | | | | | | |
| IN | 21 | | | | | | *9* | | | | | | | | |
| IT | 9 | | | | | *8* | | | | | | | | | |
| JP | 6 | | | | | | | | | | | | | | |
| KR | 12 | | | | *3* | | | | | | | | | | |
| NL | 2 | | | *2* | | | | | | | | | | | |
| SG | 14 | 2 | *2* | | | | | | | | | | | | |
| GB | 9 | *2* | | | | | | | | | | | | | |
| US | *285* | | | | | | | | | | | | | | |

Table 5: No. of co-operations between different institutions (2+ co-ops only)

## 5 Related work

In this paper we covered approximate string matching for heterogeneous variants. The linkage of short to long forms was studied recently in [Ta08]. The authors query web search engines with each form and link the short form (sf) to the long form (lf) if the lf is contained in the sf results' snippets and v.v. They also experimented with inverse hostname frequencies but did not take full URL overlaps into account, although the full URL is needed to discriminate e.g. the many Wikipedia hits. [El07] experimented with Jaccard similarity of hostnames returned by web searches, also neglecting the exact URLs. In the domain of bibliographic analysis the Citeseer project developed methods of extracting metadata from fulltexts, e.g. [Ha03] added initial support for affiliations. Location extracting or geotagging content is popular on the web, e.g. [Am04] disambiguates city names by taking context into account, [LB08] use unsupervised part-of-speech tagging to extract addresses, whereas we are concentrating on detecting location mentions by iteratively matching location strings of decreasing specificity. For erasing remaining ambiguities we plan to incorporate contextual information, e.g. email addresses. A recent citation analysis [RT05] also regards affiliations as extracted manually from the papers; for simplicity only the first author's institution were labeled,

though. As we collect data to all authors more types of analyses are possible, e.g. identifying possible co-operations between institutions.

## 6 Summary and future work

With the URL overlap similarity metric we presented a novel similarity metric for matching heterogeneous string variants denoting the same real world entity. The similarity builds on overlapping results of search engines queried with the strings. The affiliation match problem is a difficult one in that variants include acronyms and other abbreviations. We have shown that URL overlap outperforms levenshtein, trigram, and soft tf/idf in that task. Aligning affiliation strings are a needed step towards identifying outstanding institutions, i.e. only by clustering the variants to institutions and also locations (countries, states, cities) publications can be aggregated to project numbers of publications by institution and/or location. In this paper, we manually completed a first analysis of the last 10 years of SIGMOD publications. Results demonstrate again the dominance of US institutions (IBM with the most publications), followed by Canada and Germany as originating countries of many papers. With both the LMU and the TU, Munich can be seen as the top German city with the most publications (11+6) in this set.

We also see web applications making use of such data, e.g. the usual mapping mashup, probably with a timeline to illustrate not only the status quo but also change over time in the origins of publications to a conference series, journal, or topic. In web applications for categorizing publications, e.g. Caravela [AR07], institution and geographic location can serve as additional dimensions to search and navigate into the collection. Full automatic identification of outstanding institutions from scholarly papers is still to come, though. As related utilisation, names and locations of conferences and workshops could be aligned to categorize, rank, and map them by geographic location.

## References


[Am04] Amitay, E. et al. Web-a-where: geotagging web content. SIGIR, 2004

[AR07] Aumueller, D., Rahm, E., Caravela: Semantic Content Management with Automatic Information Integration and Categorization. ESWC, 2007

[CRF03] Cohen, W., Ravikumar, P, Fienberg, S. A Comparison of String Metrics for Matching Names and Records. Data Cleaning and Object Consolidation, 2003

[DMR02] Do, H.H., Melnik, S., Rahm, E. Comparison of Schema Matching Evaluations. Web, Web-Services, and Database Systems, 2002

[El07] Elmacioglu, E. et al. Web based linkage. Web information and data management, 2007

[Ha03] Han, H. et al. Automatic document metadata extraction using support vector machines. Digital Libraries, 2003

[LB08] Loos, B., Biemann, C. Supporting Web-based Address Extraction with Unsupervised Tagging. Data Analysis, Machine Learning and Applications, 2008

[RT05] Rahm, E., Thor, A. Citation analysis of database publications. SIGMOD Record, 2005

[Sp06] Spink, A. et al. A study of results overlap and uniqueness among major web search engines. Information Processing & Management, 2006

[Ta08] Tan, Y.F. et al. Efficient Web-Based Linkage of Short to Long Forms. WebDB, 2008

# Building Chemical Information Systems – the ViFaChem II Project

Sascha Tönnies[1], Benjamin Köhncke[1], Oliver Koepler[2], Wolf-Tilo Balke[1]

[1]Forschungszentrum L3S
Universität Hannover
Appelstraße 9a
30167 Hannover
(toennies, koehncke, balke)@l3s.de

[2]Technische Informationsbibliothek

Welfengarten 1B
30167 Hannover
Oliver.Koepler@tib.uni-hannover.de

**Abstract:** The interdisciplinary ViFaChem II project aims at providing a chemical digital library infrastructure for creating personalized information spaces. The value added services and scientific Web 2.0 techniques actively support chemical scientists and researchers in retrieval tasks as well as in deriving new knowledge from the collected information in a highly personalized fashion. The complex requirements of a digital library for chemists are described and an overall architecture tackling these requirements is presented. Also preliminary results regarding chemical entity recognition and automatic dynamic generated document facets are presented and discussed.

## 1 Introduction

Today digital libraries play a major part in information provisioning. Many information providers have extended their services from the traditional catalog-based search for literature to comprehensive digital portals, like the ACM digital library in computer science, searching for information over heterogeneous document collections and databases.

However, different scientific disciplines have their own demands and the respective community has different workflows and expectations when it comes to searching for literature. Hence libraries have branched out into topically centered virtual libraries for several disciplines closely focusing on the needs of each individual science. A good example of such a portal is the chemistry portal chem.de (http://www.chem.de) and it's embedded Virtual Library of Chemistry (ViFaChem). Services of this portal include searching in bibliographic databases, chemistry databases containing comprehensive factual data about molecules and reactions, and full texts of research reports.

But for chemical literature it is not sufficient just to provide keyword-based access. Chemical information basically deals with information about molecules and their reactions. To a large degree chemical information about molecules is communicated by their structural formula instead of verbal descriptions and practitioners can very efficiently discriminate between substances based on their visual representations. For a

high quality information retrieval it is therefore to cover the information provided about chemical entities based on actual chemical workflows. In order to do that strong interdisciplinary work is mandatory.

The common text based search e.g. using *entity names* for substances cannot be easily adapted to the chemical domain. For example the standardized IUPAC name, (2S,3R,4S,5R,6R)-6-(hydroxymethyl)oxane-2,3,4,5-tetrol is rarely used for D-glucose compared to the more prominent synonyms like dextrose, corn sugar, or grape sugar. An unambiguous identifier would be the *structural formula* defining a graphical representation of a molecular structure showing atoms, bonds and their spatial arrangement. We can thus see that only interdisciplinary work will lead to a high quality information provisioning platform that is promising to be accepted by a wide range of practitioners in the field. In fact, the graphical representation of chemical structure is the natural language for the communication of chemical information.

The ViFaChem II project focuses on using knowledge about chemical workflows as a basis for creating the digital library portal. The overall vision is a personalized knowledge space for the individual practitioner in the field. Building on (automatically derived) ontologies structuring the domain, openly accessible topical databases, and specialized indexes of substances derived from a set of user-selected documents, a personalized knowledge space can be created that promises to help users combating the information flood. The rest of the paper will focus on a typical chemical workflow and show how the ViFaChem II portal addresses the problems. Since ViFaChem II is still work in progress, in this paper we will discuss the overall architecture and present examples of how the particular modules work.

## 2 A Use Case Scenario for Chemical Workflows

The following scenario showcases the daily tasks of a researcher in the chemical domain. Assume our scientist is interested in anti-cancer drugs, particularly the class of taxanes see e.g. [Le05], their pharmacological activities and synthesis. He may start by looking for information about Paclitaxel and related drugs. Paclitaxel (often referred to under the brand names ‘Taxol’ or ‘Abraxane’) is a terpenoid isolated from the bark of yews, with a very high activity against several tumor cell lines. Naturally our researcher is especially interested in the mode of action of Paclitaxel and maybe other compounds with similar properties. Furthermore, he is looking for experimental procedures for the synthesis of Paclitaxel-like structures or precursors.

The common information retrieval process of a text-based search as known from other domains, will fail for this scenario for many reasons: the questions of the researcher involve information about chemical entities, concepts and facts. But as stated above queries involving chemical structure information either in form of substructures or similar structures can hardly be expressed in form of keywords. Of course, searching for the chemical entity name ‘Paclitaxel’ may return some results, but as a non-proprietary name it may not be used broadly in scientific research papers. One can try the IUPAC name of Paclitaxel as generated by a large ruleset published by the IUPAC. But

especially for complex molecules there are several ways how to interpret the IUPAC guidelines for nomenclature, so one still does not have a unique identifier for the molecule.

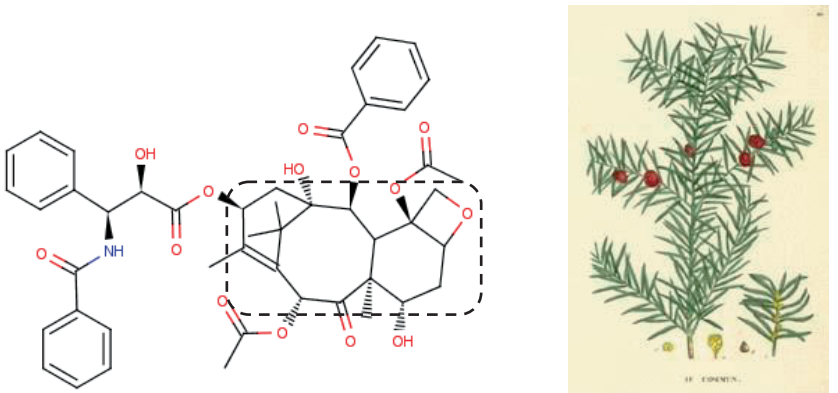



**Figure 1:** Structure of Paclitaxel (left) isolated from the yew tree (right)
(botanical image from: M.Grieve. 'A Modern Herbal', Harcourt, Brace & Co, 1931)

The only unambiguous representation of the entity Paclitaxel is its structural formula. Over the years several line annotations have been developed, which allow the conversion of graphical structure information into strings. Although compact strings they are not easy to handle by humans and therefore no alternative to a semantic rich drawn chemical structure. The following lines show the SMILES code for Paclitaxel.

**SMILES:**
CC1=C2C(C(=O)C3(C(CC4C(C3C(C(C2(C)C)(CC1OC(=O)C(C(C5=CC=CC=C5)N
C(=O)C6=CC=CC=C6)O)O)OC(=O)C7=CC=CC=C7)(CO4)OC(=O)C)O)C)OC(=O)C

Using a retrieval system with a graphical user interface our researcher can easily draw the molecular structure of Paclitaxel based on the rule sets of chemistry. Moreover, a structure based search is essential when it comes to the search for similar structures or structures which contain residues of a given lead structure.

Our scientist may find out, that taxadien-5-α-ol (cf. Highlighted C-Skeleton Fig. 1) is a central precursor in the biological synthesis of Paclitaxel. Therefore, he will search for molecules with this particular skeleton. At the latest now a text-based keyword search has become entirely useless. Such chemical structure related information retrieval can only be handled with a substructure search process. This operation is based on the information how atoms of the molecule are connected. However, the depth of structural information may vary: whereas the simplest representation contains only information about the composition of a molecule, a topological representation will also contain information about the connectivity of a molecule, showing which atoms are connected by which bond type. Moreover, the topographical representation can comprise spatial arrangements of atoms and bonds, showing the stereochemistry and conformation of a molecule, too. Therefore chemical structure databases generally contain information on

the level of topological representations. In contrast to the basic molecular formula, two-dimensional representations are the natural language of chemistry.

Thinking of integrating taxonomical information for the chemical domain, a retrieval system should also offer a navigational access over the related substance classes of the chemical entity. Thus, helpful information about structural superclasses and related substance classes is provided. Our researcher may know that Paclitaxel belongs to the taxoids, which are in turn diterpens with a taxan-like skeleton. The diterpens can be divided into acyclic, monocyclic, tri- and tetracyclic diterpens, Paclitaxel is a member of the later tetracyclic diterpens. The most prominent member of these tetracyclic diterpens is Phorbol, which interestingly is a strong carcinogen. Moreover, since our researcher is interested in drug design also information about the medical use of Paclitaxel will be used to retrieve documents and structure relevant information.

## 3 ViFaChem Architecture

Figure 2 gives a schematic overview of our ViFaChem II architecture. One problem when working with large document collections is the variety of file types like PDF, Word and HTML. All documents have to be processed in a specific workflow before they can be analyzed and indexed by suitable IR techniques before being stored in the ViFaChem II document repository. Section 4 will deal with the techniques used to fill the IR component of our architecture. The indexes created during the analysis are in turn offered for the personalized search functionality in the syndication step. We will discuss the various search functionalities in section 5. Finally, the user provides relevance feedback while building up a personal library from documents of the ViFaChem II collection that is then used for further personalization of the retrieval process.

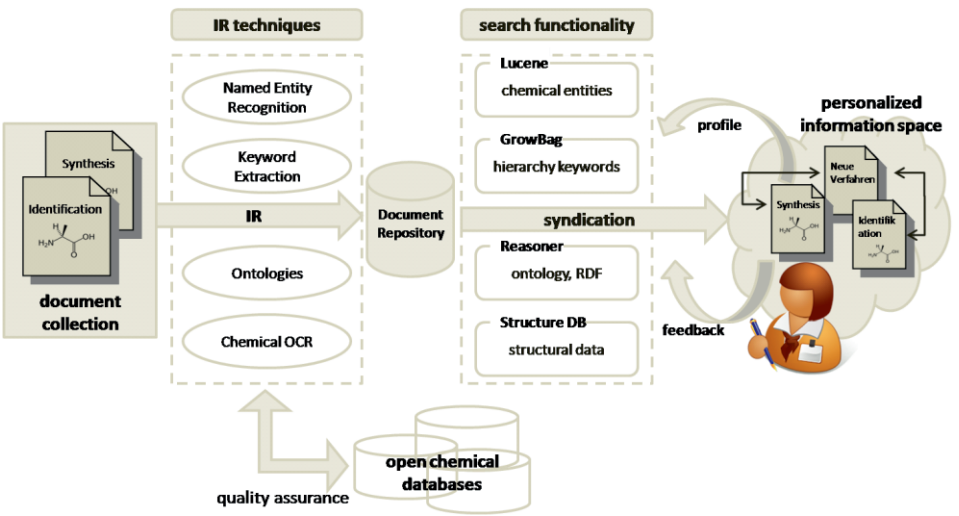



**Figure 2**: Basic ViFaChem II architecture

### 3.1 Basic Document Processing

Since we have to use many tools for deriving metadata for the use in our system, first all the different document types have to be converted into one general interface format. We rely on SciXML, which is a canonical XML format designed to represent the standard hierarchical structure of scientific articles and is originally described in [TCM99]. Its latest implementation, SciXML-CB, is based on an analysis of XML actually generated by scientific publishers in the fields of Chemistry and Biology [Ru06].

### 3.2 Search Engine Techniques

As described in our use case scenario researchers in chemistry are mainly interested in searching for structures. Since these structures can be quite complex, researchers often pose *graphical queries* to query a database. There is a wide collection of molecule editor software available to draw chemical structures. These programs generate a topological representation of each molecule for further processing; the data is usually handled in form of *connection tables*. Likewise a graphical query is transformed into such a format and the retrieval process takes place on the connection table representation. For the use in the ViFaChem II portal we integrated Marvin Sketch using Ajax techniques.

In addition, a *keyword based search* mechanism is needed enabling the user to make queries based on, e.g., brand names, CAS numbers, line annotations (InChI, SMILES), or bibliographic metadata. Here we used a simple inverted Lucene index.

We also introduced *facetted browsing* / *searching* as a navigational mode of access to the ViFaChem II portal. Here the Semantic GrowBag technology (c.f. 4.3) enables us to automatically generate facets from any subset (in the sense of defining particular user interest by an individual collection of documents) of our document repository. These facets can then be used for filtering the search result or just for browsing the data.

### 3.3 The Personalized Information Space

The actual *personal information space* relies on feedback and some organization by the user. Here it is possible to store search results together with documents, or subscribe to interesting periodicals and journals. Out of these saved documents the user profile is derived by extracting the relevant keywords from the documents. These keywords can then be used for query expansion or offered as facets for structuring the display of result sets, thus getting more relevant documents for the user. The portal is implemented with JBoss Seam using Sun JSF, RichFaces and Ajaf4JSF as main J2EE technologies.

## 4 IR Enabling Techniques

Chemical documents have to be extended by two types of metadata. The first type is the bibliographic metadata like authors, affiliation, publisher or year. Obviously in a library environment this metadata is readily available. The second and for our purposes more

important type is the *chemical metadata*. One can specify chemical metadata as the set of data regarding chemical entities, reactions, concepts and techniques, contained in the original document. This chemical metadata is not available out of the box and must be extracted, collected and structured. In the ViFaChem II architecture several IR techniques are used for extracting metadata, namely named entity recognition, keyword extraction, ontologies, and chemical OCR (c.f. figure 2).

### 4.1 Named Entity Recognition

The chemical substances considered within a document are obviously of great importance for subsequent searches. The recognition of *named entities* thus is a major step in preprocessing and indexing not only chemical documents. And indeed, natural language processing (NLP) techniques for named entity recognition in the bioinformatics domain are a highly active area. The community is assisted by a lot of publicly available resources like the well known PubMed/Medline corpus or the manually annotated PennBioIE[1] and GENIA[2] corpora. In contrast, development of NLP methodologies in the field of chemistry lags behind the biochemical world due to the lack of open access test corpora. Moreover, research in chemistry has a tradition of relying on manually edited and quite expensive commercial databases like for instance CAS Registry, the world's largest substance database with more than 15 million single- and multi-step reactions, and the respective interfaces like CAS's SciFinder. The only open source project currently available on the market is Oscar3 [CM06].

### 4.2 Keyword Extraction

Besides chemical metadata provided by the involved substances, also a keyword based search is necessary. Besides the typical recall-oriented full text search, here also means to aid high precision searches have to be provided. The task therefore is to tag documents with only the *highly relevant keywords* that best describe the focus of each document. For example interesting keywords may be the use of a substance for pharmaceutical purposes, or synthesis of naturally occurring substances, etc. To provide this feature all documents have to be analyzed regarding the so-called top-k keywords, i.e. the k most relevant keywords for discriminating documents with respect to a collection. Here different algorithms, e.g. TF/IDF [SG83] or Word Co-Occurrence [MI04], can be used to remove too general or often occurring phrases. After identifying the most relevant keywords for each document they are stored in an inverted file index using Apache Lucene.

### 4.3 Ontologies / Taxonomies

*Controlled vocabularies* or *taxonomies* have been found to be very useful, in particular for navigation in large result sets. The need for building up some ontology results from

[1] http://bioie.ldc.upenn.edu
[2] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA

the fact that chemical data can be represented in over 80 different file formats, but none of them, including the Chemical Markup Language [MR01], are capable of encoding knowledge in such a way that the meaning is completely preserved. Semantic Web ontologies aim to describe and relate objects using formal, logic-based representations that a machine can understand and process. So by leveraging Semantic Web technologies, it becomes possible to integrate chemical information at differing levels of details and granularity. Unlike the bio-medical domain with the MeSH taxonomy the chemical domain has access to just one single highly specialized controlled vocabulary openly available, called ChEBI [De08]. This also reflects a general problem of the chemical domain, in that the amount of data is split into a large number of sub domains making it extremely unrealistic to build a taxonomy for the entire area of chemistry.

One solution for this problem would be manually building up taxonomies for each sub domain leading to an enormous effort and needed man power. Since public libraries like the TIB have to cater to a vast variety of customers from industry and academia, however, the document collection is already highly diverse and constantly evolving. Thus the ViFaChem II portal cannot be restricted to a single taxonomy, but needs strong means of personalization without having to maintain hundreds of taxonomies and vocabularies. What is needed is an automatic way of generating some, at least light-weight, taxonomies for individual (groups of) users. Here ViFaChem II relies on the Semantic GrowBag technique [DBT07] with quite promising preliminary results.

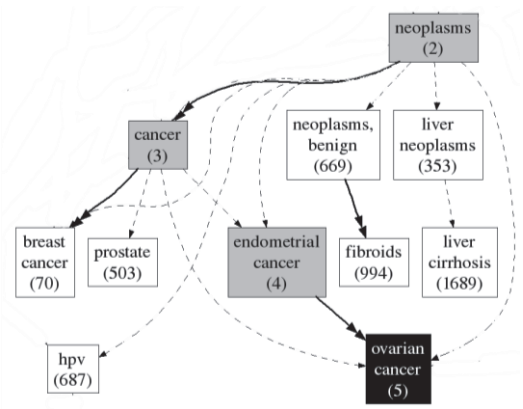



**Figure 3**: A sub-region of the Taxol graph generated by the Semantic GrowBag

Basically the Semantic GrowBag hierarchically organizes keywords provided in metadata annotations of digital objects based on the actual usage. It analyses the first and higher order co-occurrence of the keywords using a biased PageRank algorithm and generates a structure of nodes (keywords) and their relationships. By defining a set of resources as a starting point for the Semantic GrowBag (and thus defining their broad area of interest), users support the personalization task. Figure 3 shows a sub-region of the complete GrowBag graph automatically generated for the keyword 'Taxol' over the Medline document collection. Since Medline is focused on medical uses, keywords on the pharmaceutical action are prevalent in the GrowBag graph. Remember that Taxol is

often used in the area of cancer and especially for ovarian cancer, which indeed is a form of endometrial cancer and highly related to cancer of the breast and prostate.

Our ongoing research deals with question, whether it is possible to build up a lightweight ontology based on the GrowBag by qualifying relationships using additional domain knowledge gathered from open access data sources like PubChem. There are some examples of small chemical ontologies, e.g., [KLD08] and CO [Fe05], but up to now they still suffer from not representing enough knowledge to be useful for chemical information systems with a heterogeneous user community.

# 5 Search functionalities

After extracting all available metadata from the documents, this metadata is stored within a document repository which then serves as the basis of our search functionalities.

## 5.1 Chemical databases

Chemical structure databases are queried using a graphical interface, where a researcher can draw his/her query as a molecular structure or substructure, thus using the natural, semantically rich language of chemistry. The mouse drawn molecule query is further processed by the search engine, starting with a preliminary screening against the database keys followed by a pattern matching of the query across the returned subset of the data from the screening. Chemists differentiate between exact structure searches, substructure searches, and similarity searches. While the exact search will return only exact matches, the substructure search will return all structures including a given substructure. A similarity search will return structures based on the calculation of a similarity match, which can vary from database to database.

As already mentioned in the introduction chemical databases can either be used for retrieving chemical entities of interest or as a specialized index for scientific literature, where chemical structures represent some kind of abstract of an article. In ViFaChem II the chemical database is used as the later. The collection is structured according to an ER diagram, where the document is the central point and all other information are linked to this entity. Each entry in the document table is linked to an electronic version of the document, a link to its representation in SciXML, and a link to the annotated SciXML file. Besides, of course also the bibliographic metadata is stored in the database.

In addition we also store the chemical metadata: each chemical entity, reaction, technique and concept is stored in a related table always linked to all documents where it occurs. All this factual data and metadata can easily be stored in some arbitrary relational database system like MySQL. However, for efficiency and improved handling the structural data is stored in the specialized chemical structure database JChem Base.

### 5.2 Facetted Browsing and Searching

As already stated in section 4.3, taxonomies have been found very useful for navigating in large result sets, see e.g. Faceted DBLP. Hence, the ViFaChem II portal also includes a faceted search and browsing interface. The facets ViFaChem provides are adapted to the current search result. This means that there is no pre-categorization of any document relying on the expensive maintenance of a suitable category system.

When a user queries the document repository he/she will get a result set of documents related to the query. Using this result set the individual top-k keywords are calculated for personalizing the result set. Based on those top k keywords the nodes and edges of the respective pre-computed GrowBag graphs are retrieved and the most relevant keywords of the first order co-occurrence are shown as a starting point for result organization. These facets can then be selected for filtering the search results. For browsing the whole document collection the overall top k keywords are used as a basis for filtering.

### 5.3 Index based Search

Our bibliographic and chemical metadata, as well as the entity structures are stored in relational databases. But working with databases in the field of digital libraries usually lacks ranking features. Therefore ViFaChem II also uses another search structure. A wide spread technique is the duplication of the data in an inverted index. This works fine with text based data like the bibliographic metadata. But in the domain of chemical information systems, we also have to query the structure database which data cannot simply be put inside some inverted index. One solution for this problem would be the extension of a search engine framework such as Apache Lucene to be able to support graphical structure and substructure search. A more simple approach is using an object identifier for each structure inside the database as surrogate key connecting the structure to the actual representation of each entity. In that way the inverted index can be queried for the entity and all related documents can be retrieved ranking the result set.

## 6 Summary and Outlook

In this paper we have outlined the goals and discussed preliminary results and techniques of the joint interdisciplinary ViFaChem II project carried out at the Research Center L3S and the German National Library of Science and Technology. Throughout the paper we argued that chemical information systems for digital libraries with advanced content-based query methods and powerful means of personalization have to be developed in a strongly interdisciplinary fashion. ViFaChem II therefore works in tight cooperation with the Chemistry Information Centre (FIZ CHEMIE), Georg Thieme Publishers and the German Chemical Society (GdCh), analyzing and considering the specific demands and workflows for information retrieval in chemistry.

The ViFaChem II prototype offers modules for the extraction, indexing and searching of new (chemical) metadata in addition to the traditional bibliographic metadata for large

document collection. While the processing and segmentation of PDF documents is still challenging, the results with HTML and XML documents are most promising and the resulting annotated documents already offer semantically rich and –what is even more important for a library as information provisioner- correct metadata. The preliminary results of the Semantic GrowBag algorithm for the chemical domain also show interesting results for building up a chemical taxonomy, and will be further investigated with respect to automatically building lightweight ontologies.

Our future work will focus on the integration of the different modules of ViFaChem II into a single user interface to provide the semantic rich metadata not only for traditional keyword based searches, but also for navigational browsing based on taxonomies and ontologies. Combined with profile-based filter mechanisms, the ViFaChem II information retrieval process will then allow a fast and guided access to a large collection of the most relevant documents for practitioners in the field of chemical research in both industry and academia. We will also conduct user studies about the portal's usability aspects and foster the integration into the *chem.de* portal.

## 7 References


[CM06] Corbett, P.; Murray-Rust, P.: High-Throughput Identification of Chemistry in Life Science Texts,. In Proc. CompLife, 2006

[DBT07] Diederich, J.; Balke, W-T; Thaden, U.: Demonstrating the Semantic GrowBag: Automatically Creating Topic Facets for Faceted DBLP. In Proceedings of the ACM IEEE Joint Conference on Digital Libraries, Vancouver, BC, Canada, 2007

[De08] Degtyarenko, K., de Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., Alcántara, R., Darsow, M., Guedj, M. and Ashburner, M.: ChEBI: a database and ontology for chemical entities of biological interest. Nucleic Acids Res. 36, D344–D350, 2008

[Fe05] Feldman, HJ.; Dumontier, M.; Ling, S.; Hogue, C.W.: CO: A Chemical Ontology for Identification of Functional Groups and Semantic Comparison of Small Molecules. FEBS Letters. 579, 4685-4691, 2005

[KLD08] Konyk, M.; De Leon, A.; Dumontier, M.: Chemical Knowledge for the Semantic Web. In Proceedings of Data Integration in the Life Sciences (DILS2008), Evry, France

[Le05] Leistner, E.: Die Biologie der Taxane: Arzneimittel aus der Natur. In Pharmazie in unserer Zeit, 34 (2), 98-103, 2005

[MI04] Matsuo, Y.; Ishizuka, M.: Keyword extraction from a single document using word co-ocurrence statistical information. International Journal of AI Tools, 13 (1), 157-170, 2004

[MR01] Murray-Rust, P.; Rzepa, H.S.: Chemical markup, XML and the World Wide Web. 2. Information Objects and the CMLDOM, Journal of Chemical Information and Computer Sciences, 41 (5), 1113 -1123, 2001

[Ru06] Rupp, C.J.; Copestake, A.; Teufel, S.; Waldron B.: Flexible Interfaces in the Application of Language Technology to an eScience Corpus. Proc. 6th E-Science All Hands Meeting (AHM2006), Nottingham2006

[SG83] Salton, G.; McGill, MJ.: Introduction to Modern Retrieval, 1983

[TCM99] Teufel, S.; Carletta, J.; Moens, M.: An annotation scheme for discourse-level argumentation in research articles. In Proc. EACL, 1999

# GEM: A Generic Visualization and Editing Facility for Heterogeneous Metadata

Jürgen Göres, Thomas Jörg, Boris Stumm, Stefan Dessloch
Heterogeneous Information Systems Group, University of Kaiserslautern
{goeres|joerg|stumm|dessloch}@informatik.uni-kl.de

**Abstract:** Many model management tasks, e.g., schema matching or merging, require the manual handling of metadata. Given the diversity of metadata, its many different representations and modes of manipulation, meta-model- and task-specific editors usually have to be created from scratch with a considerable investment in time and effort. To ease the creation of custom-tailored editing facilities, we present GEM, a generic editor capable of visualizing and editing arbitrary metadata in an integrated manner. GEM provides a stylesheet language based on graph transformations to customize both, the mode of visualization and the available manipulation operations.

## 1 The importance of metadata

The vision of *generic model management* spurred by the works of Bernstein et al. [BHP00, MRB03] aims at reducing the effort to create metadata-intensive applications by defining generic operators that work on entire models and providing a model management system that implements these operators. Metadata-intensive applications can then be built on these systems like data-intensive applications are built on database management systems today. Examples of such applications include the broad area of information integration or the development of complex software systems.

In our research group, we work on novel approaches to create and maintain information integration systems. Creating an integration system subsumes numerous tasks, which all require the handling of metadata artifacts: Integrated schemas are designed from scratch or created by merging the source schemas. Semantic correspondences between the schemas have to be identified and be made explicit by schema matching. Based on these correspondences or "matches", mappings that perform the required data transformations have to be developed, e.g., by configuring wrappers of a federated DBMS and specifying view definitions over the wrapped sources, or by creating ETL scripts for replication-based integration. Existing integration systems require intensive maintenance operations: Changes to system components require the modification of matches and mappings.

30+ years of research have resulted in numerous approaches to automate some of these tasks, like automatic schema matching and merging techniques. However, for the foreseeable future, these approaches can at best be used in a semi-automatic fashion, therefore requiring human expertise to review, correct, and amend their results. Other tasks, like the design of schemas and software artifacts, are intrinsically manual. Human integration experts and software engineers therefore have to be provided with suitable interfaces to manipulate the many different kinds of metadata required for these tasks: Database schemas

are often designed using conceptual metamodels like one of the many E/R variants, and are only later mapped to physical schemas, represented by a data definition language of the respective data model like SQL DDL or XSD. For the design of the structure and the dynamic aspects of software components, diagrams of the UML family are used. Conceptualizations of application domains as ontologies are often represented by RDF or OWL documents.

**Problem statement** Metadata can truly be said to be omnipresent in many disciplines of computer science and manual metadata manipulation is often a necessity. Due to the complexity of metadata artifacts and the diversity of metadata representations this is by no means a trivial task. Many types of metadata have a native textual representation, but graphical representations are generally considered easier to handle by humans. Already a 1:1 mapping from textual to a graphical representation can often improve the understanding and manageability of models. But given the enormous volume and complexity of real-world metadata, different degrees of abstraction are often an absolute necessity to make handling of large models by human developers feasible.

The development of editors for such graphical representations means a significant investment in time and effort, and incurs the well-known risks of any large software project. This is acceptable when creating a commercial tool for an established and well-defined metamodel, e.g., a UML CASE tool. These tools are, however, limited to their native metamodel and only support the editing functionality anticipated by their designers. Extending the capabilities requires changing the code of the editor – if available as open source – or is simply not possible for closed-source tools.

This is especially a problem in the context of information integration and metadata or model management in general, where both the metadata and the operations on this metadata are often too diverse to be handled by a single tool. As a consequence, different tools have to be used in combination. Consequently, not only do developers have to to familiarize themselves with each of these tools, but are also impeded by the lack of interoperability, caused by the many proprietary formats to represent the metadata artifacts that are created and manipulated. In addition, often metadata from different metamodels has to be handled in an integrated fashion, e.g., for schema matching. Simply using different tools in parallel cannot help here. As an alternative, integrated tool suites avoid the metadata exchange problem (at least for those aspects covered by the suite), but will in general not be able to provide the optimal solution for each individual task. In addition, tools and tool suites are often tightly coupled with other products of their respective vendor, limiting the potential reuse of the artifacts created by such a tool.

While a manual combination of existing tools and tool suites may sometimes offer a cumbersome yet working way to perform the desired metadata management tasks in production environments, researchers often have very specific editing requirements for which no tools exist: They have to handle proprietary, complex models, where both the metamodels and the kinds of operations on the models are ever evolving as research progresses. Unable to commit the resources to create their own editor from scratch and adapt it to the changes, very often they have to go without a suitable editing tool.

**Contribution** In this paper, we present *GEM*, a generic visualization and editing facility for arbitrary graph-based data. GEM allows to rapidly develop metamodel- and task-specific editors without the need to write a single line of program code. For this purpose, GEM provides a declarative *graph stylesheet language* to easily adapt its visualization and editing functionality to any such metamodel and editing task.

Graph stylesheets are based on the concept of *graph transformations* and define a set of *visualization rules* that translate the application data provided in a graph representation to a *visualization graph*. The nodes and edges of the visualization graph correspond to an extensible set of user interface elements (so-called *widgets*). The visualization graph is directly interpreted and displayed by the editor.

For editing, a graph stylesheet contains a set of *edit operations*, which define how manipulations of the graphical elements are to be propagated to the application model. GEM stores application and visualization graphs in a relational database system; graph transformations are translated to SQL DML statements. This implementation approach has shown to have adequate performance even for large models, and outperforms any of the existing graph transformation systems we evaluated.

The remainder of the paper is structured as follows: Section 2 gives an overview of the general concepts of the editor. Section 3 introduces the graph-based representation of arbitrary (meta-)data. Section 4 gives an introduction on graph transformations in general and the specific transformation formalism underlying the GEM stylesheets, which are presented in detail in Section 5. Section 6 demonstrates GEM's practical usability in a realistic scenario. Section 7 highlights interesting aspects of the prototype's implementation and gives performance measurements. Section 8 gives an overview of related work, and Section 9 closes with a summary and an outlook on future work and usage scenarios.

## 2 Overview

In this section, we give an introduction into GEM. First, we describe the editor from a user perspective, to show its applicability in real world scenarios. Then, we give an overview from a stylesheet developer perspective, illustrating the steps needed to adapt the editor to a certain metamodel. Finally, we give a high-level architectural introduction into the editor.

The primary goal in the development of GEM was to provide a customizable editor for arbitrary metadata. However, the editor is not limited to the role of metadata editing: it works on graphs representing any kind of data. To emphasize our focus on metadata editing and to use a terminology that is in line with our application scenarios, we will refer to the data being displayed and edited as an *application model*, or *application graph*. The prototype of GEM will be made publicly available on the GEM project site[1]. Currently supported DBMSs are Apache Derby, H2, IBM DB2, and PostgreSQL.

[1] http://wwwlgis.informatik.uni-kl.de/cms/index.php?id=GEM

### 2.1 GEM from a user perspective

The basic process of editing an application model consists of several steps. First, the user imports an application model file into the editor database, comparable with the "open" action in conventional editors. Several different models can be imported and edited simultaneously. Before the model can be edited, the user has to apply a graph stylesheet to it. GEM will then visualize the application model according to the rules defined in the stylesheet. The default stylesheet gives an 1:1 view of the graph. Custom stylesheets can provide an abstracted view of the model, depending on the actual needs. For example, an SQL schema could be represented in an UML-like notation, as an E/R diagram, or even as a set of plain, pretty-printed DDL statements. The design of the GEM prototype makes it easy to enable support for multiple views of the same application model. For example, an abstract view can be combined with a detail view (maybe only of the selected part of the graph), or different spanning trees for hierarchic displays of the same model may be produced. This can be achieved by using different stylesheets on the same model.

Editing functionality can be separated into two categories: GEM directly provides a basic layouting functionality pertaining solely to the presentation of the model, like manually or automatically layouting graph elements. These operations will not change the application model. All such layout data is preserved between edit sessions. To actually edit the application model, a stylesheet provides *edit operations*. An edit operation is an arbitrarily complex, model-specific operation. A stylesheet for SQL schemas could provide operations like "create new table", "add column to table", or "rename table". For model management, there also might be more complex operations like "copy table" or "denormalize tables". GEM automatically provides a menu with all possible edit operations defined in the stylesheet. To apply an operation, the user might have to select a part of the graph (e.g., the table to which he wants to add a column) and provide input parameters (like column name and type). Then the operation is executed, directly changing the application model. After that, the visualization is updated to reflect the changes in the application model, preserving as much of the manual layouting as possible. Manipulations done through one editor window will directly update the views in the other windows. At any point, the user can export the application model for further processing by other tools.

### 2.2 Stylesheets in GEM

One of the distinguishing characteristics of GEM is the use of graph stylesheets to customize and adapt the editor to a specific data model. This allows us to visualize and edit not only SQL schemas or XSD files, but *arbitrary* metadata, possibly in a specialized format. Thus, the customization process must be powerful enough to cope with a great diversity of requirements, but also simple enough to allow for a fast adaptation of GEM to a specific data model even by non-experts. We believe that we have achieved this goal and in this section we will illustrate this by looking at GEM from the perspective of a stylesheet developer.

The first step in developing a custom stylesheet is creating the *visualization part*. The developer needs to define rules to specify how elements in the application model should be displayed in the edit pane. This is similar to the development of CSS or XSLT stylesheets.

To represent stylesheets, the current GEM prototype uses an XML format, but future versions will support a more concise textual and later a more vivid graphical representation. With a stylesheet, a developer can choose to present an SQL model as plain text DDL, or choose to hide details and only show the table names and how they are related through foreign key relations. The details of defining these rules are explained in Section 4.

The second part of a stylesheet are the *edit operations*. Instead of having users resort to tedious, fine-grained operations on the atomic elements of the graph structure like with other graph editors, the stylesheet developer defines high-level edit operations for all tasks that should later be performed by users of the stylesheet. This not only greatly improves the usability of the resulting editing functionality, as very complex modifications can now be performed with a single editing step. Properly designed edit operations can also guarantee the consistency of the application model, as users can only perform semantically valid edits. Moreover, edit rules allow us to resolve the view-update problem which arises when editing a model through an abstracted visualization.

For most requirements, using the built-in widgets and functions is sufficient for defining visualization rules and edit operations to adapt GEM to custom needs. Sometimes, however, the built-in functionality might not be enough, or might require very complex and non-intuitive stylesheets. Therefore, GEM allows customization not only by defining declarative stylesheets, but also by providing an extension mechanism for user-defined functions and widgets.

User-defined functions in GEM are written as static Java methods, allowing to implement virtually any domain-specific functionality. In a graph stylesheet, the developer can use these functions in visualization rules and edit operations. GEM provides several basic widgets to visualize application models. Besides simple boxes, ellipses, or arrows, there are grouping widgets which allow sophisticated layouts. If this is not sufficient, the stylesheet developer can define his own widgets by implementing a GEM widget interface.

### 2.3 GEM architecture

Figure 1 gives a high-level overview of the important concepts of our approach and their interactions. In the center, the complete model graph consisting of the application graph and the visualization graph is depicted. The two subgraphs are connected via RepresentedBy edges, each connecting an element in the application graph with an element in the visualization graph that represents this element. Application and visualization elements can be in n:m relationships, as an application graph element can be represented by more than one visualization node (and thus appear more than once in the edit pane), and a single visualization node can represent more than one metadata element.

On the right hand side of the figure, the editor pane and the widgets are depicted. They are responsible for the display of the visualization graph and observe it for changes. Visualization rules are used to initially create and later update the visualization graph and the RepresentedBy edges. They can refer to the application graph as well as to the visualization graph, but they can modify only the visualization graph. Edit operations can refer to both subgraphs, but modify only the application graph and never the visualization graph. To keep the visualization synchronized with the application model, each application of a

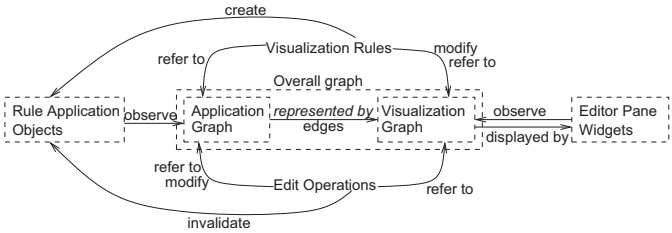



Figure 1: Overview of the stylesheet-based graph visualization and editing approach.

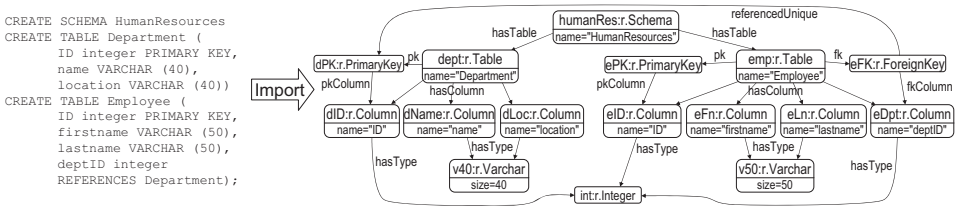
visualization rule is recorded in a rule application object (RAO). Execution of edit operations can invalidate RAOs, if after the edit operation the premise for the rule application does no longer hold. Invalidated RAOs are then undone and visualization rules are selectively re-applied, so that after a change of the application model this change is directly reflected in the visualization model. We will discuss this in greater depth in Section 7.2.

## 3 Metadata representation

In this section, we will first introduce graphs as the common representation for all metadata to be edited with GEM. Graphs are probably the most general data structure, allowing a lossless representation of virtually any metadata artifact. In general, a graph consists of a set of nodes $N$ that are connected by a set of edges $E$. Many variations of graphs exist that differ in aspects like their support for labels and attributes on nodes and edges, whether edges are directed or can connect more than two nodes (hypergraphs) etc. Likewise, the formal definitions differ, e.g., whether edges are seen as independent objects or are only given as a relation $E \subseteq N \times N$. We have chosen to use directed, attributed, labeled multigraphs, since they strike a median point in between the verbose representations that result from using a very basic graph formalism (e.g., labeled graphs) and the complexity of some of the more elaborate formalisms. Our Definition 1 is based on [KR90]:

**Definition 1 (Directed, attributed multigraph)** *A directed, attributed multigraph $G$ is a tuple $(N, E, src, tgt, \Gamma, \Sigma, att_N, att_E)$. $N$ and $E$ represent the set of nodes and edges, resp. $\Gamma$ is the set of attribute identifiers. $\Sigma$ is the set of attribute values. The mappings $src\colon E \to N$* and *$tgt\colon E \to N$ associate a start and end node with each edge. The mappings $att_N\colon N \times (\Gamma \cup \tau) \to \Sigma \cup \bot$* and *$att_E\colon E \times (\Gamma \cup \tau) \to \Sigma \cup \bot$ return the (possibly empty) value of the given attribute identifier for the given node or edge, respectively. $\tau$ is the identifier of a special attribute representing the* type *of nodes and edges.*

The advantages of this basic graph metamodel is its generality and minimality. Nodes can represent arbitrary application elements, while edges can be used to represent the relationships between them. Node attributes represent the properties of objects, while edge attributes can be used to model more refined kinds of relationships, e.g., a *position* attribute can specify an ordered relationship. Metadata to be edited with GEM first has to be transformed from its native representation (e.g., SQL DDL statements) to a multigraph according to Definition 1 (see Figure 2). The editor does not place any further restrictions on




a) SQL metadata in native representation

b) SQL metadata in graph representation

Figure 2: Representing metadata as graphs.

this representation. The graph representation can therefore be customized to the applications needs, e.g., to ease the definition of stylesheets or to provide round-trip capabilities. GEM's primary vehicle for graph import is the Graph Exchange Language [WKR01], an XML format for the exchange of graphs and type graphs. The research projects PALADIN [Gö05, GD07] and Caro [Stu06, SD07] provide converters from their internal representation of models to GXL, enabling the import of, e.g., SQL DDL and XSD. Also, direct extraction of metadata from the information schema of an RDBMS is supported. The effort for converting custom models to and from GXL is manageable, since mapping of any kind of object-oriented representation to and from our graph model is straightforward.

# 4 Graph transformations

In this section we will describe the basics of our graph transformation formalism that is the foundation for our graph stylesheet language. While graph transformations have been a research topic for approximately 40 years, they are still not widely employed outside the graph community. Only recently has the concept attracted rising interest in the context of model-driven architecture (MDA), where it is used for model-to-model transformations (e.g. [CH03]). Two main reasons for this lack of acceptance can be named: First and foremost, very few graph transformation systems (GTSs, also *graph rewrite systems*) have reached a degree of maturity beyond that of research prototypes. A second problem for the limited acceptance of graph transformation systems as a generic tool for software and information systems engineering is the diversity of existing approaches, both regarding the formal semantics of transformation rules and systems, as well as in the different languages used to describe them. We have no intention to discuss the benefits and drawbacks of each of the approaches to motivate our formalism, but instead hope that we can let its clarity and naturalness speak for itself. A common criticism on graph representations and operations is the issue of computational complexity: many graph problems, in particular the subgraph isomorphism problem, which is of high relevance for graph transformations, are known to be NP-complete. However, since most practical scenarios use labeled and regular graphs, the average case complexity is well within reasonable bounds, as we will see in our evaluation in Section 7.3.

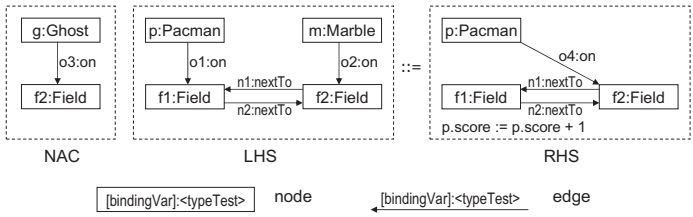



Figure 3: A simple production rule.

### 4.1 Production rules

A graph transformation system takes as inputs a *host graph* representing the input data to be transformed, and a set of graph transformations rules. It then outputs a new graph or changes the input graph. To specify an individual graph transformation, we use a syntax based on production rules (see [Hec06] for an overview). They have a *left-* and a *right-hand-side* (LHS, RHS). As an example, Figure 3 shows a simple production rule that models a part of a game of Pacman, where the host graph represents the current game state (labyrinth fields and their connections, the position of Pacman, ghosts, and marbles). The LHS specifies a *graph pattern* consisting of pattern nodes and edges (*pattern elements*), which has to be found in the host graph for the rule to be applicable. Pattern nodes and edges can specify a type test, given as the part of an element's label behind the colon. In our example rule, the LHS specifies a situation where Pacman *p* is on a Field *f1* that is next to a Field *f2* containing a Marble *m*. A subgraph of the host graph that matches the LHS is called an *occurrence*. Occurrences can be seen as a mapping from the elements of the graph pattern to those of the host graph. For the scope of this paper, we assume occurrences to be injective, i.e., an occurrence will never map two pattern elements to the same host graph element.

The RHS describes how an occurrence is modified by the rule. To indicate corresponding elements on LHS and RHS, *binding variables* are used. They are essentially an identifier for pattern elements and form the first part of a rule element's label in front of the colon. Elements on the LHS that also occur on the RHS indicate that the element is preserved, for example the Field nodes *f1* and *f2*, or the Pacman node *p*. An element on the LHS that does not occur on the RHS indicates that the element will be deleted (e.g., the Marble node *m* or the on edges *o1* and *o2*), while an element occurring only on the RHS signals the creation of a new element (e.g., the on edge *o4*). The part of a RHS element's label behind the colon is a type assignment, to indicate the type of new nodes and edges, or to change the type of preserved elements. In addition to changing node types, a RHS can modify attributes of nodes and edges by referring to their binding variables (e.g., incrementing Pacman's score attribute). So our example rule will move Pacman from his current field to a neighbouring field containing a marble. In the process, the marble will be consumed and Pacman scores a point.

Additional *application conditions (ACs)* can be specified to limit the applicability of the rule. These can refer to attributes of LHS elements (pure *attribute conditions*) or specify graph patterns that must or must not occur for the rule to be applicable (structural ap-

plication conditions): Positive structural application conditions (PACs) specify additional structures that have to be fulfilled, while negative structural application conditions (NACs) specify structures that must not occur. Pattern elements in ACs can also be connected to the LHS by binding variables. In our Pacman example, the rule is only applicable when there is no Ghost *g* on the Field *f2* containing the Marble *m*.

For a more compact presentation of rules, we will use an *integrated representation* in the remainder of this paper. Figure 4 shows the rule of Figure 3 in integrated representation: deleted nodes and edges are indicated by dashed lines, while created elements are shown with bold lines. Elements of application conditions are grouped and marked with PAC or NAC, respectively. While the expressiveness of this representation has some limitations, it is sufficient for the compact representation of the rules used in this paper.

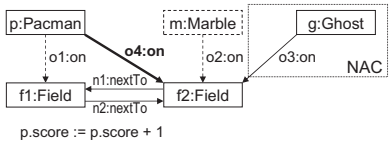



Figure 4: The production rule of Figure 3 in integrated representation.

### 4.2 Operational semantics

The operational semantics of graph transformation systems varies for the different approaches proposed in the literature. The original mode of application of a set of production rules is similar to that known from formal languages: As long as production rules are applicable, the system randomly chooses both a rule and an occurrence, and applies the rule. This non-deterministic rule application often makes it difficult for the author of a graph grammar to ensure that the rules do exactly what is intended. Very often, rules have to be applied in a certain sequence or to specific regions of a host graph. A number of approaches have been introduced to control the application of rules more precisely (see [BFG95] for an overview). It should be noted that none of them increases the expressive power of a GTS as a whole, but can considerably reduce the number and complexity of individual rules.

*Rule layering* is a concept that was first introduced by AGG [Tae99], a graph transformation system implemented in Java. Layers define a partial order among rules: A layer contains a number of rules which are applied non-deterministically as described above, until no more rules of the layer are applicable. Rule application then continues with the rules from the next layer. Within a layer, the formal properties of non-deterministic graph grammars hold, while the ordering gives a basic degree of control over the application of rules. In GEM, the visualization of application graphs is performed using a restricted layering approach, where each layer contains exactly one rule (see Section 5.1).

While layering is sufficient for simple sequential application of rules, complex operations cannot be expressed as easily. We wanted to be able to specify complex operations in a natural way, so we chose the three basic control structures introduced by Göttler [Gö88] due to their limited complexity and minimality for the definition of edit operations: *Se-*

*quential application (SAPP)* applies the contained rules (or nested control structures) once in the given sequence. *Case application (CAPP)* will apply the first applicable rule of a given ordered list of rules. *While applicable (WAPP)* will apply a contained rule (or nested control structure) repeatedly, until it is no longer applicable .

## 5 Graph stylesheets

With the basics on graph transformations introduced in the previous section, we now present GEM's graph stylesheet language. As described earlier, a graph stylesheet consists of *visualization rules* and *edit operations*: Visualization rules describe what kind of visualization model elements should be produced for which structures (subgraphs) of the application model. All visualization rules together create the complete visualization graph for the application model. The semantics of stylesheet application is non-destructive, i.e., application model subgraphs are not replaced by their visualization model counterparts, but instead application and visualization model coexist in one common graph. All visualization model elements are connected by special RepresentedBy edges to all those (possibly many) application model elements they represent. In conjunction with the currently selected widgets, an edit rule can use the RepresentedBy edges to determine the affected application model elements and apply the appropriate changes.

### 5.1 Visualization rules

As previously mentioned, visualization rules follow a restricted layering approach, where each rule is applied until no further occurrences exist, after which processing continues with the next rule. Since the visualization of large graphs requires a large number of rule applications, some restrictions have been defined to allow an optimized processing, and also help to make the visualization rules "safe": First, visualization rules must not modify, delete or create any elements outside the visualization model. Further, visualization rules may only *create* new elements in the visualization model and may not overwrite attribute values. These restrictions are needed to allow for the selective updating described in Section 7.2. Visualization rules may only be applied once per occurrence. This is enforced by an implicitly created (negative) application condition that corresponds to the newly created elements on the RHS. Moreover, negative application conditions must not refer to elements that could have been created by a previous application of the same rule. That way, we assure that the sequence in which a visualization rule is applied to its occurrences does not change the outcome of the stylesheet. It also allows optimizing the search for occurrences, as we do not have to recompute them after each rule application.

To illustrate the basic use of visualization rules, we defined a *default stylesheet* shown in Figure 5, which visualizes any application graph in a 1:1 fashion. The XML representation of the stylesheet was omitted due to space constraints. The stylesheet consists of two rules: displayNodes visualizes the application model nodes: Each one is represented by a Box widget, carrying the node's type attribute as label. To connect application and visualization model, we add a RepresentedBy edge between the application model node and the

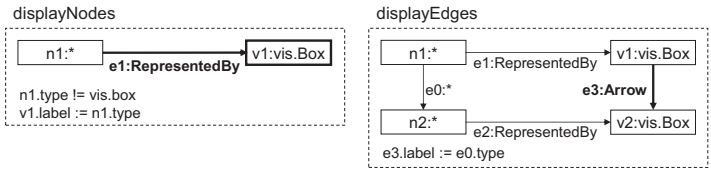



Figure 5: Visualization rules for a generic labeled digraph, integrated graphical syntax.

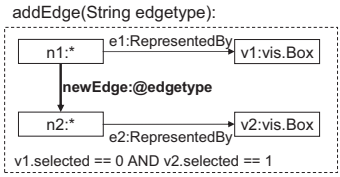



Figure 6: A simple edit operation (integrated representation)

visualization model Box node. Due to the layered application of visualization rules, the displayNodes rule will be applied on all graph nodes. To make sure that we do not create boxes for other boxes, we add an attribute condition indicating that the node *n1* must not be of type vis.Box. The second visualization rule displayEdges looks for two nodes which are connected by an arbitrary edge. For each such pair of nodes, a new Arrow edge is created between their Box nodes, taking the type of the connecting edge as label. Note that we can omit tests to guarantee that *n1* and *n2* are actually application model elements, since the first rule already makes sure that only application model nodes will have RepresentedBy edges to box nodes. This simple stylesheet will already reproduce the graph's structure and typing. In Section 6 we will demonstrate how a more complex set of visualization rules can be used for a customized representation of proprietary metadata.

### 5.2 Edit operations

For metadata manipulation, a stylesheet defines a set of edit operations. An edit operation consists of a number of production rules (edit rules) and a control flow specification using the control flow constructs (*WAPP*, *CAPP*, *SAPP*) from Section 4.2. A user modifying the graph through an edit operation in general wants the operation to be applied on a specific set of elements (e.g., a specific table or column) at a time. To indicate the place of rule application, the user selects the appropriate widgets in the editor pane. Their selection state is propagated to the underlying visualization graph, where it can be referred to by edit rules. Besides selecting the widgets of affected elements, often user input is needed for the rule, e.g., the name of a new table or element. Edit operations can therefore specify *user parameters*. When the edit operation is activated, the user can enter the appropriate information.

Figure 6 shows a very basic edit operation with only one edit rule that adds an edge between two nodes selected by the user. The rule specifies a user parameter edgetype, which is used in the created part to set the type attribute of the new edge. Note the reference

to the selected state of the visualization model nodes: A selected value $< 0$ indicates that the element is not selected, while values $\geq 0$ indicate the sequence of selection. So the rule adds the edge pointing from the node selected first to the node selected second. Also note that the edit rule does *not* add an Arrow edge between the two Box nodes representing the application model nodes, as updating the visualization model is left to the visualization rules. The method for updating will be explained in Section 7.2.

## 6 Example scenario

The development of GEM was initiated by the practical needs in the projects PALADIN [Gö05] and Caro [Stu06] of our group. To demonstrate GEM's usefulness as a generic metadata editor, we will present a stylesheet that displays arbitrary models represented in the internal representation of the PALADIN metamodel architecture and allows the editing of semantic correspondences between these elements. We will use a simplified version of the stylesheet to illustrate GEM's customizability beyond the basic examples shown so far. A first step to integrate GEM with PALADIN was to provide a serialization of models given in PALADIN's internal object-oriented representation to GXL graphs. Objects and values of atomic fields map naturally to GXL nodes and their attributes, while values of complex fields themselves map to nodes. An object's class is stored as the type label. References between objects and complex fields are mapped to edges. Since many multi-valued references are ordered, we use a position attribute on edges to preserve the ordering. This very basic mapping is already able to provide full round-trip capability, i.e., PALADIN models can be losslessly reconstructed from the GXL representation.

PALADIN uses a *Core metamodel* to capture common aspects of different actual metamodels. Elements of concrete metamodels inherit from the Core elements. This allowed us to start with a single set of rules that uniformly displays arbitrary schemas (e.g., SQL or XML Schema models), since it is sufficient to refer to the common base classes. For example, SQL tables and columns as well as XML elements and attributes are all subclasses of the Core class Feature. In addition, concrete relationships like hasColumn (between table and column) or subelement (between an element and its subelements and attributes) are derived from the Core relationship content. So only two visualization rules were sufficient to display any PALADIN model. A more elaborate version of the stylesheet can easily provide metamodel-specific representations of schemas. In the current GEM prototype, node type inheritance is implemented via a user-defined function *subclassOf* which takes the full-qualified names of two node types A and B as input, and returns true if A is the same class or a subclass of B, or false otherwise. We use this function in the rule's application conditions.

Figure 7 shows the displayFeatures rule to create properly labeled widgets for PALADIN Features. Unlike in the default stylesheets' displayNodes rule, a Group node instead of a plain Box is created for each Feature node. A Group can contain other widgets and display them in a hierarchical fashion. By using two application conditions we make sure that the node is a subclass of Feature and no subclass of Match (matches are displayed by a separate rule). The Group's label is created by concatenating the Feature's actual type (e.g., Column, Table, etc.) and the name attribute common to all PALADIN classes.

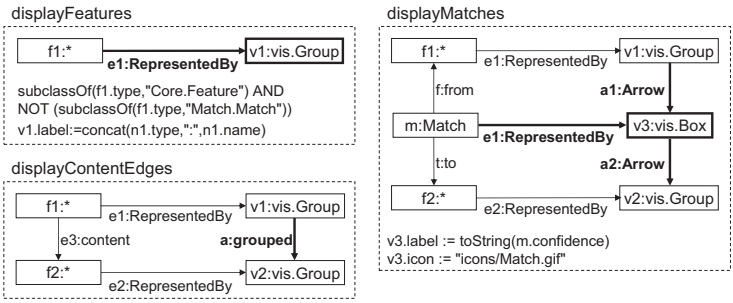



Figure 7: The visualization rules for PALADIN models and semantic correspondences.

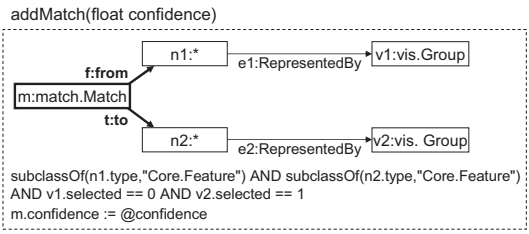



Figure 8: An edit rule to add a match between two PALADIN model elements

The displayContentEdges rule is based on the displayEdges rule and searches for pairs of nodes that are already represented by a Group widget and connected by a content edge. We can directly refer to the content edge type in the stylesheet, since we decided for our model serialization to only create edges for base references, i.e., there are no distinct edges for references that are derived from content, like hasColumn. For each content edge, a new grouped edge is created that indicates membership of its target node in the source group.

The third rule displayMatches creates the representation for Match nodes and their to and from edges that indicate which two features the match connects. It searches for pairs of nodes that are already represented by a Group each and are connected by a Match node. The Match node itself is represented by a newly created Box, which is distinguished from other boxes representing schema elements by adding an icon. The Match node's confidence attribute is set as a label. The to and from edges are represented by unlabeled Arrows. Note how both Arrows now point into the direction of the to edge, so that the match's direction can be determined visually without edge labels.

To create new matches, an addMatch (Figure 8) edit rule has been defined. Like the createEdge rule, it refers to the selection state of the visualization elements and adds a Match node whose from edge points to the node selected first, and whose to edge points to the node selected second. The match's confidence is queried from the user via a user parameter. Figure 9 shows a schema matching session with GEM using only this simply stylesheet. The result of a somewhat more elaborate stylesheet for editing entity-relationship models is shown in Figure 10.

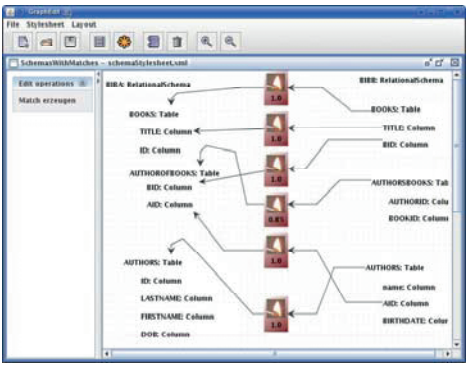


Figure 9: A schema matching session.

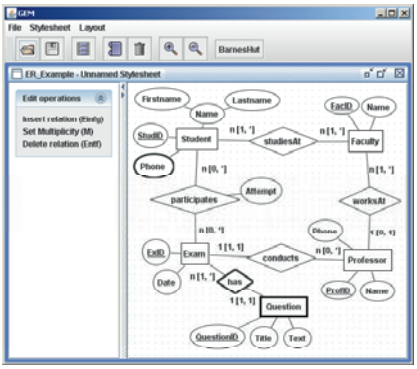


Figure 10: An ER-modeling session.

## 7 Implementation and performance

In this section, we discuss the most interesting aspects of our GEM prototype. Also, we give results on GEM performance for different sized application models. The GEM prototype was developed as part of a diploma theses [Jö07] in our group. For the display of graphs we use the JGraph[2] library. It also provides the basic user interaction functionality, like automatic layout, moving and resizing nodes, rerouting edges etc.

### 7.1 Using a relational database for graph transformations

As previously mentioned, the graph transformation engine used by GEM has been built in-house taking advantage of the features of relational database systems. The main rationale behind a custom implementation was the insufficient performance or maturity of the few existing publicly available graph transformation engines.

For each graph loaded into GEM, a new schema with four relations is created, to store nodes, edges, node attributes and edge attributes. The schema definition is straightforward, with the exception of the `Edge` table, which is denormalized. Besides the edge information, it also contains the complete information for nodes (their id, type and current selection status) to improve query performance by reducing the number of joins needed. A separate `Node` table is still needed to represent unconnected nodes.

**Query generation** For each production rule of a stylesheet, an SQL query based on the rule's LHS (which corresponds to the union of the elements in the preserved and deleted sets when shown in our integrated representation) and application conditions is generated, as well as a number of `DELETE`, `INSERT`, and `UPDATE` statements for the RHS. To illustrate query generation, the query generated for the displayEdges rule of the default stylesheet from Figure 5 is shown in Figure 11.

For each edge in the LHS, a reference to the `Edge` relation is added to the `FROM` clause

[2] http://www.jgraph.com

```
1 SELECT e0.id AS e0, e1.id AS e1, e2.id AS e2, e1.dest AS v1, e0.src AS n1, e2.src AS n2, e2.dest AS v2
2 FROM EDGE e0, EDGE e1, EDGE e2
3 WHERE e1.edgetype = 'vis:RepresentedBy' AND e1.desttype = 'vis:Box' AND
4     e2.edgetype = 'vis:RepresentedBy' AND e2.desttype = 'vis:Box' AND
5     e0.src = e1.src AND e2.src = e0.dest AND
6     NOT e1.dest IN (e0.src, e2.src, e2.dest) AND NOT e0.src IN (e2.src, e2.dest) AND
7       NOT e2.src IN (e2.dest) AND NOT e0.id IN (e1.id, e2.id) AND NOT e1.id IN (e2.id) AND
8     NOT EXISTS (SELECT * FROM EDGE e3
9       WHERE e3.edgetype = 'vis:Arrow' AND e3.src = e1.dest AND e3.dest = e2.dest AND NOT e3.id IN (e0.id))
```

Figure 11: The query generated for the LHS of the *displayEdges* rule of Figure 5

(line 2), and the edge's ID is added to the `SELECT` clause (line 1). If the edge specifies a type test, an appropriate predicate is added to the `WHERE` clause (lines 3, 4). For each node in the LHS, the query generator determines its in- and outbound edges. If at least one such edge exists, the node's ID is already available in this edge's reference to the edge relation. Then simply a reference to the corresponding `src` or `dest` column is added to the `SELECT` clause. If the pattern node has several connecting edges in the LHS (i.e., its ID occurs in several edge relation references), we have to make sure that all edges refer to the same host graph node. So the IDs of source and target nodes of edges connecting to the same node are tested for equality in the `WHERE` clause (line 5). Only if the pattern node is isolated (i.e., not connected via edges to other elements of the LHS), a reference to the `Node` relation is added to the `FROM` clause, as well as a type test predicate to the `WHERE` and the node ID to the `SELECT` clause (not in this example). As discussed in Section 4.1, we are by default looking for subgraphs of the host graph that are *isomorphic* to the rule's pattern graph, i.e., we have to make sure that no single host graph element matches several rule elements. So for each node and edge, we conjunctively add an *isomorphic* predicate to the `WHERE` clause that ascertains that its ID is not the same as the IDs of one of the other pattern elements (lines 6–7 and 9). For structural application conditions, conditional subqueries are generated in the `WHERE` clause. For example, lines 8–9 show the implicit negative application condition generated based on the rule's *created* part. The algorithm to create the subquery is the same as the one for the main query, just with a simplified `SELECT` clause, and with outer references for those elements that have been identified with LHS elements (line 9). For each node and edge attribute occurring in an application condition, a join with the `Node` or `Edge` relation reference and the `NodeAttribute` or `EdgeAttribute` relation is added to the `FROM` clause (not in this example). An join condition and the application condition itself are conjunctively added to the `WHERE` clause.

**Processing pattern query results** Each row in the result set of a pattern query represents a single occurrence in the host graph. One transformation step selects one of these occurrences and modifies the host graph appropriately. If the rule is to be applied repeatedly, as is the case for visualization rules, in general it would not be possible to continue with the results obtained from the first issuing of the query, since the modification could have invalidated previously valid occurrences or created new possible occurrences: (1) nodes or edges of the occurrence or an application condition could have been deleted, or (2) new elements could be created so that a negative application condition would now fail. Also, (3) attribute values could have been modified causing an attribute condition to fail.

However, with the restrictions defined for visualization rules (Section 5.1) none of these cases can occur: Visualization rules must not delete any elements, eliminating case (1). Their negative application conditions must not refer to elements possibly created by a previous application of the same rule, so case (2) is avoided. In addition since attributes once initialized must not be modified, case (3) cannot occur as well. So for each visualization rule, its respective query has to be issued only once, instead of reissuing it after applying the RHS on a single occurrence. Since the actual number of occurrences for a rule depends on the rule and on the structure of the application graph, we can give no general estimate on the improvement due to this optimization. However, for most real-world stylesheets and application graphs, the number of occurrences scales with the size of the graph, which is several orders of magnitude larger than the number of rules or queries. For each result tuple, the `INSERT`, `UPDATE`, and `DELETE` statements prepared for the rule's RHS are executed, parameterized with the node and edge IDs taken from the result tuple. To set the type and selected state of the source and target nodes of a new edge, subqueries are used. Once all tuples of a visualization rule are processed, the query for the next visualization rule is issued. When all rules are processed, the editor can interpret the visualization graph and display it to the user.

The statement generation is the same for edit rules. The control flow statements SAPP, CAPP, and WAPP are interpreted by the editor, who will, depending on the outcome of rule applications, choose the query for the next rule and move forward in the control flow. Since the restrictions imposed on visualization rules do not apply for edit rules, the optimization described for visualization rule processing can not be applied to a single edit rule surrounded by a WAPP statement. Instead, the query has to be issued again after one occurrence is processed. Since edit rules are usually only applied selectively to small parts of the graph, this does not imply a noticable performance penalty.

### 7.2 Selective updating of the visualization

As described in Section 2.3, edit rules are not supposed to modify the visualization graph, but only the application graph. Therefore, once an edit operation applied its changes on the application model, the visualization rules have to be applied again, so that the representation reflects the changes. A naive implementation would simply discard the existing visualization graph and re-apply the stylesheet on the entire application graph. This is undesirable for two reasons: First, the application of a stylesheet is a costly operation which can take a noticeable amount of time. Second, simply discarding the visualization graph would also loose all those kinds of information that were not derived from the application model but result from *non-application relevant edit operations* (Section 2.1), like moving or resizing of widgets or adding bends to edges etc. Given the effort needed for a manual layout, this information has to be maintained as completely as possible even in the presence of edits. We therefore employ a *selective updating* of the visualization model, which only includes the elements that were affected by the edit operation.

In a first step, the effects of visualization rule applications that were possible in the original application graph, but are no longer applicable in the edited graph, have to be undone. Afterwards, the new application graph has to be tested to find out if new visu-

alization rule applications are possible. To be able to trace rule applications, we use *rule application objects* (RAOs). They contain a reference to the rule and the occurrence used, as well as information on the nodes, edges and attributes that have been created or set by the rule application. Following the observer pattern, a RAO registers itself as observer to certain events on those elements of the application model that are part of its occurrence: (1) If a *deletion* event occurs on the nodes and edges of a RAO's occurrence, the rule is no longer applicable and is undone using the information in the RAO on created objects and set attribute values. (2) If an attribute used in an *attribute* condition *changes*, the rule has to be retested for applicability. If it is not applicable, the rule is undone. (3) If the value for an attribute assignment on the rule's RHS is *derived* from one or several attributes in the LHS, the value has to be recalculated if one of the base attributes changes.

Besides observing existing objects, negative application conditions that were fulfilled when the rule was first applied might no longer hold after an edit operation. Since these objects were non-existant, the RAO cannot register itself to any concrete object, but only to abstract insertion or deletion events of nodes and edges that have the same type as one of the elements in one of the rule's NACs. Since NACs can also contain attribute conditions, the RAO also has to register itself to any modifications of attributes that belong to a node or edge which has the same type as the node or edge in the NAC and have the same attribute name. For positive application conditions (PACs), similar considerations hold, but here we are interested in *deletion* modification events on nodes, edges, and their attributes.

### 7.3 Performance

In the conceptualization phase for GEM, we evaluated existing graph transformation systems. The only system that seemed both mature enough and did not depend on legacy system environments was the AGG system [Tae99]. We evaluated its performance by measuring the time required for applying a basic stylesheet consisting of two visualization rules. Our measurements discouraged the use of AGG as GEM's underlying engine. We therefore chose to implement our own GTS, building on a relational DBMS to do subgraph search and transformation. We first implemented a basic version of the DBMS-based GTS and tested it against the same input schemas using three different underlying database systems, two open-source systems (Derby and PostgreSQL), and IBM's DB2. All three systems scaled nearly linearly with application graph size and number of occurrences, with the commercial system outperforming its open-source contenders by an approximate factor of two. For small- to medium-sized graphs, both showed acceptable matching times, DB2 outperforming AGG by a factor of up to 120. Encouraged by these promising results, we decided to go for a DBMS-based solution.

While the initial measurements helped us estimate which degree of performance and scalability to expect, they were no final indication of the actual performance of the editor. We therefore ran a series of benchmarks of the latest version of GEM against four different DBMS (now also including the embedded H2 DBMS), application graphs of different sizes, and stylesheets of different complexity. The *default* stylesheet is the same as presented in Section 5. The *PALADIN* stylesheet is a slight variation of the stylesheet from Figure 6, modified to provide a representation that better suits SQL metadata. For a realis-

| Stylesheet | Graph size | DB2 | Derby | PostgreSQL | H2 |
|---|---|---|---|---|---|
| default | 157N/239E | 1.4 | 2.3 | 2.0 | 0.7 |
| default | 237N/405E | 1.7 | 3.0 | 1.6 | 1.0 |
| default | 599N/1026E | 3.0 | 4.7 | 3.5 | 2.6 |
| default | 1068N/2088E | 4.7 | 9.5 | 7.5 | 4.6 |
| default | 1785N/3723E | 8.1 | 14.2 | 11.5 | 7.7 |
| PALADIN | 6T/67C | 1.3 | 2.2 | 1.4 | 0.4 |
| PALADIN | 12T/111C | 1.6 | 2.6 | 2.0 | 0.5 |
| PALADIN | 25T/294C | 3.1 | 5.4 | 4.1 | 1.2 |
| PALADIN | 50T/602C | 6.6 | 8.4 | 8.4 | 2.2 |
| PALADIN | 100T/1071C | 9.6 | 14.6 | 12.0 | 3.4 |

Table 1: Stylesheet application time (in seconds) on different application graph sizes and DBMSs.

tic example scenario, we chose the schema of the open-source OpenExchange groupware system[3], scaled to five different sizes. We ran the benchmarks on a typical desktop environment, a 2GHz Core Duo machine with 2GB RAM and locally installed DBMSs. We repeated each of the individual tests three times and averaged the results.

The first part of Table 1 shows the results of applying the default stylesheet to the five different graphs using the four DBMSs. We see that all perform adequate for small and medium sized graphs, with DB2 maintaining its lead over Derby and PostgreSQL. A pleasant surprise was the H2 database, which – unlike Derby – appears to take considerable advantage of the reduced communications overhead of an embedded DBMS. This is particular obvious both in the significantly higher base response time of the other three systems and in H2's better scalability with increasing graph size and occurrence count. The second part of Table 1 shows the results for the PALADIN stylesheet: H2 extends its lead, as it apparently benefits from the reduced insert and update load of this stylesheet, which aggregates metadata and thus creates less nodes and edges for the visualization model. While H2's snappy response times make it the ideal choice for our example stylesheets, additional experiments (not presented due to space constraints) have shown its performance to drop considerably once rules (and thus queries) get more complex. This behavior can be explained with known limitations of H2's current cost-based optimizer. So for larger graphs and complex rules, DB2 is usually the better choice, as here its principal disadvantages and its significant schema setup overhead during graph import are offset.

## 8 Related Work

The basic principle we follow with our approach is the separation of model and view in a generic way. This is a pervasive paradigm in computer science. A similar approach to ours, although limited to displaying and editing XML files, is the XMLmind XML Editor (XXE)[4]. It uses Cascading Stylesheets with some proprietary extensions to display XML files. For editing, the user can insert and delete elements or change attribute values. XXE makes sure that the resulting document is always valid and corresponds to its DTD or XML

[3] http://www.open-xchange.com/

[4] http://www.xmlmind.com/xmleditor

schema definition. For more complex operations it is possible to define element templates, the equivalent to the edit operations in GraphEdit. Another system that inspired the development of GEM are the Graph Stylesheets used in IsaViz[5]. They are used to style RDF graphs, but are not able to abstract from the structure of the underlying RDF graph, as they just provide means to change color, shape and fonts of the nodes. The Eclipse Modeling Framework (EMF)[6] supports the generation of basic editors for object-oriented models. However, representation is limited to a tree-like structure following the containment relationships among objects. Further, the default editors can only perform atomic editing operations (creating objects and references between them, setting values of fields etc.). Any kind of higher-level editing functionality requires complex customizations through Java code. The Graphical Modeling Framework (GMF)[7] builds on EMF and allows the creation of truly graphical editors. However, the graphical definition models used by GMF are less expressive than GEM stylesheets. Consequently, the shape of the visualization graph is largely predetermined by the structure of the underlying EMF application model and therefore does not allow the degree of abstraction that is possible with GEM. Any customizations of a generated GMF editor require Java coding.

The idea to implement graph transformations with a relational database system is in itself not new. Previous implementation attempts have been criticized for severe performance deficits: In [VSV05], the authors compare a number of native GTSs (among them the aforementioned AGG) with an RDBMS implementation. According to their initial measurements, performance of the DBMS solution was overall far inferior to the native GTSs. These findings not only conflicted with our observation that even in a prototypical state, GEM's graph transformation engine significantly outperforms the AGG system by several orders of magnitude, but have also since been revised by the authors in [VFV06].

## 9 Conclusions and Outlook

In this paper, we presented GEM, a visualization tool and editor for arbitrary metadata. To internally represent the metadata to be visualized and edited, we use a model based on attributed, typed multigraphs, which is general enough to losslessly represent any kind of metadata. Visualization of the metadata is customized via user-provided stylesheets, which allow for a wide variety of different visual representation, like arranging metadata elements in – possibly nested – tables or in trees. In order to hide details or to focus on a special aspect of the model, stylesheets can perform aggregations that abstract from the concrete metadata representation. Also, one can visualize and edit heterogeneous metadata in an integrated way. Both visualization rules and edit operation are based on the concept of graph transformations.

Our metadata editor is not only useful for editing schemas and matches, but supports any data which can be reasonably represented as a graph. In the semantic web community, visualizing and editing ontologies and RDF data in an adequate and user-friendly manner is still an open problem. Here, GEM can possibly be of great value, because it allows the

[5] http://www.w3.org/2001/11/IsaViz/
[6] http://www.eclipse.org/modeling/emf/
[7] http://www.eclipse.org/modeling/gmf/

fast creation of domain-specific visualization and editing tools for arbitrary ontologies, and allows to "'mix"' different ontologies within one document. Other possible applications include networking problems, mind maps, or any other area that uses graphs to represent its data. The editor's underlying graph transformation system can also be used independently to do reasoning on the application graph, or to perform model-to-model transformations. Some open issues remain: Currently, GEM supports node and edge type inheritance only by user-defined functions. This is a flexible mechanism and allows the adaptation of the type model to the respective needs. Introducing type graphs into GEM would make type inheritance a first class concept, allowing for better consistency checking in application models. These aspects, and a much more powerful GT language with a more concise textual syntax are supported by an advanced GT engine developed in the context of the PALADIN project. We intend to use this engine in future versions of GEM.

## References


[BFG95] D. Blostein, H. Fahmy, and A. Grbavec. Practical use of graph rewriting. Technical Report 95-373, CDN, 1995.

[BHP00] Phillip A. Bernstein, Alon Y. Halevy, and Rachel A. Pottinger. A vision for management of complex models. *SIGMOD Rec.*, 29(4):55–63, 2000.

[CH03] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. volume OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.

[Gö88] Herbert Göttler. *Graphgrammatiken in der Softwaretechnik: Theorie und Anwendungen*, volume 178 of *Informatik-Fachberichte*. Springer-Verlag, Berlin, 1988. Venia Legendi Thesis (Habilitation).

[Gö05] Jürgen Göres. Towards Dynamic Information Integration. In *Data Management in Grids*, number 3836 in LNCS, pages 16–29, 2005.

[GD07] Jürgen Göres and Stefan Dessloch. Towards an Integrated Model for Data, Metadata, and Operations. In *BTW 2007*, 2007.

[Hec06] Reiko Heckel. Graph Transformation in a Nutshell. *Electr. Notes Theor. Comput. Sci.*, 148(1):187–198, 2006.

[Jö07] Thomas Jörg. Entwicklung eines regelbasierten Editors für Graphmodelle (Development of a Rule-based Editor for Graph Models). Master's thesis, University of Kaiserslautern, February 2007. in German.

[KR90] Kreowski and Rozenberg. On structured graph grammars. 1990.

[MRB03] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD 2003*, 2003.

[SD07] Boris Stumm and Stefan Dessloch. Change Management in Large Information Infrastructures – Representing and Analyzing Arbitrary Metadata. In *BTW 2007*, 2007.

[Stu06] Boris Stumm. Change Management in Large-Scale Enterprise Information Systems. In *EDBT Workshops*, pages 86–96, 2006.

[Tae99] Gabriele Taentzer. AGG: A Tool Environment for Algebraic Graph Transformation. In *AGTIVE*, pages 481–488, 1999.

[VFV06] Gergely Varró, Katalin Friedl, and Dániel Varró. Implementing a Graph Transformation Engine in Relational Databases. *Software and System Modeling*, 5(3):313–341, 2006.

[VSV05] Gergely Varro, Andy Schurr, and Daniel Varro. Benchmarking for Graph Transformation. In *VLHCC 2005*, pages 79–88, Washington, DC, USA, 2005.

[WKR01] Andreas Winter, Bernt Kullbach, and Volker Riediger. An Overview of the GXL Graph Exchange Language. In *Software Visualization*, pages 324–336, 2001.

# An Evolution-based Approach for Assessing Ontology Mappings - A Case Study in the Life Sciences

Andreas Thor[1], Michael Hartung[2], Anika Gross[2], Toralf Kirsten[2,3], Erhard Rahm[1,2]

[1]Dept. of Computer Science, University of Leipzig
[2]Interdisciplinary Centre for Bioinformatics, University of Leipzig
[3]Institute for Medical Informatics, Statistics and Epidemiology, University of Leipzig
{thor, rahm}@informatik.uni-leipzig.de
{hartung, gross, tkirsten}@izbi.uni-leipzig.de

**Abstract:** Ontology matching has been widely studied. However, the resulting ontology mappings can be rather unstable when the participating ontologies or utilized secondary sources (e.g., instance sources, thesauri) evolve. We propose an evolution-based approach for assessing ontology mappings by annotating their correspondences by information about similarity values for past ontology versions. These annotations allow us to assess the stability of correspondences over time and they can thus be used to determine better and more robust ontology mappings. The approach is generic in that it can be applied independently from the utilized match technique. We define different stability measures and show results of a first evaluation for the life science domain.

## 1 Introduction

Ontology matching aims at identifying semantic correspondences between concepts of different ontologies. It has been recently studied in diverse scientific and commercial application domains and various match approaches and prototypes (see e.g., [RB01, KS03, Do06, ES07] for surveys) have been developed. The ontology mappings determined by ontology matching are used in many ways, such as for enhanced data analysis or data integration. However, ontologies underlie continuous modifications because domain knowledge or community agreements represented by ontologies often change over time. Typical changes include additions of new elements, deletions of outdated elements, modifications of element attributes, such as names and descriptions, and structural changes, such as element moves within an is-a hierarchy. Therefore new ontology versions are released periodically and, thus, ontology mappings need to be re-created or updated accordingly. This is also necessary when new versions of secondary sources used for matching become available, such as instance sources, thesauri and their associations to ontologies.

As a result of evolving ontologies and secondary sources the generated ontology mappings can be rather unstable, i.e., the similarity between two given concepts may vary significantly over several versions. As an example from the life science domain, Figure 1 shows the history of concept similarity for two selected correspondences over 21 mapping versions. The considered mapping relates two sub-ontologies of the Gene Ontology

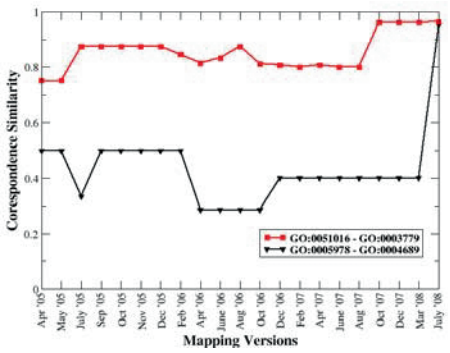


Figure 1: History of similarity values for Gene Ontology correspondences

[Go04], Molecular Functions and Biological Processes, and was determined by an instance-based match technique which derives the similarity of two concepts from their associations to the same instances (see [KTR07] for details). Both correspondences have a relatively high similarity value of 0.95 in the last mapping version but significantly differ in their history. The first correspondence (GO:0051016-GO:0003779) is more stable and at a higher level of similarity than the second correspondence (GO:0005978-GO:0004689). Since the similarity value of the latter correspondence jumps from 0.4 to 0.95 its correctness might be questionable and should be further verified. This observation of fluctuating concept similarities is mainly influenced by evolving ontologies, modifications on secondary sources as well as changes of associations between them and was quantitatively evaluated in [HKR08].

In order to obtain ontology mappings of high quality, previous work (e.g., [ADMR05]) has shown that a combination of different match approaches can be helpful. However, current ontology match approaches do not take into account that ontologies evolve. They only utilize information about the specific ontology versions which are matched with each other. Studying the correspondence history for past ontology versions is orthogonal to a match technique or matcher combination. One aim is to utilize the stability of correspondences to identify robust mappings, while unstable correspondences may be subject to a focused manual validation. To that end we make the following contributions:

- We propose a generic approach to annotate generated ontology mappings independently from the used match approach, i.e., the computation of the similarity values. Besides the similarity value, the mapping annotation shows the stability for each correspondence of the mapping by taking its historic changes into account. Hence, the correspondences can not only be ranked and evaluated using the similarity value but also by considering the computed stability values.
- We define two stability measures quantifying the evolution of similarity values for a given concept correspondence. While the average stability considers changes during evolution steps, the weighted maximum stability inspects stability w.r.t. the current similarity value.
- We show results of a first evaluation using the proposed approach in an example domain, the life sciences. In particular we use the stability measures to classify the correspondences of a mapping into several groups, such as accepted, candidates, and questionable.

The rest of the paper is organized as follows. In Section 2, we introduce the approach and define the stability measures. Section 3 shows the evaluation results for life science applications. We discuss related work in Section 4 before we conclude in Section 5.

## 2 Evolution-aware ontology mappings

In this section we first introduce a versioning scheme that is used for the definition of version-based correspondence similarity. We then motivate and define two stability measures that characterize the correspondence similarity over several versions.

### 2.1 Versioning scheme and correspondence similarity

Since different ontologies as well as secondary data are typically developed independently, new versions of different ontologies *A* and *B* are usually not introduced at the same time. Figure 2 schematically illustrates the evolution process for two ontologies *A* and *B* and a secondary data source *D* over time. Each diamond represents a new ontology version and each cylinder indicates changes in the (instance) data of *D*.

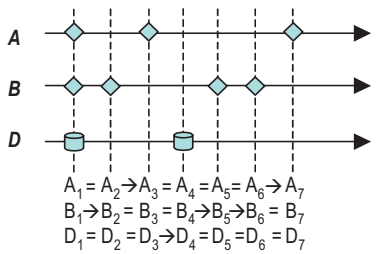


Figure 2: Schematic illustration of the versioning scheme

We start with two initial versions $A_1$ and $B_1$. After a while a new version $B_2$ comes up whereas $A_1$ is still the current valid version for *A*. We bridge this gap by introducing a *virtual version* for *A* ($A_2$) that is, of course, equivalent to $A_1$. By doing so we may only consider matching ontologies of the same virtual version, e.g., $A_2$ and $B_2$. Note from Figure 2 that new versions for *A* and *B* need not be introduced alternately but an ontology version ($A_3$) may outlast several versions of the other ontology ($B_5$ and $B_6$). Since changes in the secondary data source *D* may also affect the results of ontology matching, we additionally introduce new virtual versions when such changes occur (see $A_4$ and $B_4$). In summary, during an evolution step $i \rightarrow i+1$ from a virtual version *i* to *i+1* at least one of the participating ontologies (*A* or *B*) or a secondary data source (*D*) used for matching has changed.

In the following, we consider concepts *a* and *b* from the ontologies $A_i$ and $B_i$ (of a virtual version *i*), respectively, and a match technique *m*. We do not make any assumption about the matcher to keep our approach generally applicable. For the similarity computation a matcher may not only use the concepts *a* and *b* itself, e.g., the concept names, but may also utilize other parts of the involved ontologies $A_i$ and $B_i$, e.g., their structure or associated instances of a source $D_i$.

We denote the similarity between *a* and *b* as $sim(a,b,m \mid A_i,B_i)$ with $a \in A_i$ and $b \in B_i$, i.e., the concept similarity determined by a matcher *m* between *a* and *b* is defined in consideration of the ontologies in their versions $A_i$ and $B_i$. Since ontology evolution may be triggered by concept additions and deletions, it is not given that a concept *a* is always contained in all versions $A_i$ ($i$=1…$n$). Thus, we further define $sim_i(a,b,m)$ as the *correspondence similarity* between two concepts *a* and *b* w.r.t. a version *i*.

$$sim_i(a,b,m) := \begin{cases} sim(a,b,m|A_i,B_i), \text{if } a \in A_i \wedge b \in B_i \\ 0 \quad\quad\quad\quad , \text{otherwise} \end{cases} \in [0,1]$$

If (at least) one of the two concepts does not appear in the considered versions the correspondence similarity $sim_i(a,b,m)$ is set to the minimal value of 0. This definition allows for an easy similarity comparison between two concepts over different virtual versions.

### 2.2 Definition of stability measures

We aggregate the different similarity values of a given correspondence by calculating stability measures which consider the similarity value of a current version $n$ as well as the $k>0$ previous versions. However, the maximum number $kmax$ of available previous versions is limited by both the correspondence $(a,b)$ and the applied matcher $m$. Obviously we may only consider versions from the time when both concepts $a$ and $b$ have been appeared together in the involved ontology versions for the first time. Moreover, we further restrict $kmax$ to the first version with $sim_i(a,b,m)>0$, i.e., we determine the first version where matcher $m$ calculates a positive similarity value for the correspondence $(a,b)$. Thereby the "initial jump" from 0 to a positive similarity value is not considered for any stability calculation because we do not want to penalize this as instability. Hence, $kmax$ which will be used in later stability definitions is defined as follows:

$$kmax_n(a,b,m) = \max_{k=1..n-1}\left(\left\{k \middle| sim_{n-k}(a,b,m) > 0\right\}\right)$$

Note that this definition is only well-defined if there is at least one correspondence with $sim_i(a,b,m)>0$ within the previous $k$ versions. However, this is not a relevant restriction because correspondences with $sim_i(a,b,m)=0$ for all $i<n$ do not carry evolutionary information and, thus, can not be annotated with any stability values.

In the following, we define two stability measures *average* and *weighted maximum*. The first stability captures the average fluctuation of the similarity values over the last $k$ evolution steps. For a correspondence $(a,b)$ the *average stability* is defined as follows:

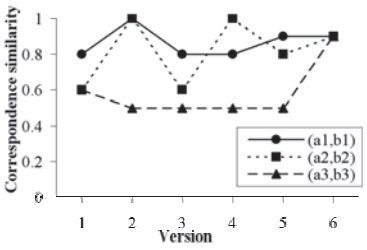
$$stabAvg_{n,k}(a,b,m) = \begin{cases} 1 - \frac{1}{k} \cdot \sum_{i=n-k}^{n-1} \left| sim_{i+1}(a,b,m) - sim_i(a,b,m) \right| & \text{, if } k \le kmax_n(a,b,m) \\ stabAvg_{n,kmax_n(a,b,m)}(a,b,m) & \text{, otherwise} \end{cases} \in [0,1]$$

The measure captures both small as well as big differences of the correspondence similarity (as determined by $m$) as indicators for an unstable correspondence $(a,b)$. Hence, we interpret $(a,b)$ as stable if only few and small similarity changes occur during the evolution process. To that end, the absolute differences of correspondence similarities of consecutive ontology versions are summarized for all evolution steps from version $n$-$k$ to $n$. This sum is normalized by the total number of evolution steps ($k$) giving a value in a 0 to 1 range because each of the $k$ evolution steps contributes a similarity change between 0 and 1. Finally, we compute 1 minus this normalized sum to have an average stability value of 1 (0) for perfect stability (complete instability).

For a correspondence $(a,b)$, a current version $n$, and a matcher $m$ we further define the *weighted maximum stability* for the last $k$ evolution steps as follows:

$$stabWM_{n,k}(a,b,m) = \begin{cases} 1 - \max\limits_{i=1\ldots k}\left[\dfrac{|sim_n(a,b,m) - sim_{n-i}(a,b,m)|}{i}\right] & \text{, if } k \le kmax_n(a,b,m) \in [0,1] \\ stabWM_{n,kmax_n(a,b,m)}(a,b,m) & \text{, otherwise} \end{cases}$$

The idea of weighted maximum stability for a correspondence $(a,b)$ is to identify how close similarity values are to the current similarity $sim_n(a,b,m)$, i.e., we focus on version $n$ to assess the stability within the last $k$ evolution steps. For a version $n$-$i$ we therefore consider the distance of $sim_{n-i}(a,b)$ to the current similarity $sim_n(a,b)$. This distance is normalized (weighted) by the number of evolution steps ($i$) and, thus, differences in later versions have a higher impact than in previous versions. This allows us to inspect trends, i.e., to evaluate if the evolution of a correspondence similarity is constant, slightly increasing (decreasing) or gamboling within the last versions. The maximum value over all considered versions $i$ defines the maximum deviation and indicates the stability. Again, we compute 1 minus this max-value so that complete stability (instability) is equal to 1 (0). If and only if all previous similarities $sim_i$ of the considered correspondence $(a,b)$ are equal to the current similarity $sim_n$, we calculate a perfect stability with $stabWM$=1. Contrary, the minimal weighted maximum stability of 0, i.e., complete instability, is obtained if and only if $|sim_n - sim_{n-1}| = 1$, i.e., the correspondence similarity has changed from 0 to 1 (or vice versa) during the last evolution step $n\text{-}1 \rightarrow n$.




| | $stabAvg_{6,5}$ | $stabWM_{6,5}$ |
|---|---|---|
| $(a_1,b_1)$ | 0.9 = 1-(0.2+0.2+0+0.1+0)/5 | 0.95 = $1-\max\left(\frac{0}{1};\frac{\mathbf{0.1}}{\mathbf{2}};\frac{0.1}{3};\frac{0.1}{4};\frac{0.1}{5}\right)$ |
| $(a_2,b_2)$ | 0.7 = 1-(0.4+0.4+0.4+0.2+0.1)/5 | 0.9 = $1-\max\left(\frac{\mathbf{0.1}}{\mathbf{1}};\frac{0.1}{2};\frac{0.3}{3};\frac{0.1}{4};\frac{0.3}{5}\right)$ |
| $(a_3,b_3)$ | 0.9 = 1-(0.1+0+0+0+0.4)/5 | 0.6 = $1-\max\left(\frac{\mathbf{0.4}}{\mathbf{1}};\frac{0.4}{2};\frac{0.4}{3};\frac{0.4}{4};\frac{0.3}{5}\right)$ |

Figure 3: Computation of stability values for three example correspondences

*Example:* Figure 3 (left) shows an example for the evolution of similarity values for three correspondences. All correspondences have a similarity value of 0.9 in the current version ($n$=6) but show a different behavior in the previous $k$=5 versions. Figure 3 (right) also illustrates the computation of both stability measures, i.e., $stabAvg_{6,5}$ and $stabWM_{6,5}$. The first correspondence is very stable over all considered versions and, thus, achieves high stability values. By contrast, the second correspondence shows high fluctuations which diminish in the last versions. Thereby the average stability is rather low but the correspondence obtains a high weighted maximum stability. The similarity values of the third correspondence are in a very similar range for the first five versions but we observe a steep increase from version 5 to 6. This behavior is reflected by a high average stability together with a low weighted maximum stability because the change in the last evolution step primarily accounts for the computation of the weighted maximum stability.

The given example illustrates that the proposed stability measures quantify different aspects of the stability over a period of versions. We will further demonstrate the usefulness of our stability measures in the following evaluation.

## 3 Evaluation

In this section we show the results of a first evaluation of our approach in order to demonstrate its benefit and practicability for assessing (available) ontology mappings. For this initial work we utilize the life sciences as an example domain. In this domain a huge amount of (meta-)data is generated and the very active research community often revises data due to new (experimental) findings. Among others this process regularly triggers new versions of ontologies and instance data sources that use ontology concepts for semantic annotation of their molecular-biological objects, e.g., genes or proteins.

For this evaluation we consider the two sub-ontologies Biological Processes (BP) and Molecular Functions (MF) from the popular Gene Ontology (GO) [Go04]. These two ontologies consist of 15,131 and 8,827 concepts, respectively (as of Apr. 2008). Furthermore, the evaluation takes into account associated instance data of Ensembl [HAB+06], a data source containing proteins annotated with ontology concepts of the GO. Particularly, Ensembl of July 2008 comprises 46,704 proteins which exhibit 80,705 (100,195) annotations to BP (MF). Referring to our versioning scheme we consider 26 different versions (from Feb. 2004 to July 2008) due to changes in the ontologies and modifications in the associated instance data.

### 3.1 Quantitative statistics

We utilized an instance-based match approach that we have already applied in the life sciences [KTR07] and in e-commerce [TKR07]. The similarity between two concepts is thereby derived from the overlap of the instances that are associated to each of the two concepts. More precisely, we used the similarity measure $sim_{min\text{-}3}$ whereupon each correspondence must have an overlap of at least three instances. Overall, our instance-based match approach computed 3,280 match correspondences between MF and BP for the latest version (26). All correspondences exhibit a minimum similarity value ($sim_{26}(a,b,min\text{-}3)$) of 0.8.

In a first analysis we inspect the presence of match correspondences in different versions, i.e., we determine how many correspondences of version 26 are also present in previous versions (1-25). Figure 4 illustrates the cumulative frequency of correspondences w.r.t. the minimal version number representing the first occurrence of a correspondence. We observe an almost linear slope from version 1 to 22 (see Figure 4: linear fit) followed by a considerably increasing number of correspondences that are only present in the last five versions. Approx. 76% of all correspondences (2,497) have at least five versions, i.e., they exist since version 22 or earlier. Since the significance of stability information is limited for correspondences appearing in only a few versions, we consider these 2,497 correspondences in the following investigation.

We utilize our stability measures in the following way. On the one hand, we set $k$=25 (i.e., the complete history) for average stability ($stabAvg_{26,25}$) to reflect long-term stability of a correspondence. Note that this does not imply that all correspondences must appear in all previous 25 versions. The stability measure definitions make sure that

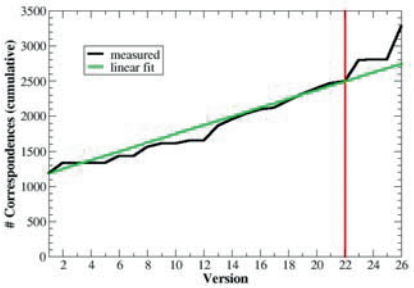



Figure 4: Cumulative frequency of correspondences (w.r.t. their occurrence) in a version

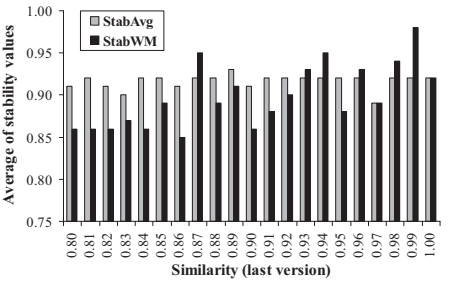



Figure 5: Average stability grouped by correspondences with a certain similarity in the last version (rounded to 2 decimals)

$stabAvg_{26,25}$ is well-defined for all 2,497 correspondences due to the use of *kmax*. On the other hand, $k$=4 is used for weighted maximum stability ($stabWM_{26,4}$) to assess a correspondence short-term stability, e.g., trends in the recent history.

A further evaluation scenario deals with the statistical independence of the similarity measure compared to our stability measures. We aim at establishing new measures which provide additional benefit to annotate ontology match results. For this purpose, we distinguish groups of similarities (each of size 0.01) between 0.8 and 1 for the latest version and calculate the average $stabAvg_{26,25}$ and $stabWM_{26,4}$ stability in each group (see Figure 5). The values for $stabAvg_{26,25}$ ($stabWM_{26,4}$) range from 0.89 to 0.93 (0.85 to 0.98) having a mean value of 0.92 (0.9) without definite order or trend. Thus, we observed that the stability measures are statistically independent from the similarity computed by the match technique. This indicates that the proposed stability measures may be very helpful for classification of match correspondences.

### 3.2 Classification of match correspondences

The stability measures may be used to assess computed match correspondences and to classify them more reliably as match or non-match compared to the sole use of the similarity measure. Note that the following evaluation exemplifies a possible procedure for supporting a manual match decision based on separating correspondences into several groups of quality. We leave the use of automatic matching rules for future work.

We use an upper ($t_{high}$) and a lower ($t_{low}$) threshold for each considered measure ($stabAvg_{26,25}$, $stabWM_{26,4}$, $sim_{26}$). Correspondences with a value higher than $t_{high}$ are considered as the best correspondences w.r.t. the stability criterion, whereas others between $t_{high}$ and $t_{low}$ or lower than $t_{low}$ are considered as intermediate or insufficient, respectively.

In order to set the thresholds for the stability measures we firstly analyze both, $stabAvg_{26,25}$ and $stabWM_{26,4}$. Figure 6 reveals the cumulative frequency of correspondences w.r.t. the two stability measures in the range of 0.6 to 1. A comparatively small number of correspondences possesses stability values of about 0.85 or lower in both measures. Particularly, about 6% (20%) of all correspondences show values lower than

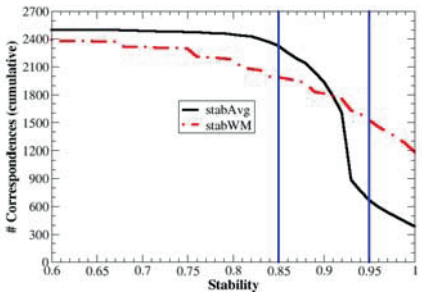



Figure 6: Cumulative frequency of $stabAvg_{26,25}$ and $stabWM_{26,4}$.

| $sim_{26}$ > 0.9 / $sim_{26}$ ≤ 0.9 | stabWM > 0.95 | 0.95 ≥ stabWM ≥ 0.85 | 0.85 > stabWM | Σ |
|---|---|---|---|---|
| stabAvg > 0.95 | 424 / 44 | 37 / 55 | 11 / 25 | 596 |
| 0.95 ≥ stabAvg ≥ 0.85 | 863 / 96 | 203 / 212 | 235 / 125 | 1734 |
| 0.85 > stabAvg | 17 / 5 | 13 / 16 | 85 / 31 | 167 |
| Σ | 1449 | 536 | 512 | 2497 |

Table 1: Number of correspondences classified by $stabAvg_{26,25}$, $stabWM_{26,4}$ and $sim_{26}$.

0.85 in $stabAvg_{26,25}$ ($stabWM_{26,4}$). This is supported by the fact that all cumulative frequencies are nearly stable in the range of 0.6 to 0.85. However, the behavior for $stabAvg_{26,25}$ and $stabWM_{26,4}$ differs for stability values higher than 0.85. While the curve for $stabAvg_{26,25}$ highly decreases between 0.85 and 0.95 (from 2,330 to 669), the cumulative frequency of $stabWM_{26,4}$ slightly decreases (from 1,985 to 1,528). Perfect stability for $stabAvg_{26,25}$ ($stabWM_{26,4}$) is achieved by 377 (1,178) correspondences. As a result, we set a threshold of 0.85 for both $t_{stabAvg,low}$ and $t_{stabWM,low}$ and thresholds for $t_{stabAvg,high}$ and $t_{stabWM,high}$ are set to 0.95 (indicated as lines in Figure 6). Furthermore, based on experiences from former match tasks utilizing $sim_{min-3}$ we set the higher threshold for $sim_{26}$ ($t_{sim,high}$) to 0.9. Note that we implicitly adapted $t_{sim,low}$ to 0.8 before (see Section 3.1).

These threshold settings are specific to our match scenario and can differ for other match tasks since other match data, match techniques and application purposes need to be considered. For instance, the frequency of ontology modifications depends on the domain and therefore may influence stability thresholds. Moreover, the number of previous versions ($k$) that are respected for stability measuring may depend on the extent of ontology changes, e.g., only versions after a major ontology release might be considered.

To exemplify our approach we discuss one result table using the described threshold configuration. Table 1 represents a classification of the correspondences w.r.t. the two stability measures and the last similarity value. In general, the reliability of the sets of correspondences decreases from top left to down right. For a better understanding, we distinguish the results into three groups: I (white), II (light-gray) and III (dark-gray). The correspondences in group I represent 54.8% (1,368) of all correspondences. They possess the best quality as they exhibit high values for at least two criteria and no insufficient criterion. To our understanding they are classified as *accepted*. Group II covers 15.3% (382) and includes correspondences that mainly achieve intermediate and high values in each measure. As these correspondences still obtain quite high stability and similarity values we ranked them as *candidates*. Finally, group III contains a total of 29.9% (747). These correspondences predominantly possess insufficient values for at least one of the stability criteria and, thus, are considered as *questionable*.

In summary, these first evaluation results show that the proposed stability measures can be used to more precisely classify correspondences of an ontology mapping as match or non-match. We thereby utilized upper and lower thresholds (see Table 1) in this study.

## 4 Related Work

Overviews of approaches for ontology matching in general are given in [RB01, ES07]. Match techniques can be roughly categorized into the following groups: metadata-based, instance-based, or hybrid. More precisely, match techniques consider available information of ontologies (e.g., concepts names or descriptions, structure) or utilize associated data (e.g., instances that are annotated with ontology concepts) to compute match correspondences between different ontologies. However, none of these techniques incorporates historical information about match correspondences. Hence, our work complements previous match techniques to enhance their match results by considering evolutionary information of correspondences to produce more stable mappings.

The evolution of ontologies was primarily studied in the context of the Semantic Web. Versioning of ontologies was investigated by Klein [KF01,Kle04]. Furthermore, [NK04] defined change operations to describe the evolution between ontology versions. Supplementary, [SMM+02, SM02] formulized the process of ontology evolution by proposing strategies to unambiguously handle critical ontology changes. Whereas these approaches focus on consistent evolution and versioning of ontologies, we look at the evolution of correspondence similarities to produce more stable match results, i.e., we neither do change ontologies themselves nor migrate dependent instances if ontologies evolve.

The combination of both matching and evolution is investigated within the model management framework of Bernstein et al [BM07]. The schema evolution problem described in [Ber03] utilizes operators such as match, compose and diff to adapt views that would break if a schema evolves to a new version. The approaches proposed in [VMP03, YP05] can be seen as a part of this framework. Whereas [VMP03] presented a framework for incremental adaptation of schema mappings as participating schemas evolve, [YP05] studied the mapping adaptation by employing mapping composition and pruning techniques. These approaches differ from ours in that they reuse match results to efficiently compute new mappings (e.g., for views) if schemas change.

The evolution of association rules has been studied in data mining. Particularly, [AG95] proposed to monitor rules in different time periods by considering changes in their support and confidence. An advanced approach presented in [LHM01] aims to find fundamental association rule changes. Similar to mapping correspondences, association rules also relate (sets of) items to describe semantic relationships. However, data mining approaches attempt to detect rules that significantly differ in their evolution from others whereas we focus on finding the most stable correspondences in an ontology mapping.

## 5 Conclusions and future work

We presented an evolution-based approach for assessing ontology mappings. The approach utilizes two stability measures taking the history of correspondences w.r.t. their computed similarity values into account. While the average stability considers changes between all consecutive ontology versions in the analyzed evolution history, the weighted maximum stability inspects stability w.r.t. the computed similarity value of the

current mapping. In addition to the similarity values, the proposed stability values can be used to rank and evaluate the correspondences of an ontology mapping. Moreover, the stability values are useful to classify correspondences into several groups such as *accepted*, *candidate* and *questionable*. The approach is complementary to existing match approaches that do not consider historical information of correspondences. Furthermore, it is generic in that it can be applied independently from the utilized match technique. The defined stability measures offer the possibility for an individual (application-specific) adjustment. The results of a first evaluation using ontologies of the life science domain show the usability of our approach to classify mapping correspondences.

In future work, we will comparatively analyze and hence prove the applicability of our approach in different domains, e.g., product catalogs in e-commerce or matching web directories. Moreover, we will establish alternative stability measures by taking further knowledge into account, e.g., the structure of ontologies that participate in an ontology mapping. Finally, the management of versioned ontology mappings and the efficient computation of stability measures will be investigated in more detail.

## References


[AP95] Agrawal; Psaila: Active Data Mining. KDD, 1995

[ADMR05] Aumüller; Do; Massmann; Rahm: Schema and ontology matching with COMA++. SIGMOD, 2005

[Ber03] Bernstein: Applying Model Management to Classical Meta Data Problems. CIDR, 2003

[BM07] Bernstein; Melnik: Model Management 2.0-Manipulating Richer Mappings. SIGMOD, 2007

[Do06] Do: Schema Matching and Mapping-based Data Integration. PhD thesis, 2006

[ES07] Euzenat; Shvaiko: Ontology matching. Springer Verlag, 2007

[Go04] The Gene Ontology Consortium: The Gene Ontology (GO) database and informatics resource. Nucleic Acids Research 32, 2004

[HAB+06] Hubbard; Aken; Beal et al.: Ensembl 2007. Nucleic Acids Research 35, 2006

[HKR08] Hartung; Kirsten; Rahm: Analyzing the Evolution of Life Science Ontologies and Mappings. DILS, 2008

[Kle04] Klein: Change Management for Distributed Ontologies. PhD thesis, 2004

[KF01] Klein; Fensel: Ontology versioning on the Semantic Web. Int. Semantic Web Working Symposium (SWWS), 2001

[KS03] Kalfoglou; Schorlemmer: Ontology mapping: The state of the art. The Knowledge Engineering Review Journal 18(1), 2003

[KTR07] Kirsten; Thor; Rahm: Instance-based matching of large life science ontologies. DILS, 2007

[LHM01] Liu; Hsu; Ma: Discovering the Set of Fundamental Rule Changes. KDD, 2001

[NK04] Noy; Klein: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems 6(4), 2004

[RB01] Rahm; Bernstein: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 2001

[SM02] Stojanovic; Motik: Ontology evolution within ontology editors. OntoWeb-SIG3 Workshop, 2002

[SMM+02] Stojanovic; Maedche; Motik et al.: User-driven ontology evolution management. Conf. On Knowledge Engineering and Knowledge Management, 2002

[TKR07] Thor; Kirsten; Rahm: Instance-based matching of hierarchical ontologies. BTW, 2007

[VMP03] Velegrakis; Miller; Popa: Mapping Adaptation under Evolving Schemas. VLDB, 2003

[YP05] Yu; Popa: Semantic Adaptation of Schema Mappings when Schemas Evolve. VLDB, 2005

# Model-based Classification of Data with Time Series-valued Attributes

Christian Böhm
University of Munich
boehm@dbs.ifi.lmu.de

Leonhard Läer
Technische Universität München
leonhard.laeer@lrz.tum.de

Claudia Plant
Technische Universität München
plant@lrz.tum.de

Andrew Zherdin
Technische Universität München
andrew.zherdin@lrz.tum.de

**Abstract:** Similarity search and data mining on time series databases has recently attracted much attention. In this paper, we represent a data object by several time series-valued attributes. Although this kind of object representation is very natural and straightforward in many applications, there has not been much research on data mining methods for objects of this special type. In this paper, we propose a novel model-based classifier exploiting the benefits of representing objects by several time series. Classification decisions are based on class-specific interaction patterns among the time series of an object. Experimental results on benchmark data and real-world medical data demonstrate the performance of our classifier.

## 1 Introduction

Time series data are collected in many applications including finance, science, natural language processing, medicine and multimedia. The content of large time series databases can not be analyzed manually. To support the knowledge discovery process from time series databases, effective and efficient data mining methods are required. Often, the primary goal of the knowledge discovery process is classification, which is the task to automatically assign class labels to data objects. To learn rules, strategies or patterns for automatic classification, the classifier needs to be trained on a set of data objects for which the class labels are known. Typically, this so-called training data set has been labelled by human experts. Based on the patterns learned form the training data set, the classifier automatically assigns labels to new, unseen objects.

There is a huge volume of papers on classification of time series, e.g. [Kad99, KP97, Man95]. However, most existing approaches consider a single time series as the data object to be classified. In this paper, we focus on the representation of a data object by several time series-valued attributes and propose a novel algorithm for classification of such objects. Representing an object by a set of time series appears very natural and straightforward in many applications.

Consider for example a meteorological database storing measured data from different meteorological stations. Each station can be regarded as an object which is represented

by several time series-valued attributes, including temperature, rainfall and air pressure. Based on the measured data over a certain period, stations can be automatically classified e.g. into stations with normal weather conditions and stations with exceptional conditions which are presented to a human expert.

In many medical applications, it is natural to represent a data object by a system of time series. Consider for example EEG data: Electroencephalography measures the electric activity in the brain by an array of usually 64 electrodes which are distributed over the scalp [PM07]. It is very natural to consider the EEG of one person as a data object, where the 64 time series-valued attributes represent the 64 electrodes. Another example is functional magnetic resonance imaging (fMRI) [GVS+07]. This imaging modality yields time series of 3-dimensional images of the brain. Also in this case, it is reasonable to regard each subject as a data object which is represented by time series-valued attributes. Each 3-d pixel called voxel of the fMRI image is an attribute of time series type.

Increasing amounts of captured motion stream data are collected in multimedia applications [CLP07]. Gesture sensing devices, such as a CyberGlove usually contain multiple sensors to capture human movements. Motion classification is the task to automatically assign movements, usually performed by different people, into predefined motion classes and has important applications in gait analysis and virtual reality. Again, it is reasonable to regard each movement as a data object which is represented by the time series obtained from the different sensors of the gesture sensing device.

Although representing an object by a system of time series is very natural in many applications, there has not been much research on data mining methods for objects of this special type. We propose a classifier which exploits the benefits of this kind of object representation by focussing on interaction patterns between the time series within a data object. We derive a description of each class in terms of a system of linear models characterizing class-specific interaction patterns. Our experiments demonstrate that the information on interactions substantially improves the classification accuracy. In addition, the models provide interesting insights for understanding class-specific differences.

The remainder of this paper is organized as follows. After a brief survey of related work in the next section, we introduce our classifier in Section 3. Section 4 presents an experimental evaluation and Section 5 concludes the paper.

## 2 Related Work

To describe the interaction patterns within the time series of an object, we follow a model-based approach first introduced in [KKP+08] for the purpose of efficient compression of time series. To derive an approximation to speed up similarity search, in this approach each time series is represented by a combination of a set of specific reference time series. We use equivalent linear models to represent interactions of time series and a similar algorithm for model finding but with a very different objective.

In comparison to general time series classification, only relatively few papers focus on classification of multivariate time series. In [YSYT03] a decision tree for data with time-

series attributes is proposed. At the internal nodes of this tree, time series are stored and splits are performed based on dissimilarity between pairs of time series. The method selects upon split a time series which exists in data by exhaustive search based on class and shape information.

In [KS05] the authors present tClass, an algorithm for classifying multivariate time series data based on so-called meta-features. These meta-features model some kind of recurring substructure in the instances, such as strokes in handwriting recognition, or local maxima in time series data. The types of substructures are defined by the user, but the substructures are extracted automatically and are used to construct attributes. The parameter space defined by the meta-features is segmented into regions beneficial for classification using a heuristic technique. After feature construction, classification is performed with conventional propositional learning algorithms, such as decision trees.

For EEG data, some specialized classifiers have been proposed, e.g. in [PM07] to use EEG as a biometric for person identification. To identify persons with high accuracy based on EEG data with visual stimuli, after selection of the best discriminative channels classification is performed by a neural network.

Also in [PR01] a classification method especially designed for EEG with visual stimuli is proposed. This method extracts the visual evoked potential (VEP) which is embedded in the ongoing EEG and blurred by noise. The VEP contained in the gamma band is extracted by digital filters. Similar to [PM07], classification is performed by a neural network.

In [ZG02] a distance-coupled Hidden Markov model (HMM) is introduced for classification of EEG data. The distance-coupled HMM applies several HMMs to model multivariate time series. The link between the different HMMs is established by the fact that the state of one model at some particular time stamp $t$ depends on the states of the other models at time stamp $t-1$. In their experiments the authors demonstrate the usability of coupled HMMs for EEG classification. In contrast to [ZG02], our models capture dependencies not between time points but between the dimensions. Our models are applicable to classes and not to single objects. So our models have a very different structure compared to HMMs.

## 3 Model-based Classifier

Our method for model-based classification involves 3 steps:

1. model finding,
2. model refinement,
3. classification.

In the training phase of the classifier, in the first step the interaction patterns of the time series within each class are characterized by a system of linear models. In the second step,

these models need to be refined to be suitable for classification. The final step is the classification of test objects.

**Notations and Definitions.** We represent each class as a system of linear models. Following [KKP+08], a single model can be defined as follows.

**Definition 1 (Model)** *A mathematical model $M = (\vec{X}, \vec{\alpha}, f)$ for a dependant variable $Y$ (output) consists of the following parts:*

- *a set of* exploratory variables $X_1, \ldots, X_k$ *(inputs),*
- *a* mathematical function
$$f(\vec{X}, \vec{\alpha})$$
*that is used to describe the dependency between $Y$ and $X_1, \ldots, X_k$. $\vec{\alpha}$ denotes the model parameters also called* coefficients of the model.

*The general form of the model is given by $Y = f(\vec{X}, \vec{\alpha}) + \varepsilon$, where $\varepsilon$ denotes the random error.*

In this paper, we focus on linear models, i.e. $f$ is a linear combination of the inputs.

Further, we consider a data set $DS$ with $n$ objects, i.e. $DS = \{O_1, ..., O_n\}$. Each object $O_i$ is represented by $d$ time-series valued attributes $X_j$, each consisting of a real-valued time series with $m$ time points, i.e. $O_i = \{X_1, ...X_d\}$ and $X_j = \{t_1, ..., t_m\}$. For the training data, a categorical class label is known. To facilitate our argument, we focus on a 2-class problem with class labels *class1* and *class2*. However, our method can be straightforwardly applied in a multi-class setting.

**Model Finding.** To capture the interaction patterns of time series within a class, e.g. *class1* we first have to generate a summarization of the time series attributes of the objects of the class. For each of the $d$ dimensions, we concatenate the time series of all objects of the class. As a result we have a novel object $O_s$ with $d$ time series-valued attributes with $|class1| \cdot m$ time points. Now, we can derive a system of linear models as specified in Definition 1 for $O_s$. All $d$ attributes of $O_s$ are applied as output $Y$ once. The idea is to try to explain each individual attribute by a linear combination of (potentially all) other attributes. To avoid over fitting, models with few inputs are preferable. In addition, an exhaustive search for all combinations of inputs is not feasible for large $d$. To cope with these problems, stepwise algorithms are often a good choice [Lar06, Geo00].

We propose an iterative stepwise algorithm for model finding which is displayed in pseudocode in Figure 1. We start with the output $Y$ and the set of inputs $X$. Our purpose is to find a subset of $X$ containing only that inputs which are most relevant to explain $Y$. To achieve a balance between model complexity and goodness of fit, we apply the Bayesian Information Criterion (BIC) [Geo00]. Theoretically well founded in Bayesian statistics and information theory, this criterion penalizes overly complex models and allows us to select the most relevant inputs in a parameter-free way. Our algorithm starts with an empty set $relevantInputs$. It then iteratively either adds a new input to $relevantInputs$ or

```
algorithm findModel(Output Y, Inputs X): Model M
  relevantInputs := ∅;
  oldMIN := +∞;
  do {
//(1) find a relevant input
      [minPlus, x+] := min_{x∈(X\relevantInputs)} (BIC(relevantInputs ∪ {x}, Y));
      [minMinus, x−] := min_{x∈relevantInputs} (BIC(relevantInputs \ {x}, Y));
      if(min(minPlus, minMinus) > oldMIN) then
// (2) no more relevant inputs
        break;
      end if
      if(minPlus < minMinus) then
        relevantInputs.add(x+);
        oldMIN := minPlus;
      else
        relevantInputs.remove(x−);
        oldMIN := minMinus;
      end if
  } while()
// (3) build the Model with inputs
  M := new Model(relevantInputs, Y);
return M;
```

Figure 1: Algorithm for Model Finding.

removes an already existing element depending on which of these two actions leads to a greater improvement of BIC (1). Based on these $relevantInputs$ the model is constructed by least-squares fitting. The algorithm terminates if no further improvement of BIC can be achieved (2). The algorithm finally returns the model based on $relevantInputs$ (3).

**Model Refinement.** After application of the model finding algorithm using each attribute of $O_s$ as an output, we obtain a set of models $SM$ consisting of $d$ models per class. However, these models can not be directly applied for classification. $SM$ may contain models with large error and models representing general trends in the data which are not class-specific. Therefore the set of models needs to be refined to build a classifier with good discriminatory power. As depicted in pseudocode in Figure 2, the algorithm for model refinement involves two steps: First, we exclude models with large error since they do not capture any distinct interaction information. This is implemented by removing all models having an error larger than three times the median of the error on training data (1). From the remaining models we select those which discriminate between classes. For all classes and models, we compute the cumulative error on the training data. Models having a smaller cumulative error for the correct class than for the wrong class are included in the classifier (2).

**Classification.** The classification of the test objects is now simple. To classify an object, we sum up the mean square error for all relevant models for all classes. We assign the object to the class with the smallest mean square error.

```
algorithm refineModels(set of models SM, training objects O): refined set of models RSM
// (1) remove corrupt models
   median := median error of SM;
   for each i ∈ {1 ... SM.size()} do
     if(SM[i].error > 3 · median) then
       SM.remove(i);
     end if
   end for

// (2) find class separating models

   for each m ∈ {1 ... SM_class1.size} do
     for each o ∈ {1 ... obj_class1.size} do
       difference[i]+ = mean_square_error(m, o);
     end for
     for each o ∈ {1 ... obj_class2.size} do
       difference[i]− = mean_square_error(m, o);
     end for
   end for
   for each m ∈ {1 ... SM_class2.size} do
     for each o ∈ {1 ... obj_class1.size} do
       difference[i]− = mean_square_error(m, o);
     end for
     for each o ∈ {1 ... obj_class2.size} do
       difference[i]+ = mean_square_error(m, o);
     end for
   end for

// result

   for each i ∈ {1 ... difference.size} do
     if(difference[i] < 0) then
       RSM.add(i);
     end if
   end for
return RSM;
```

Figure 2: Algorithm for Model Refinement.

**Runtime Complexity.** The worst-case runtime complexity of the stepwise model finding algorithm is quadratic in $d$. Usually, the number of time points $m$ is much larger than the number of dimensions $d$. Due to matrix inversion required for least square model fitting, we then have cubic complexity in $m$. The model refinement step, as well as the classification of an object are quadratic in $d$ and linear in $m$.

## 4 Evaluation

**Data Sets.** Table 1 provides a summary on the data sets used for evaluation. Data sets 1 to 8 are different EEG data sets available at the UCI machine learning repository[1]. These data sets are derived from a study on alternations of brain activity in alcoholic subjects in comparison to a control group. Each subject was exposed to either a single visual stimulus (S1) or to two visual stimuli (S1 and S2). When two stimuli were shown, they were presented in either a matched condition where S1 was identical to S2 or in a non-matched condition where S1 differed from S2. Data sets 1 to 4 correspond to the so-called

[1] http://archive.ics.uci.edu/ml/databases/eeg/eeg.data.html

| Name | Description | Data Objects | #Objects | #Classes | #Dimensions | Length |
|---|---|---|---|---|---|---|
| DS1 | EEG single stimulus | subjects | 40 | 2 | 64 | approx. 2560 |
| DS2 | EEG matched stimulus | subjects | 40 | 2 | 64 | approx. 2560 |
| DS3 | EEG non-matched stimulus | subjects | 40 | 2 | 64 | approx. 2560 |
| DS4 | EEG combined | subjects | 40 | 2 | 64 | approx. 7680 |
| DS5 | EEG single stimulus | runs | 20 | 2 | 64 | 256 |
| DS6 | EEG matched stimulus | runs | 20 | 2 | 64 | 256 |
| DS7 | EEG non-matched stimulus | runs | 20 | 2 | 64 | 256 |
| DS8 | EEG combined | runs | 20 | 2 | 64 | 768 |
| DS9 | fMRI | subjects | 26 | 2 | 90 | 325 |
| DS10 | motion stream | signs | 2565 | 95 | 22 | approx. 57 |

Table 1: Data Sets.

Large Data Set at UCI where training and test data are available. The task is to classify the subjects into the classes alcoholic and control based on 10 time series-valued attributes. To generate a representation of the objects of a class, we concatenated the time series of all objects and all runs resulting in a time series of 2,560 time points. Data sets 5 to 8 are are derived from the so-called Small Data Set which contains data of 2 subjects, one alcoholic and one of class control. For each of the 3 experimental paradigms, 10 runs of EEG have been recorded. The task here is to classify the EEG runs to the classes alcoholic and control.

DS9 [GVS+07] consists of 26 functional MRI images. This data set has been obtained from a study on somatoform pain disorder and consists of images of 13 diseased subjects and 13 healthy controls. For classification we selected 90 regions of interest as proposed in [TMLP+02].

DS10, also from UCI, [2] contains motion stream data of Australian sign language. For each of the 95 Australian language signs, 27 examples were captured from a native signer using high-quality position trackers and instrumented gloves.

**Comparison Methods and Validation.** For comparison, we implemented two basic classifiers: Nearest neighbor (1NN) and classification to the nearest class centroid (centroid). Both basic classifiers do not consider interaction patterns among time series. To classify a test object, for all training objects and all attributes the nearest neighbor classifier sums up the Euclidean distance and assigns the object to the class of that training object with the smallest overall distance. The centroid classifier computes for each class the centroid by averaging all time series-valued attributes of all training instances. A test object is assigned to the label of the closest centroid.

We obtained the code of TClass from the authors [KS05]. The parametrization of this method is very difficult, since the type of meta-features as well as the base learner need to be selected by the user. As recommended in [KS05], we use a universal set of meta-features for temporal domains. In addition, we use following aggregate global attributes: mean, min, max and mode. Mean, max and min were recommended by authors for DS10.

[2] http://kdd.ics.uci.edu/databases/auslan2/auslan.data.html

| Data Set | Centroid | 1-NN | Models | TClass | Others |
|---|---|---|---|---|---|
| DS1 | 50% | 50% | **100%** | 70% | n.a. |
| DS2 | 55% | 55% | **90%** | 74% | n.a. |
| DS3 | 80% | 55% | **100%** | 72% | n.a. |
| DS4 | 60% | 50% | **100%** | 72% | 96% [PR01]/ 98% [PM07] |
| DS5 | 65% | 90% | **98%** | 89% | n.a. |
| DS6 | 40% | 80% | **100%** | 86% | n.a. |
| DS7 | 90% | 90% | **100%** | 89% | n.a. |
| DS8 | 85% | 90% | **100%** | 91% | 90% [ZG02] |
| DS9 | 85% | 77% | **96%** | 58% | n.a. |
| DS10 | 11% | 46% | 75% | 96% | **98%** [KS05] |

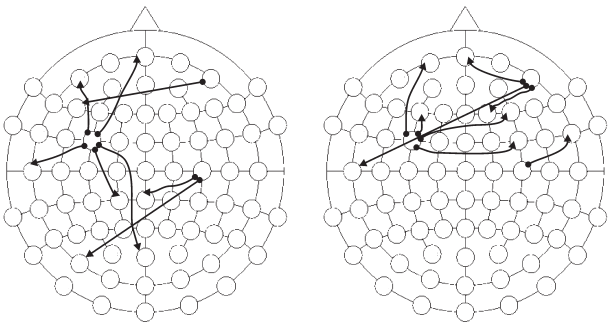
Table 2: Results.

Mode improved results on DS1-DS8. In all experiments, we use AdaBoost with J48 decision tree as learner. According to the experiments in [KS05], this is the third-best base learner for TClass. Two slightly better learners are Naive Segmentation and voting. Although Naive Segmentation is implemented, we could not repeat the very good results. We do not use voting since it is not implemented and it is unclear, which voting algorithm was used in [KS05]. Because TClass is not deterministic, unless otherwise specified, we report the mean of accuracy of 10 runs.

In addition we compare our method to [PM07, ZG02, PR01], 3 classifiers especially designed for EEG data (cf. Section 2). For DS1-DS4 train and test data are given. Classification results are therefore directly comparable. For DS5-DS8 we implement 5-fold cross-validation as in [ZG02]. We do not provide a comparison to [YSYT03] since the authors arbitrarily selected runs from the full EEG data set. We validate DS9 with leave-one-out, since there are only few instances. For DS10 we implement 10-fold cross-validation as in [KS05].

**Results.** Our model-based classifier demonstrates an excellent accuracy of 100% on 6 out of 10 data sets and strongly outperforms the two basic classifiers. Table 2 summarizes the results. On EEG data, our classifier even outperforms the classifiers especially designed for this type of data [PM07, ZG02, PR01]. These results demonstrate that the information on interaction patterns is highly relevant for the classification of EEG and fMRI data. On data set DS9 (fMRI) our model-based classifier shows very good results. Only one object is incorrectly classified. The most important models for classification decision, are judged as reasonable by domain experts.

Only on DS10, TClass performs significantly better than all comparison methods. But notice that DS10 has been originally published by the authors of [KS05] and TClass has been designed for motion stream data. With 75% in accuracy, our model-based classifier performs much better than the basic methods centroid and 1-NN. This indicates that attribute interactions are also useful for the classification of motion stream data. However, evidently, the interactions in this data set can not perfectly be captured by linear models.



(a) Alcoholic. (b) Control.

Figure 3: Illustration of Best Class-separating Models on UCI EEG Data.

We could not reproduce the accuracy of TClass reported in the publication (98%) which was obtained using voting, since it is not clear which voting strategy has been applied. In addition, TClass with voting is very inefficient. The run[3] without voting for DS10 lasts approximately 24 hours. For 11 voters, as recommended in [KS05] a naive voting algorithm would take 11 days. The model-based classifier run needs less than 40 minutes under equal conditions. Moreover, our algorithm is parameter-free and deterministic, whereas TClass exhibits a very high variation of results. E.g. for DS1 accuracies between 20% and 85% are obtained.

As an additional advantage, the interaction models generated by our classifier provide interesting information for interpretation. Figure 3 displays the 3 most relevant models for separating the classes on DS1. For both classes, the layout of the 64 electrodes on the scalp is displayed together with the best separating interaction patterns. The models of the alcoholic subjects involve different brain regions than those of the healthy control group. The subtle differences of interactions need to be systematically evaluated in future work.

## 5 Conclusion

In this paper, we proposed a novel model-based classifier for data with time series-valued attributes. Classification decisions are supported by class-specific interaction patterns within the time series of a data object. The experimental evaluation demonstrates that interaction patterns characterized by linear models are very useful for classification, especially of EEG and fMRI data.

In ongoing work we focus on interpretation of the identified models. We will interpret the models of fMRI for somatoform pain disorder patients to find the most important regions. In addition, we are currently evaluating the classifier on EEG data from our cooperation partners for the identification of subjects in different stages of anesthesia. It would also be

[3] 10-fold cross-validation; 33 runs; Intel Core 2 Quadro(2,66 GHz), 8 GB RAM

interesting to design model-based classifiers for fMRI data and combined fMRI-EEG data sets which is very challenging because of the large number of time series in fMRI data. We also plan to evaluate the classifier on time series data from other domains.

Furthermore, we want to extend our model notion to more general models. Among building blocks to be investigated are differential equations and kernel functions.

## References


[CLP07] L. Khan C. Li and B. Prabhakaran. Feature Selection for Classification of Variable Length Multiattribute Motions. In V. A. Petrushin and L. Khan, editors, *Multimedia Data Mining and Knowledge Discovery*. Springer, 2007.

[Geo00] E. I. George. The variable selection problem. *J. Amer. Statist. Assoc*, 95:1304–1308, 2000.

[GVS+07] H. Gündel, M. Valet, C. Sorg, D. Huber, C. Zimmer, T. Sprenger, and T.R. Tölle. Altered cerebral response to noxious heat stimulation in patients with somatoform pain disorder. *Pain.*, 137:413–421, Nov 2007.

[Kad99] M. W. Kadous. Learning Comprehensible Descriptions of Multivariate Time Series. In *ICML*, pages 454–463, 1999.

[KKP+08] H.-P. Kriegel, P. Kröger, A. Pryakhin, M. Renz, and A. Zherdin. Approximate Clustering of Time Series Using Compact Model-Based Descriptions. In *DASFAA*, pages 364–379, 2008.

[KP97] A. Kehagias and V. Petridis. Predictive Modular Neural Networks for Time Series Classification. *Neural Networks*, 10(1):31–49, 1997.

[KS05] M. W. Kadous and C. Sammut. Classification of Multivariate Time Series and Structured Data Using Constructive Induction. *Mach. Learn.*, 58(2-3):179–216, 2005.

[Lar06] D. T. Larose. *Data Mining Methods and Models*. John Wiley & Sons, 2006.

[Man95] S. Manganaris. Learning to classify sensor data, 1995.

[PM07] R. Palaniappan and D. P. Mandic. EEG Based Biometric Framework for Automatic Identity Verification. *VLSI Signal Processing*, 49(2):243–250, 2007.

[PR01] R. Palaniappan and P. Raveendran. Single trial VEP extraction using digital filter. *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, pages 249–252, 2001.

[TMLP+02] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot. Automated Anatomical Labeling of Activations in SPM Using a Macroscopic Anatomical Parcellation of the MNI MRI Single-Subject Brain. *NeuroImageVolume 15*, pages 273–289, January 2002.

[YSYT03] Y. Yamada, E. Suzuki, H. Yokoi, and K. Takabayashi. Decision-tree Induction from Time-series Data Based on a Standard-example Split Test. In *ICML*, pages 840–847, 2003.

[ZG02] S. Zhong and J. Ghosh. HMMs and coupled HMMs for multi-channel EEG classification, 2002.

# Bridging the Gap Between Variability in Client Application and Database Schema

Norbert Siegmund[1]*, Christian Kästner[1], Marko Rosenmüller[1]*,
Florian Heidenreich[2], Sven Apel[3], and Gunter Saake[1]

[1]School of Computer Science, University of Magdeburg, Germany
[2]Department of Computer Science, Dresden University of Technology, Germany
[3]Department of Informatics and Mathematics, University of Passau, Germany
[1]{nsiegmun,ckaestne,rosenmue,saake}@ovgu.de,
[2]florian.heidenreich@tu-dresden.de, [3]apel@uni-passau.de

**Abstract:** Database schemas are used to describe the logical design of a database. Diverse groups of users have different views on the global schema which leads to different local schemas. Research has focused on view integration to generate a global, consistent schema out of different local schemas. However, this approach is too constrained when the generated global view should be variable and only a certain subset is needed. Variable schemas are needed in software product lines in which products are tailored to the needs of stakeholders. We argue that traditional modeling techniques are not sufficient for expressing variable database schemas. We show that software product line methodologies, when applied to the database schemas, overcome existing limitations and are able to generate tailor-made database schemas.

## 1 Introduction

Database client applications become more and more customizable and variants can be tailored to specific use cases and user requirements. Prominent techniques to achieve variability and create custom variants are software product lines [BCK98, PGvdL05], component systems [Szy02] and plug-in based frameworks [JF88]. However, an increasing demand on variability in the client application requires to tailor also the underlying database schema. In prior research, only few researchers addressed this issue, e.g., by means of view integration [SP94, Par03, BLN86] or view tailoring [BQR07]. In practice, usually a fixed global schema for all variants of the client application leads to a gap between variability in client applications and database schemas. The database does not support the variability of the application.

As a running example, we use a simple software product line of university management systems. University management systems can be generated for different clients. For

*The author is funded by the German Ministry of Education and Research (BMBF). The work is part of the ViERforES project.

example, the client application can be delivered (1) as standard variant to support course offerings and a basic book recommendation system, (2) as extended variant which replaces book recommendations by full library support, or (3) as an enterprise variant with connectors to existing systems for room planning. By combining different components of the system, thousands of variants can be generated. The problem is that these variants require different underlying database schemas, which must be flexible accordingly.

A standard approach is to create a single database schema that contains all elements that might ever be needed by any variant of the client application. However, in large applications with high variability there are several problems. First, if variability is high, the global database schema will be large and complex. Second, the maintainability is reduced, because it is difficult to understand and change a global schema incorporating all variants, e.g., tables not needed and maintained in most variants are still present in the global database schema. Furthermore, the lack of separation of concerns and, thus, the suboptimal traceability of requirements to database schema elements complicate development and evolution. This may even hinder application developers to introduce new variants in the application. Finally, variability in applications can be high, there can be millions of variants; this does not scale for one global database schema.

In order to solve the above problems, we propose to tailor database schemas according to client applications. We present two software engineering techniques that are useful depending on the development and deployment scenario. (1) *Physically decomposed* database sub-schemas can be composed to generate a tailored local schema. (2) A *virtually decomposed* schema (an annotated global schema) can be used to remove unneeded elements to also tailoring the database schema. We offer two tools that allow developers to tailor database schemas using physical or virtual decomposition.

## 2 Software Product Line Engineering

A *software product line (SPL)* is a family of software products sharing a common, managed set of features that satisfy specific needs of a domain and that are developed from a common set of core assets in a prescribed way [BCK98, PGvdL05]. For instance, the university management software described above can be considered as an SPL. Different variants in an SPL are distinguished in terms of features, where *features* are distinguishable, end-user visible characteristics of a system or domain [CE00]. For example, the variants "basic" and "extended" of our university SPL are distinguished in that variant "basic" does not have feature *Library* while variant "extended" has this feature. The overall goal of SPL engineering is to systematically reuse software artifacts for different software products [PGvdL05].

Commonly, a feature model is used to describe the variability of an SPL [CE00]. A feature model is represented as a tree (cfg. Fig. 1). For example, a feature connected by an empty dot is optional, one connected with a filled dot is mandatory, and arcs are used to denote alternatives. Additional cross-tree constraints are possible as well. Using constraints like *requires* or *excludes* between features allows developers to define dependencies. For

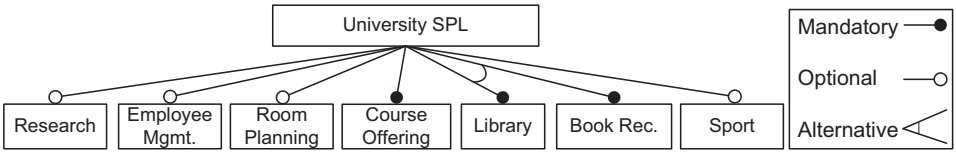



Figure 1: Simple Feature Diagram of a University SPL.

example, if a feature reads a column of a database schema and another writes at this column than the developer may create a constraint between these features in order to enforce correctness of the generated database schema. The concepts of optional and alternative features introduce variability into an SPL. In Figure 1, we show a subset of the feature model for our university SPL with already 32 possible variants. It is easy to see that with further features the number of products can grow exponentially.

Software product line engineering involves two processes: domain engineering and application engineering. While, in traditional software development, only a single application is developed, an SPL covers an entire domain. Thus, *domain engineering* starts with domain analysis, i.e., analyzing commonalities and differences between different products of an SPL. As a result, a feature model describing the variability is created and can be implemented. Creating a single product is the process of assembling the desired product based on a feature selection. The concrete assembly process depends on the implementation mechanism (e.g., code generation [CE00], composition [BSR04, ALS08], preprocessors [KAK08, AKL09]).

## 3 Variable Database Schemas

Before we explain our approach, we will outline the variability problem in more detail and show which workarounds exist in practice to overcome the problem.

### 3.1 Problem Statement

To illustrate the problem of database schema variability, we show in Figure 2 an excerpt of our university SPL's schema in the form of an *entity relationship (ER)* diagram [Che76]. The schema describes the entities that are required by the following features from Figure 1: *Course Offering*, *Room Planning*, *Library*, and *Book Recommendations*. Roughly the connection is as follows: in the "basic variant" with only *Course Offering* and *Book Recommendation* features, only the entities *Catalogue*, *Course*, and *SBook* are required. If feature *Room Planning* is selected, additionally entity *Room* and attribute *Capacity* of *Course* are needed, and, finally, entities *Book*, *Author*, and *Book Copy* are required for the *Library* feature.
Although this schema shows only a small excerpt, it is reasonable to expect that large and complex schemas are difficult to maintain, because they merge elements required for different features. A developer who is interested only in a single feature must read and

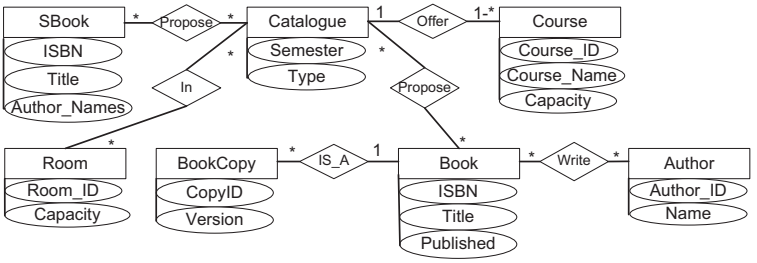



Figure 2: Database schema of the university SPL (excerpt).

understand the entire schema including many elements not relevant for her task. Even attributes such as *Capacity* may be used only in application variants with certain features.

The problem described above gets even worse in case of alternative features. In our university example, we have two alternative variants of book recommendations for a course. In the "basic variant" *Book Recommendation*, the lecturer uses a simple Web formular to enter author, title and ISBN of books she recommends. In the more advanced variant, the information is taken directly from the library (in the case the university bought the "extended variant" with library support, i.e., the *Library* feature is selected). In Figure 2, this is modeled with two alternative entities *SBook* for the "basic variant" and *Book* (and *Author* and *BookCopy*) for the extended library variant, because the same name *Book* could not be used for both, even though only one of them is needed at a time (as specified by the feature model, Fig. 1). This makes the schema difficult to understand and to maintain.

### 3.2 Workarounds

In this section, we present an analysis of two common approaches used in practice. The first example is taken from the Web content management system Typo3 (`http://typo3.org`), which can be extended and configured with a large amount of plug-ins (e.g., for guest books, for forums, for picture galleries). Dependencies between plug-ins are specified in a plain description and can be considered as feature model. In this system, plug-ins may extend the database schema with additional tables or attributes. The second example is taken from a social network Web project called webuni.de (several hundreds of thousands lines of PHP, HTML and SQL code, evolved and extended over 6 years) that aims at building a set of social networks that are each local for a certain town. However, for different instances of the social network, different functionality is provided, e.g., article system, forums, or calendar module. The software is modularized and modules can be configured flexibly for each town, however the underlying database schema is the same for all variants.

In both cases, the client implementation is already modularized in terms of features, but the database schema is not. In Typo3, plug-ins can provide extensions to the database schema on SQL level (new 'create table' scripts). Because of the absence of a conceptual modeling infrastructure each extension defines their own 'create table' statements. This results in

consistency problems, as the global name space is not checked (different definitions for the same table are possible) and foreign keys between tables of different plug-ins are not created. Instead of foreign keys, consistency checks are implemented in the application logic.

In the social network software, all modules use the same global schema. This way, an extension for a single town requires to change all local databases and external plug-ins that require additional tables are not possible. Developers working on a single module always have the burden of understanding the entire database schema, including attributes of entities required only be few modules. This makes learning the framework difficult for new developers. Alternative features are hardly used in this project which may be caused by the involved complexity to support this. Finally, in local installations there may be many unused schema elements, which still need to be maintained by local administrators.

### 3.3 Our Approach

We aim at a proper separation of concerns already at the level of conceptual modeling. We want to be able to trace features not only to the implementation level as already done in software product line engineering, but also to database schema elements. In the generated local schema for a specific application variant, only the required database schema elements should be included. The basic idea is to decompose the schema in terms of features. Decomposition means the mental process of structuring a complex problem into manageable pieces. Following recent research in software engineering [KAK08, HKW08, AKL09], we propose two principle approaches to decomposition, namely virtual and physical decomposition.

**Virtual Decomposition.** In a virtual decomposition, schema elements are not physically separated into different files, but only annotated. That is, schema elements belonging to features (such as entity *SBook* belongs to the *Book Recommendation* feature) are left intermixed inside a single schema, but it is highlighted which elements belong to which features. To generate a variant for a given feature selection, the annotations are evaluated and elements that are not necessary are removed. The resulting schema in the generated variant is a subset of the global database schema, i.e., a view on it. Note, that this is in fact the deployed schema and not only an external view on a global schema. A virtual decomposition can easily be achieved if there is already an existing database schema. The existing global schema is annotated with features to allow the tracing of the schema elements to the features, and to enable a generation of tailored variants.

A standard approach for annotations on code level is to use *#ifdef* statements as provided by the C preprocessor. In more recent approaches [KAK08, HKW08], separately stored annotations and background colors on the representation layer are used. The advantage is that the developer can easily distinguish between the modeling artifacts and information regarding features. We have implemented an Eclipse-based ER modeling tool that can be used with FeatureMapper [HcW08] to annotate database schemas with features. Fea-

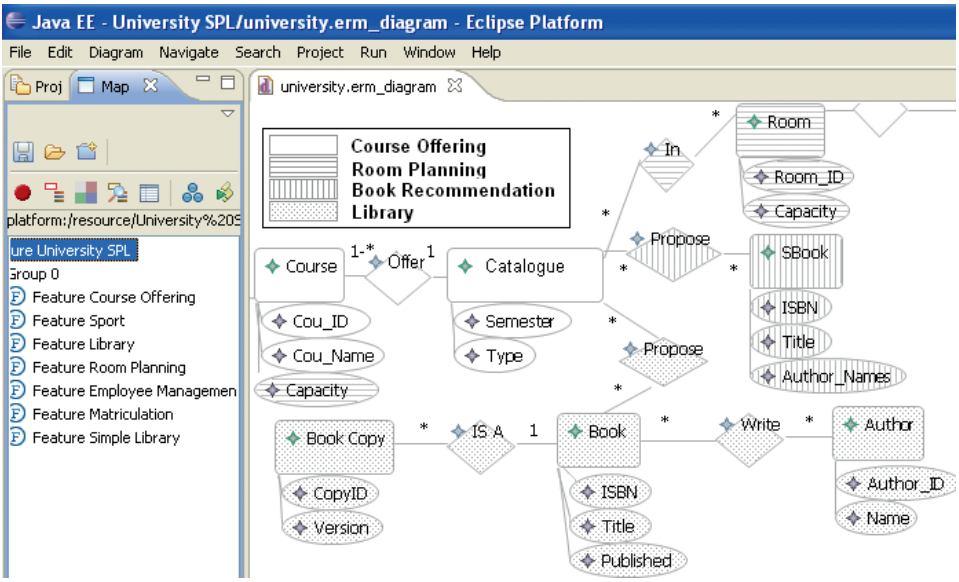



Figure 3: A database schemas annotated with FeatureMapper.

tureMapper supports annotations of all models that are based on the Eclipse GEF/EMF frameworks, e.g., UML models or statechart models. We extended Eclipse with a new Ecore-based ER model editor and integrated it with FeatureMapper. Features can be shown in the user interface by different background colors, additional to the formal underlying mapping model. To achieve a separation of concerns, the editor provides views that show only schema elements of individual features, or previews of a configured product, already during development. This way, although there is no physical decomposition, it is still possible to reason about features or variants in isolation.

In Figure 3, we show a screenshot of the university SPL database schema as annotated in our editor, based on the description in Section 3.1. For the printed version of this paper, we used different background patterns to highlight features. We can even make fine-grained annotations as to annotate the *Capacity* attribute with the *Room Planning* feature.

**Physical Decomposition.** In several scenarios a virtual decomposition is not feasible. For example, when plug-ins can be provided by third parties as in the Typo3 example, an annotated global database schema cannot be formulated. In such case a physical decomposition is useful. Schema elements are stored in separate files, one file per feature. To generate a schema for a specific variant, those files corresponding to the feature selection are composed. This way, a feature can be traced not only to the application's implementation, but also to the according sub-schema in a single file.

Feature: *Course Offering*

```
1 entity Catalogue { Semester, Type };
2 entity Course { Cource_ID, Course_Name);
3 relationship Offer between Catalogue (1) and
      Course (1-*);
```

Feature: *Library*

```
6  entity Book { ISBN, Title, Published };
7  entity Author { Author_ID, Name };
8  entity Catalogue {};
9  relationship Propose between Catalogue (*)
       and Book (*);
10 relationship Write between Author (*) and
       Book (*);
11 ...
```

Feature: *Book Recommendation*

```
4 entity Book { ISBN, Title, Author_Names };
5 relationship Propose between Catalogue (*)
      and Book (*);
```

Feature: *Room Planning*

```
12 entity Room { ... };
13 entity Course { Capacity };
14 relationship ...;
```

Figure 4: Physical decomposition of a database schema.

The composition mechanism can be explained best in a textual representation[1]. In Figure 4, we show a simple textual representation of the schema from Figure 2 that has been physically decomposed into four files according to our four features. For composition, the according files are selected and *superimposed*. Superimposition is a language-independent composition mechanism that has been applied to various code and non-code languages and was formalized in an algebra [AL08, ALMK08, AKL09]. We can compose all combinations (except composing *Library* with *Book Rec.* as specified in the feature model, see Fig. 1). In our case, composition is simple because the schema language is simple: All elements, stored in different import files (schemas), are assembled in a single file, and elements with the same name are merged (attributes are concatenated with removing duplicates, cardinalities are composed with a special algorithm). Entities can also be repeated if desired. However, the common case is that entities are defined in only one feature and extended (e.g., adding attributes to an entity) by other features. Again the feature model describes the relationships and constraints between features and, thus, the entities as well. The resulting schema for one variant contains the composition of all entities and relations including attributes. Entities may be superimposed and elements may be overridden according the constraints in the feature model. For extended ER models [CRYG07, Bad04, GJ99], further extensions of the composition mechanisms are necessary.

We implemented the composition mechanism for the textual representation as an extension of the general software composition tool suite FeatureHouse [AL08, AKL09]. The extension required 13 lines of code, all remaining implementation can be reused as described in [AL08, AKL09]. With FeatureHouse it is possible to compose schema fragments to a global schema; this makes a physical decomposition along features feasible, and naturally supports alternative features.

[1] Nevertheless, a graphical representation in the ER editor, or the native XMI-based storage format are possible as well. The same mechanism can also be applied to physical schemas or even SQL scripts. In fact, we implemented transformations between various different representations.

## 4 Related Work

Czarnecki and Pietroszek presented a methodology for a mapping features to models [CP06] which was extended and implemented as Eclipse plug-in by Heidenreich et al. [HKW08]. This idea forms the basis for our virtual decomposition approach. In contrast, physical decomposition is rooted in the idea of feature-oriented programming [Pre97, BSR04], in which software is synthesized by composing code units. We adapted these ideas to ER modelling.

Rashid proposed using aspect-oriented programming to support database schema evolution in object-oriented databases [Ras03]. While he also identifies a lack of variability in database schemas, he concentrates only on evolutionary changes and not on the derivation of different variants of the database schema.

View integration or view merging are well studied mechanisms in database research [SE06, SP94, BLN86]. The main focus lies on consistency checking, covering different, incomplete, and overlapping aspects in local views to retrieve a single valid global view or schema. In contrast, we concentrate on generating a number of tailor-made schemas by refining sub-schemas to support variability and customizeability.

Another approach for data tailoring is presented by Bolchini et al. [BQR07]. Their approach aims at generating and composing views to tailor the data of an underlying large database schema for a given context. In contrast to our approach, Bolchini et al. target context dependent data retrieval and do not tailor or change the underlying schema.

Furthermore, work on model composition is highly related to a physical decomposition. Hermann et al. defined on algebraic level an operator for model composition [HKR+07]. Their theorems prove that syntactic and semantic property preserving model compositions can be achieved. In addition, Sabetzadeh et al. use also SPL concepts to describe different variants of models [SSEC07] in the automotive domain. They use views on function nets to model different features of the system. While they describe model composition on an abstract level, we use feature models to enforce the consistency of the schema composition. Moreover, we can refine already existing schemas, which was not considered in their research.

## 5 Conclusions

With increasing variability in client application, there is a growing mismatch between variable applications and conceptual database schemas. We have presented two approaches based on software product line techniques to overcome this mismatch by providing a tailor-made database schema for variable applications. These approaches – virtual and physical decomposition – allow us to decompose a database schema according to a feature model. They overcome the mismatch that conceptual database schemas are usually not aligned with the variability of their client applications. We can tailor a database schema exactly to the requirements of the customer in line with the customized application.

Both approaches have different strength and weaknesses [KAK08]. Therefore, we have adapted both to support ER modeling. First, a virtual decomposition is well suited for decomposing existing database schemas. It makes minimal changes to the development process and can be adopted easier in an industrial scenario [CK02]. In contrast a physical separation is well suited for projects started from scratch, for projects with many alternative features, and projects where third-party developers are involved (as in the Typo3 project). Based on both approaches, automatically checking all possible variants for consistency (well-formedness) is possible [CP06], but outside the scope of this paper.

We offer tool support for both approaches and made them available online (`http://fosd.de/er/`). With this contribution, we want to raise attention to variability in conceptual modeling and stimulate further discussions and development of according modeling tools.

In further work, we plan to compare both approaches in more detail in a practical case study. Furthermore, we aim at other database artifacts as physical schemas or SQL scripts, and plan to implement consistency checks that ensure well-formedness for all variants.

## References


[AKL09] Sven Apel, Christian Kästner, and Christian Lengauer. FeatureHouse: Language-Independent, Automatic Software Composition. In *Proc. Int'l Conf. on Software Engineering*. IEEE Computer Society, 2009.

[AL08] S. Apel and C. Lengauer. Superimposition: A Language-Independent Approach to Software Composition. In *Proc. Int'l Symp. Software Composition*, pages 20–35, 2008.

[ALMK08] S. Apel, C. Lengauer, B. Möller, and C. Kästner. An Algebra for Features and Feature Composition. In *Proc. Int'l Conf. on Algebraic Methodology and Software Technology*, volume 5140 of *LNCS*, pages 36–50, 2008.

[ALS08] S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. *IEEE Transactions on Software Engineering*, 34(2):162–180, 2008.

[Bad04] A. Badia. Entity-Relationship modeling revisited. *SIGMOD Record*, 33(1):77–82, 2004.

[BCK98] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley Longman, 1998.

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[BQR07] C. Bolchini, E. Quintarelli, and R. Rossato. Relational Data Tailoring through View Composition. In *Proc. Int'l Conf.ön Conceptual Modeling*, volume 4801 of *LNCS*, pages 149–164, 2007.

[BSR04] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Trans. Softw. Eng.*, 30(6), 2004.

[CE00] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[Che76] P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[CK02] P. Clements and C. Krueger. Point/Counterpoint: Being Proactive Pays Off/Eliminating the Adoption Barrier. *IEEE Software*, 19(4):28–31, 2002.

[CP06] K. Czarnecki and K. Pietroszek. Verifying Feature-based Model Templates against well-formedness OCL Constraints. In *Proc. Int'l Conf. Generative Programming and Component Engineering*, pages 211–220. ACM, 2006.

[CRYG07] G. Chen, M. Ren, P. Yan, and X. Guo. Enriching the ER model based on discovered association rules. *Information Science*, 177(7):1558–1566, 2007.

[GJ99] H. Gregersen and C. Jensen. Temporal Entity-Relationship Models-A Survey. *IEEE Knowledge and Data Engineering*, 11(3):464–497, 1999.

[HcW08] F. Heidenreich, I. Şavga, and C. Wende. On Controlled Visualisations in Software Product Line Engineering. In *Workshop on Visualisation in Software Product Line Engineering*, 2008.

[HKR+07] C. Herrmann, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel. An Algebraic View on the Semantics of Model Composition. In *European Conf. on Model Driven Architecture*, pages 99–113, 2007.

[HKW08] F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In *Companion Proc. Int'l Conf.ön Software Engineering*, pages 943–944. ACM, 2008.

[JF88] R. Johnson and B. Foote. Designing Reusable Classes. *Object-Oriented Programming*, 1(2), 1988.

[KAK08] C. Kästner, S. Apel, and M. Kuhlemann. Granularity in Software Product Lines. In *Proc. of Intl. Conf. on Software Engineering*, pages 311–320. ACM, 2008.

[Par03] J. Parsons. Effects of Local Versus Global Schema Diagrams on Verification and Communication in Conceptual Data Modeling. *J. Manage. Inf. Syst.*, 19(3):155–183, 2003.

[PGvdL05] K. Pohl, G.Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[Pre97] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proc. Europ. Conf. Object-Oriented Programming*, volume 1241, pages 419–443. Springer Verlag, 1997.

[Ras03] A. Rashid. A Framework for Customisable Schema Evolution in Object-Oriented Databases. In *Proc. Int'l Symp. on Data Engineering and Applications*, pages 342–346. IEEE Computer Society, 2003.

[SE06] M. Sabetzadeh and S. Easterbrook. View Merging in the Presence of Incompleteness and Inconsistency. *Requirements Engineering*, 11(3):174–193, 2006.

[SP94] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *IEEE Knowledgde and Data Engineering*, 6(2):258–274, 1994.

[SSEC07] M. Sabetzadeh, S.Nejati, S. Easterbrook, and M. Chechik. Consistency Checking of Conceptual Models via Model Merging. In *Proc. Int'l Conf. on Requirements Engineering*, pages 221–230. Springer-Verlag, 2007.

[Szy02] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

# Multi-Objective Scheduling for Real-Time Data Warehouses

Maik Thiele Andreas Bader Wolfgang Lehner

Dresden University of Technology
Faculty of Computer Science, Database Technology Group
Nöthnitzer Str. 46
D-01187 Dresden
{maik.thiele,andreas.bader,wolfgang.lehner}@tu-dresden.de

**Abstract:** The issue of write-read contention is one of the most prevalent problems when deploying real-time data warehouses. With increasing load, updates are increasingly delayed and previously fast queries tend to be slowed down considerably. However, depending on the user requirements, we can improve the response time or the data quality by scheduling the queries and updates appropriately. If both criteria are to be considered simultaneously, we are faced with a so-called multi-objective optimization problem. We transformed this problem into a knapsack problem with additional inequalities and solved it efficiently. Based on our solution, we developed a scheduling approach that provides the optimal schedule with regard to the user requirements at any given point in time. We evaluated our scheduling in an extensive experimental study, where we compared our approach with the respective optimal schedule policies of each single optimization objective.

## 1 Introduction

Today's enterprises increasingly have to make real-time decisions concerning their daily operations in response to the fast changes in their business domains. As a result, enterprises switch to real-time data warehouses, where the information is automatically captured and pushed into the data warehouse. This approach represents a clear contrast to traditional data warehouses, where modifications are loaded at defined points on a nightly or weekly basis. The data in real-time data warehouses, however, are subject to permanent modifications (trickle-feed), which induces two options from the user's point of view: 1) outdated or slightly outdated data may be used in order to get faster query results, or 2) only the most current data shall be used, i.e., all modifications are committed before the next query is executed. In abstract terms, the first criterion refers to Quality of Service (QoS), the latter criterion is Quality of Data (QoD). Both abstract criteria can be implemented with various metrics considered, e.g., response time or throughput for the QoS objective and time-based or lag-based metrics for the QoD objective.
The push-based data propagation leads to a continuous stream of updates, which compete for system resources with queries from the user side. The decision on which stream to

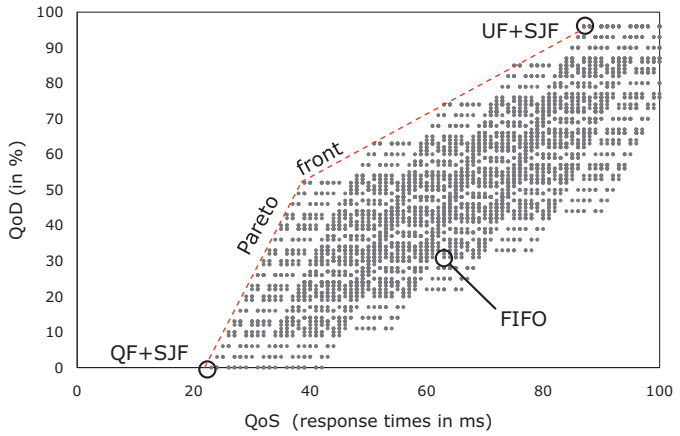



Figure 1: All 7! schedules for a workload with 5 queries and 2 updates

favor depends on the users. For each query submitted to the data warehouse, the users define the required quality-of-data and quality-of-service degree. The data quality is improved by the execution of updates, but this slows down the processing of queries. If only queries are executed, this improves the response time but the data quality gets worse. Hence, both requirements are in conflict with one another and cannot be optimized independently. An improvement of one objective inevitably leads to a decrease of the other objective. Optimizations with regard to competing objectives, such as QoS and QoD, are called multi-objective optimizations. This paper addresses the problem of how to schedule two conflicting types of transactions (queries and updates) in order to assure a user-specified data quality constraint and to keep the contention between queries and updates as low as possible.

**Motivating Example** We will sketch our main principles with an example. For this purpose, we use a workload that consists of 7 transactions: 5 queries $q_1$ to $q_5$ with increasing execution times, and 2 updates $u_1$ and $u_2$. To visualize the solution space for this workload (see Figure 1), we enumerated all possible schedules and evaluated them in terms of two objectives: QoS (e.g., response time) and QoD (e.g., number of unapplied updates). Depending on the execution order of queries and updates, this results in different QoS and QoD values for each schedule. With the focus on one objective, the optimal response time can be achieved by prioritizing the queries based on shortest-job-first (SJF [SM66, Sch68]) and executing the updates subsequently, i.e., queries-first ($QF$). The optimal schedule in terms of data quality is acquired by executing updates before queries. If the queries are additionally sorted by the shortest job first, this results in schedule $UF + SJF$ shown in Figure 1. Scheduling based on the $FIFO$ principle leads to an arbitrarily good or bad scheduling, depending on the arrival times of the transactions, i.e., it does not provide any guarantees.

The simultaneous consideration of both optimization goals, QoS and QoD, results in a front of pareto-efficient schedules (see Figure 1), which connects $QF + SJF$ and $UF + SJF$. The goal is to find the one schedule from the pareto-efficient front that best meets the given user requirements for QoS and QoD.

**Contributions** In detail, our main contribution comprises the following:

- We propose the push-based update propagation in order to build a real-time data warehouse, where the term "real-time" is defined as a user-specified freshness limit.
- We mathematically define the trade-off between the abstract quality-of-service (QoS) and quality-of-data (QoD) criteria and treat it as a multi-objective optimization problem.
- We formulate this optimization problem as a special variant of the knapsack problem and solve it with the help of dynamic programming in pseudo-polynomial time.
- We use this algorithm as a foundation to develop a non-preemptive single-machine scheduling approach in order to minimize the response time and to maximize the data currency.
- We evaluate our approach in various experiments.

Our focus in this paper does not lie on the query scheduling. Instead, we assume a fixed query order, which is prescribed, for example, by query generators or other scheduling policies.

**Structure of the paper** The paper is organized as follows. Our system model is outlined in Section 2. We introduce the concept of pareto-efficient schedules in Section 3, and we provide a dynamic programming algorithm in order to find such schedules. In Section 4, we outline some dynamic aspects regarding the computation of pareto-efficient schedules. Next, in Section 5, we describe the experimental setup and present our experimental results. Section 6 surveys related work. Finally, we conclude in Section 7.

## 2 System Model

Our scenario setup consists of the following three components: 1) a central data warehouse, 2) a staging area to provide a permanent stream of updates trickling into the data warehouse, and 3) some application on top which feeds the data warehouse with user-defined queries. The structure and the elements of the workload as well as the scheduling objectives will be considered in more detail in the subsequent sections.

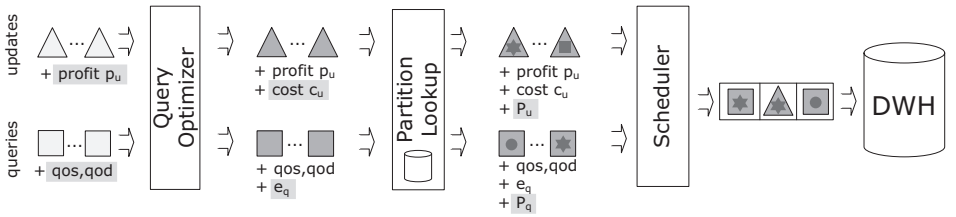



Figure 2: Workload model

### 2.1 Workload Model

The workload $W$ consists of two kinds of transactions: read-only user queries $q_i \in W_q$ and write-only updates $u_j \in W_u$, i.e., an insert, delete, or update, in the following shortly referred to as updates, where $W = W_q \cup W_u$. Mixed transactions do not occur, since the push-based approach implies that both queries and updates are submitted independently to the system.

In order to apply the scheduling algorithm proposed in this paper, queries and updates are associated with a set of parameters that are derived in a pre-processing step illustrated in Figure 2. Each query $q_i$ is annotated by a pair $<qos_{q_i}, qod_{q_i}>$ , which specifies the preferences of the user who issued the query (with $qos_{q_i} \in [0, 1]$ and $qos_{q_i} + qod_{q_i} = 1$). A higher value for $qos_{q_i}$ denotes a higher demand for QoS (e.g., low response times), whereas a higher value for $qod_j$ signifies a higher QoD demand (e.g., few unapplied updates). Each update $u_j$ disposes of a profit parameter $p_{u_j}$, which specifies the user benefit if the update is applied (see Section 2.2.1 for further details). The profit depends on the respective application and can often be calculated easily (e.g., the age of an update or the number of lines to be inserted, etc.).

In order to estimate the execution time of queries and updates, both need to be compiled, which results in a parameter execution time $e_{q_i}$ for each query and a parameter cost $c_{u_j}$ for each update. The compiled query plans are directly used for their later execution. Updates are assumed to be independent from each other to keep the scenario simple. However, our approach can be easily extended by update execution orders. In order to determine which query profits from which update, the dependencies between queries and updates will have to be determined. Therefore, we assume that the data warehouse is divided into a set of partitions. The set of partitions that are accessed by a query or an update can be determined via an efficient lookup. There exists a dependency between a query $q_i$ and an update $u_j$ if the partition sets of both overlap with one another ($P_{q_i} \cap P_{u_j} \neq \emptyset$). A closer look at different partitioning models and their impact on the scheduling quality can be found in [TFL07, TFL08].

In this paper, we build on the parameters described above and focus on the issue of multi-objective scheduling.

### 2.2 Scheduling Performance Objectives

Scheduling in the context of real-time data warehouses requires optimization with regard to two criteria. On the one hand, the term *real-time* represents the demand for updated data. On the other hand, the use of *data warehouses* for analysis purposes implies the desire for fast response times. Both objectives will be defined in more detail and raised to a more abstract level in the following sections.

#### 2.2.1 Quality-of-Service Objective

A lot of research efforts attempt to find good metrics that represent the user experience. In [ZZ96, DK99], the authors found that users accept higher mean response times under the condition that the individual response times show less variance, i.e., the maximum response time is to be minimized. Another widely used objective is the stretch metric, where the response time of a query is additionally normalized with its execution time, which often leads to more adequate results from a user perspective.
In order to measure the QoS, we use the response time, which is typically the primary goal for most computer applications. The response time $rt_{q_i}$ of query $q_i$ is composed of its execution time $e_{q_i}$, the waiting time caused by the execution of preceding queries, and the waiting time caused by the execution of preceding updates (whose cost $c_{u_j}$ is added to the next query to be executed). Thus, the QoS objective for a given workload $W$ is defined as follows:

$$QoS(W) = \sum_{i=0..|W_q|} (|W_q| - i) \cdot (e_{q_i} + c_{u_j}). \quad (1)$$

As can be seen, both queries and updates affect the response time and hence the QoS objective. In order to avoid opposing optimizations with regard to different criteria, the QoS objective for the updates should consider the schedule policy of the queries, i.e., if queries are prioritized with regard to the response time, the resulting QoS objective for the updates should be the response time as well.
Other objectives, for example, the throughput that is to be maximized, have to be transformed accordingly, so that the maximization problem is turned into a minimization problem. Without loss of generality, all metrics outlined above would be suitable QoS objectives. However, in the remainder of this paper, we focus on the response time, which provides a very good indication of the end-user experience, together with the QoD objective to be discussed in the next section.

#### 2.2.2 Quality-of-Data Objective

A variety of metrics exist for the evaluation of the freshness (QoD) of query results: (1) lag-based metrics define the freshness based on the number of non-inserted updates. (2) Divergence-based or value distance metrics quantify the difference in values for the current and the most up-to-date version. (3) Time-differential metrics reflect the delay between a certain query result and the most current change in the real world. Whenever an update to a partition is received, all query results derived from that partition become and remain

stale until the update is executed.
For the approach outlined here, any of the presented metrics as well as any other arbitrary metrics may be used. We only need to make sure that the profit $p_{u_j}$ of an update $u_j$ can be expressed with a positive integer number, where a higher value for the profit corresponds to an increase in the data quality. Thus, if an update $u_j$ is executed before a query $q_i$ that depends on this very update (both access the same partitions: $P_{q_i} \cap P_{u_j} \neq \emptyset$), the data quality of the query $q_i$ is improved by the value specified by the profit. The QoD objective of a workload is then computed as follows:

$$QoD(W) = \sum_{q_i \in W_q,\ u_j \in W_u,\ P_q \cap P_j \neq \emptyset} p_{u_j}. \tag{2}$$

We use the example of the lag-based metric for a more detailed explanation: Here, the profit of each update is encoded by the value 1 ($\forall p_j = 1$) and the set of non-inserted updates is represented by a negative number. The execution of an update before a query would then contribute to the maximization of the data quality (increase of the QoD by 1). Similar to the QoS objective, the minimization problem can be turned into a maximization problem.

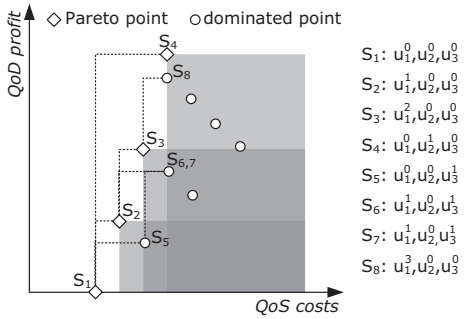
# 3 Multi-Objective Scheduling

In the previous section, we outlined the QoS and QoD objectives individually. In this section, we now focus on the optimization of both objectives at the same time. For this purpose, we first define the concept of pareto-efficient schedules and then develop an efficient algorithm for the computation of such schedules.

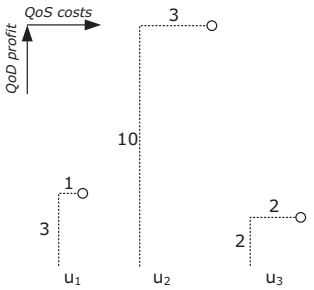
## 3.1 Pareto-Efficient Schedules

In Section 1, we already identified $QF + SJF$ and $UF + SJF$ as pareto-efficient schedules, whose characteristics shall be defined in the following. The basis for our considerations is given by selected schedules from Figure 3 for an example workload that consists of three queries and three updates. All three updates are described by a cost-profit pair, which can be illustrated with the help of two vectors: the cost vector on the x-axis and the profit vector on the y-axis (see Figure 3b). To clearly illustrate this, we assume that each of the three queries profits from all updates. Later on, we will also consider the more general case. The fixed order of the queries shall be defined by an arbitrary scheduling scheme or by user prioritization [KDKK07, SHBIN06, DG95, TFL07, TFL08].
The position of an update $u_j^i$ in a schedule $S$ is denoted by $i$. The value of $i$ is limited by the number of queries in the system $|Q|$, i.e., an update cannot be moved further than by the number of queries present in the workload.

**Definition 3.1 (Schedule $S$)** *A schedule $S$ is defined as the set of all existing updates in the system, $u_j \in U$, with their positions $i$ in the query queue. $S = \{u_j^i \mid \forall j, 0 \leq i \leq |Q|\}$.*




(a) First eight schedules for the given workload




(b) Vector representation of the given updates

Figure 3: Workload $q_1, q_2, q_3, u_1, u_2, u_3$

An update with $i = 0$ is executed after all queries, while an update with $i = |Q|$ is executed before all queries. For example, the schedule $QF + SJF$ is given by $S_1 = (u_1^0, u_2^0, u_3^0)$ (Figure 3a).
The set of all schedules for a given set of queries $Q$ and a given set of updates $U$, and thus, the possible solution space for our optimization problem, should be denoted as $P$.
In order to determine the pareto-efficient schedules, we need the concept of $dominance$, which is defined as follows:

**Definition 3.2 (Dominance $\lhd$)** *If a schedule $S$ is better than another schedule $S'$ (with $S, S' \in P$) in at least one criterion (minimization of cost, maximization of profit), while the second criterion is still at least as good as before, S' is dominated by S. We write $S' \lhd S$ if $c(S) \leq c(S')$, $p(S') \geq p(S)$ and $c(S) \neq c(S')$ or $p(S) \neq p(S')$.*

For example, $S_5$ is dominated by $S_2$ in both criteria and $S_8$ is dominated by $S_4$ with regard to the QoD criterion (Figure 3a).
From the dominance relation, we derive the definition of pareto efficiency:

**Definition 3.3 (Pareto Efficiency)** *Let $S$ and $S' \in P$; then, $S$ is called pareto-efficient if there is no $S'$ with $S \lhd S'$. $P^* = \{S \in P \mid \nexists S' \in P, S \lhd S'\}$.*

Thus, a pareto-efficient schedule guarantees that the only way to further improve a certain criterion is to decrease the result quality for another criterion. For our example, this condition is met for schedules $S_1$, $S_2$, $S_3$ and $S_4$.

### 3.2 Computation of Pareto-Efficient Schedules using a Knapsack Formulation

The task to select those updates that maximize the data quality while keeping the increase in the response time to a minimum can be stated as a 0-1 knapsack problem (i.e., there may be no update at all but never more than one update): We have to choose a subset of $n$

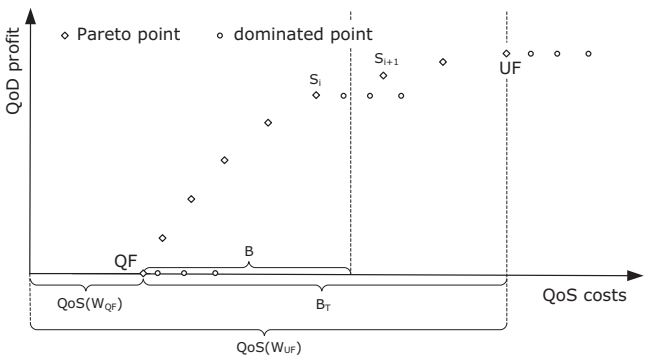



Figure 4: Definition of the maximum knapsack size $B$

updates $u_j$, each with a positive integer profit $p_j$ and positive integer cost $c_j$, such that the corresponding profit sum is maximized without having the sum of the cost exceed a bound $B$:

$$maximize \sum_{j \in |U|} p_j u_j \tag{3}$$

$$subject\ to \sum_{j \in |U|} c_j u_j \leq B \tag{4}$$

$$u_j \in \{0, 1\},\ j = 1, ..., |U|. \tag{5}$$

Maximizing the profit corresponds to a maximization of the data quality by prioritizing updates instead of queries but without exceeding a given response time.

### 3.2.1 Specification of Bound $B$

To calculate bound $B$, which is the available time slot for the execution of updates, we need to know the minimal and maximal response time of a workload. The minimal response time is given by executing the queries before the updates, i.e., by the queries-first principle ($QF$) (see Figure 4). In analogy, the maximum response time is given by executing the updates before the queries, i.e., by the updates-first principle ($UF$). The difference of both values then delivers the maximum time slot $B_T$ that would be necessary to execute all updates first:

$$B_T(W) = QoS(W_{UF}) - QoS(W_{QF}). \tag{6}$$

In order to compute the size of the knapsack regarding the user requirements, we use the mean QoS weights $qos_{q_i}$ of all queries and multiply them by $B_T$ from equation (6) (see Figure 4):

$$B(W) = \frac{B_T}{|W_q|} \sum_{q_i \in W_q} 1 - qos_{q_i}, \quad qos_{q_i} \in [0, 1]. \tag{7}$$

Thus, $B$ is the constraint for the knapsack problem stated in equations (3)-(5). The intuition is that a large bound $B$ allows to put many updates into the knapsack, which increases the QoD and decreases the QoS, whereas a small bound only allows a small number of updates in the knapsack, which improves the QoS but degrades the QoD.
However, we cannot guarantee that a pareto-efficient schedule with a QoS value of exactly the size as the defined knapsack parameter $B$ exists at all. This is shown in Figure 4. Bound $B$ lies between two schedules $S_i$ and $S_i + 1$, with $S_i$ as the result of the knapsack algorithm. The difference between $B$ and the response time $S_i$ depends on the density of the Pareto front, i.e., the number of pareto-efficient schedules $|P^*|$, which is between $n + 1$ and $2^n$. In the former case, the profit and cost of all updates are identical ($p_{u_j} = p_{u_k} \wedge c_{u_j} = c_{u_k} \ \forall u_j, u_k \in W_u$); in the latter case, the profit and cost of the respective updates are identical ($p_{u_j} = c_{u_j} \ \forall u_j \in W_u$). In practice, the number of pareto-efficient schedules lies between these two values, but in any case, it is sufficiently large to ensure that the determined schedule is very close to the specified bound $B$.

### 3.2.2 Generation of Input Items

The knapsack items are derived from the updates $u_j$ that are to be inserted into the query schedule at a certain position. Therefore, we create a dependency matrix $D$ of size $|Q| \times |U|$, that specifies which query profits from which update (see Figure 5). If such a dependency exists, the respective value $u_j^i$ is set to 1. According to the position $i$ of an update $u_j^i$, the profit and cost associated with this update change. An update $u_j^i$ that is to be executed after all queries ($i = 0$) does not incur any cost but neither does it create any profit, since there is no query left to use the updated data. If we move the update accordingly by incrementing $i$, the cost increases with every move but the profit only increases if there is a dependency in the matrix:

$$cost(u_j^k) = c_j \cdot k \tag{8}$$

$$profit(u_j^k) = p_j \cdot \sum_{i=1}^{k} u_j^i. \tag{9}$$

An example for the costs and the profits of an update with the values $c_1 = 5$ and $p_1 = 10$ for different schedule positions is given in Figure 5. Update positions $i$ with $u_j^i = 0$ do not have to be considered, since Definition 3.2 states that they are dominated by other updates $u_j^k$ ($k < i$), which have the same profit but lower costs. In the worst case, i.e., if every query profits from every update, this results in $|Q| \cdot |U|$ input items for the knapsack algorithm. However, our experiments have shown that, in practical scenarios, only very few queries profit from a specific update, which means the number of input items will be significantly smaller.
Thus, every update $u_j$ defines a class $N_j$, whose elements are given by update $u_j$ and their possible positions in the query schedule. In order to ensure that a maximum of one update from class $N_j$ is included in the result set of the knapsack, the knapsack problem is extended by the following condition:

$$\sum_{i \in N_j} u_j^i \leq 1, \ i \in N_j. \tag{10}$$

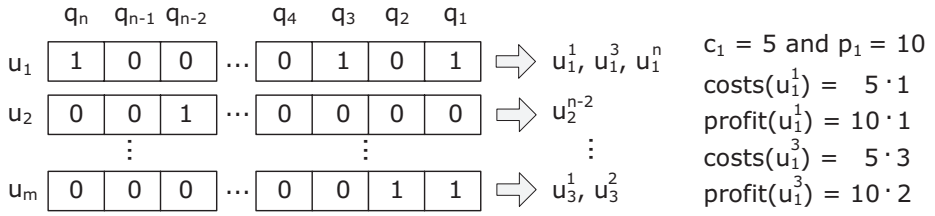



Figure 5: Update-query dependency matrix - example

The problem defined here does not correspond to the Multiple-Choice Knapsack Problem (MCKP) [Nau78], where the requirement is to choose exactly one item per class for the result set.

### 3.2.3 Dynamic Programming Solution

As is known, the knapsack problem belongs to Karp's list of 21 NP-complete problems [Kar72]. However, there are also pseudo-polynomial algorithms for the knapsack problem, which, according to Garey and Johnson [GJ79], "[..] will display 'exponential behavior' only when confronted with instances containing 'exponentially large' numbers."' Hence, in many applications, pseudo-polynomial algorithms behave like polynomial algorithms, as our experiments in Section 5.3 confirm.

To solve our variant of the 0-1 knapsack problem, we make use of a dynamic programming algorithm [Tot80] and extend it accordingly to meet the requirement given in Formula (10) (see Algorithm 1 *UpdatePrioritizing*). As input, our algorithm expects the number of items $N$ (the updates combined with their potential positions in the schedule; see 3.2.2), the bound $B$ (the time slot available for updates; see Section 3.2.1), as well as the profit and cost values for each input item. In addition, we assume that all update items $u_j^i$ are sorted by profit within their class $N_j$ and that the respective order within a class is given through an array $classpos[N+1]$.

We assume that all data are scaled to be integers (see 3.2.2 again). To store the partial solutions, we created an $N+1 \times B+1$-matrix $P$, whose elements are initially set to 0. Let $1 \leq n \leq N$ and $1 \leq c \leq B$; then the value $P[n][c]$ returns the optimal solution for the (partial) knapsack problem. For $P[n][c]$, the following holds: Either the $n$-th item contributes to the maximal profit of the partial problem or it does not (line 3). In case of the former, we get $P[n][c] = profit[n]+P[n-classpos[n]][c-cost[n]]$ (line 4); in case of the latter, we get $P[n][c] = P[n-1][c]$. That is to say, we set $P[n][c] = max(P[n-1][c], p)$ in the algorithm (line 6). A hint on whether or not an item has contributed to a respective partial solution $P[n][c]$ is stored in a second matrix $R$ (line 7).

In the original solution for the 0-1 knapsack problem, only the respective last item per step was considered, i.e., update item $n-1$. However, since we also have to meet the requirement from Formula (10), we thus consider the respective update item with the highest profit from the last class ($n - classpos[n]$; line 4). Thereby, we guarantee that no update is represented more than once in the result set.

**Algorithm 1** UpdatePrioritizing(N, B, profit[N], cost[N])

**Require:** $P[N+1][B+1]$ *// initialized with 0*
$R[N+1][B+1]$ *// initialized with 0*
$classpos[N+1]$ *// position of the update items within their classes*
$result[N]$ *// initialized with false*

```
1: for n = 1 to N do                                   // for each update item
2:   for c = 1 to B do                                 // for each bound c < B
3:     if cost[n] ≤ c then                             // if item fits into c
4:       p = profit[n] + P[n − classpos[n]][c − cost[n]]
5:     end if
6:     P[n][c] = max(P[n − 1][c], p)                   // choose the one with the most profit
7:     R[n][c] = (p > P[n − 1][c])                     // store it as partial solution
8:   end for
9: end for
10: // compute the final result set
11: c = B, n = N                                       // set the counters to the size of the matrix
12: while n > 0 do                                     // from the last to the first item
13:   if R[n][c] then                                  // if it is a partial result
14:     result[n] = true                               // store the item in the final result set
15:     c = c − cost[n]                                // reduce c by the cost of the item
16:     n = n − classpos[n]                            // jump to the last item of the previous class
17:   else
18:     n = n − 1                                      // jump to the previous item
19:   end if
20: end while
```

After both loops have been passed completely, the content of the knapsack can be reconstructed with the maximal profit value by backtracking the calculation of $P$ (lines 10-20). Thus, the algorithm described above returns all updates $u_j$ and their positions $i$ where they contribute most to the solution of the knapsack problem. For all other updates that do not appear in the result set, we set $i = 0$ and add them to the output, that is to say, they would be executed after all queries have been processed. Hence, the result is a pareto-efficient schedule $S$ that delivers the maximum data quality gain for a given response time $B$.
Due to the nested for-loops, which iterate over $N$ and $B$, the algorithm requires a runtime of $O(N \cdot B)$, and due to matrix $P$, it demands $O(N \cdot B)$ of space. For brevity reasons, we omit any further details on the complexity with regard to parameter $B$, but we state that it is bounded by a polynomial in our application.

### 3.3 Static Scheduling Process

Having presented the individual scheduling components, we will summarize them in one comprehensive process model, as illustrated in Figure 6. The continuous streams of queries and updates, respectively, will initially be considered independent from each other. In step (1), the queries are scheduled based on a policy that is optimal for the respective appli-

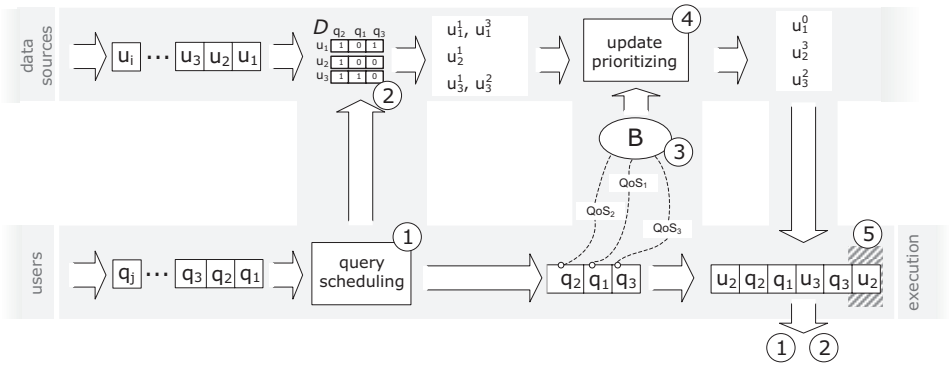



Figure 6: Static scheduling process

cation scenario. In our case, we use an extended shortest-job-first approach (SJF), which increases the priority of queries in dependence on the time they spend waiting in the system to avoid starvation. In step (2), we use the dependency matrix $D$ to extract those input items from the updates and queries existing in the system that shall be used for the solution of the knapsack problem. In order to calculate bound $B$, i.e., the size of the knapsack, step (3) evaluates the user requirements associated with the queries as well as the maximal and minimal response times. During step (4), we execute the *UpdatePrioritizing* algorithm, which then returns the positions of the updates in the query queue that will lead to the maximum quality gain with regard to the user criteria while keeping the response time below the specified bound $B$. Let us point out again that the objective with regard to the QoS cannot be chosen independently from the scheduling policy from (1), but it should match this policy instead. That is to say, if the queries are scheduled in such a way that the response time is minimized (e.g., SJF), this should also be the QoS objective for the knapsack algorithm. According to the positions determined in step (4), we then use step (5) to insert the updates into the query schedule.
So far, we have assumed that the workload is already fully known at the time of the scheduling, that is to say, we have only considered the static case. Thus, we will now take a closer look at various dynamic aspects, i.e., a rescheduling if new jobs arrive or old jobs are executed.

## 4 Dynamic Scheduling

In contrast to static scheduling, the set of queries and updates as well as their processing information are not known a priori in the dynamic scheduling case. Instead, they are added continuously to the data warehouse. For this reason, pareto-efficiency can only be said to exist during the actual processing times, e.g., a pareto-efficient schedule at time $t_1$ is very likely to be dominated by other schedules at a later time $t_2$ (see Definition 3.3). Dynamic factors include the arrival of new queries and updates as well as the processing of existing

ones. In detail, we can differentiate between four cases, each of which will have different effects on the recomputation of the schedule:

- Execution of a query $q_i$: The execution of a query leads to a recomputation of the dependency matrix $D$ (step (2) in Figure 6) if $\exists u_i^j = 1$, for $1 \leq j \leq |U|$. When the respective QoS value is taken away, bound $B$ will have to be updated (step (3)) and subsequently, the update items will have to be recomputed (step (4)).
- Execution of an update $u_j$: When executing an update $u_j$, the respective values $u_j^i$, for $1 \leq i \leq |Q|$, must be deleted in the dependency matrix $D$ and as a consequence, the update items have to be recomputed.
- New query $q_i$: A new query $q_i$ results in the recomputation of the dependency matrix if $\exists u_i^j$, for $1 \leq j \leq |U|$. Additionally, bound $B$ must be updated and the update positions need to be recomputed.
- New update $u_j$: If a new update $u_j$ arrives, matrix $D$ must be updated and the update positions have to be recomputed if $\exists u_j^i$, for $1 \leq i \leq |Q|$; otherwise, the current schedule can still be used.

**Stability Measure** So far, we have shown that we can calculate the pareto-efficient schedule for a given set of queries and updates. Now, we will analyze the stability of these schedules when faced with modifications. In dynamic systems, the degree of difference between a solution at time $t_1$ and a solution at a later time $t_2$ is referred to as *severity of change* [BScU05]. If the severity of change for two solutions, i.e., for two schedules in our case, is considerably high, an instance of the problem is completely unrelated to the next. In order to compare two schedules, we use Definition 3.1 to introduce the following distance function:

$$d(S_1, S_2) = \frac{1}{|S_1 \cap S_2|} \sum_{u_i^{p_1} \in S_1, u_j^{p_2} \in S_2, i=j} |p_1 - p_2|,$$

i.e., the distance is defined as the mean position difference of all updates that exist in both schedules. An example illustrates this: Take schedule $S_1 = (u_1^6, u_2^2, u_3^4)$ and another schedule $S_2 = (u_2^1, u_3^5)$ that exists after the execution of update $u_1$. Thus, the distance $d$ between both schedules is $(|2-1| + |4-5|)/2 = 1$.
In Section 5.4, we will analyze the severity of change for different factors and evaluate how much the individual pareto-efficient schedules for a workload change over the course of the simulation period.

## 5 Experiments

We conducted an experimental study to evaluate 1) the performance with respect to the QoS and QoD objectives and compared to other baseline algorithms, 2) the runtime be-

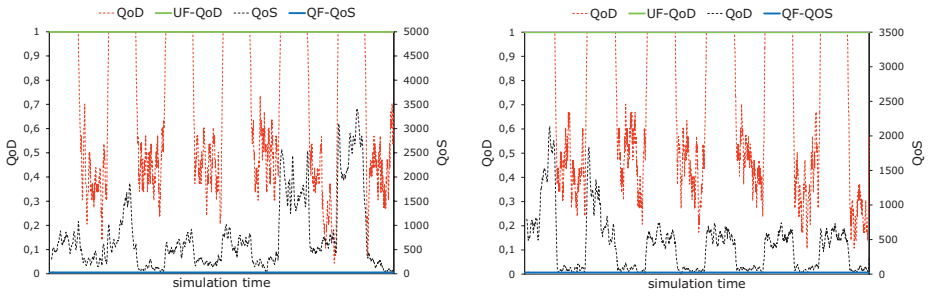
havior of the scheduling algorithm under various workloads, and 3) the severity of change of consecutive pareto-efficient schedules.

### 5.1 Experimental Setup

Our experimental setup consists of a scheduling component, which implements various scheduling policies, and a workload generator. Both are located on the same machine: an Intel Pentium D 2.2 GHz system running Windows XP with 2 GB of main memory. The queries and updates generated with the workload generator can be varied with regard to the different parameters: number of queries and updates, time distance between the addition of transactions, user requirements regarding QoS and QoD, query execution time $e_{q_i}$, update profit $p_{u_j}$, and update cost $c_{u_j}$ (the last three alternatively follow a Gaussian or a Zipf distribution). Furthermore, we can modify the degree of dependency between queries and updates. Thereby, we may create different query types: large range queries that depend on many updates or point queries that depend only on a few updates or on no updates at all.

### 5.2 Performance Comparison and Adaptivity

In the first set of experiments, we investigated the QoS and QoD objectives for different workload types and varying user requirements. Further, we compared the results to two baseline algorithms, $QF$ and $UF$. $QF$ always favors queries over updates and thus minimizes the QoS objective. $UH$ favors updates over queries and thereby maximizes the QoD objective. Thus, both are optimal with regard to the respective objectives. The specific objectives we applied in our experiments include the response time for QoS and the number of unapplied rows for QoD, normalized to the value 1 (i.e., $1/(1 + unapplied_rows)$). First of all, we want to illustrate that pareto-efficient scheduling adapts quickly to changing trends in user behavior. Therefore, we used two kinds of workloads, $W_{GAUSS}$ and $W_{ZIPF}$. Both consist of 5,000 queries and 5,000 updates, whereas the values for execution time, profit and cost are drawn from a Gaussian distribution for $W_{GAUSS}$ (with $\mu_{e_q} = 5,000\ ms$, $\mu_{c_u} = 500\ ms$, $\mu_{p_u} = 50\ rows$ and $\sigma = 1$) and from a Zipf-like distribution for $W_{ZIPF}$ (with $e_q = 10-20,000\ ms$, $c_u = 5-500\ ms$ and $p_u = 1-500\ rows$). For both workloads, we changed the user behavior six times during the workload execution, i.e., we switched $qos_{q_i}$ for all queries from 0 to 1, and vice versa. Figures 7a and 7b plot the QoS and QoD values of the queries that were executed at the respective measurement points. In order to smoothen the data, we applied a moving average with a window size of 30 queries. It can be seen that every change in the user behavior from one extreme (e.g., high data quality) to the other (e.g., fast queries) also results in a scheduling adjustment. High demand for data quality results in a large knapsack size, which leads to stronger prioritization of more updates ($QoD = 1$). The demand for fast query results leads to a smaller knapsack, which means that fewer or no updates at all are executed before queries.




(a) Gaussian Distribution of $e_q$,$c_u$,$p_u$

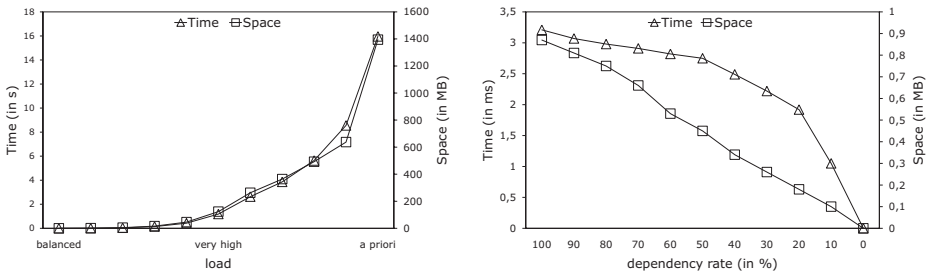
(b) Zipf Distribution of $e_q$,$c_u$,$p_u$

Figure 7: QoS and QoD performance under changing user requirements

Furthermore, it can be seen that the pareto-efficient scheduling is as good as the respective optimal scheduling, $QF$ or $UF$, in the respective phase (see the green and blue lines in Figures 7a and 7b).

### 5.3 Time and Space Consumption

The second set of experiments investigate the time and space consumption that is required for the scheduling. The rate of dependencies between queries and updates was set to a fixed value of 10% - a rather large value for realistic scenarios. The load was decreased by a step-wise shortening of the time span between the addition of 1,000 queries and 1,000 updates (see Figure 8a). For the smallest load, the number of queries and updates added in each step equals the number the DWH is able to process until the next step (*balanced*). For the highest load, all queries and updates were added at once (*a priori*), i.e., there were 2,000 transactions in the system at the same time. Figure 8a shows the average runtime and the average space consumption, both of which show an identical increase with increasing load. For realistic scenarios with a few dozens up to a few hundreds of transactions, we determined runtimes of 2.5 to 150 ms and a memory consumption of 0.1 MB to 15 MB. For 2,000 transactions, the scheduling required 16 seconds and 1,400 MB of memory, which can be neglected in comparison to the runtime of several hours for all 2,000 transactions.
In a second step, we increased the dependency rate between 1,000 queries and 1,000 updates from 0% to 100% (see Figure 8b) and chose a balanced load. It can be seen that a rising number of dependencies leads to a steady increase in both the runtime and the space consumption.
The majority of the computation efforts are directed at the solution of the knapsack problem. This becomes even more complex the more input items are generated. The number of input items depends on both the load and the dependency rate between queries and updates (see 3.2.2). However, for realistic workloads of a few hundred transactions at the same time and dependency rates of 10% on average, the computation overhead can be ne-




(a) Increasing Load

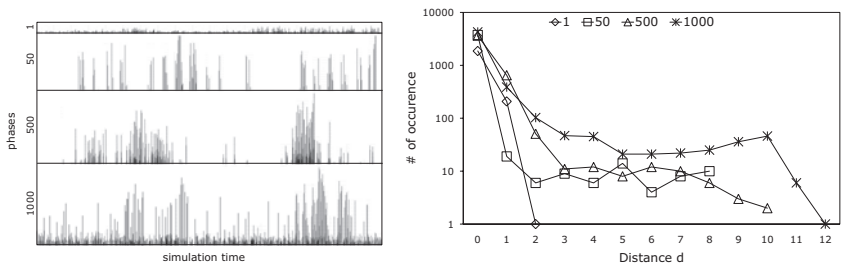
(b) Increasing Query-Update Dependency

Figure 8: Mean execution time and space consumption for the scheduling

glected.
The existing approach does not preserve the update execution order to keep the model simple. An implementation of this constraint would additionally reduce the number of update items, which in turn would reduce the runtime and space complexity.

### 5.4 Evaluation of the Severity of Change

Finally, we examined the severity of change (stability) of pareto-efficient schedules during workload execution. Therefore, we applied the distance measure introduced in Section 4 to all consecutive schedule pairs. We used 5,000 queries and 5,000 updates and switched the user requirements 1, 50, 500 and 1,000 times between $qos_{q_i} = 0$ and $qos_{q_i} = 1$ for all queries $q_i$. Figure 9a shows the development of the distances for the different workloads over the course of the whole simulation time. It can be seen that stable workloads result in very small values for the distances. For frequently changing user requirements, however, the distances between consecutive schedules are significantly larger.
The number of occurrences of distinct integer distances allows us to draw conclusions on the stability of the individual solutions (see Figure 9b). For stable user requirements, the expected result is that a pareto-efficient schedule at time $t_1$ will still (or almost) be pareto-efficient after the subsequent optimization step at time $t_2$. For continuously changing requirements, the consecutive schedules are expected to differ. This is confirmed by the results in Figure 9b, which shows the occurring rounded distance values and their frequencies (in logarithmic scale). For stable requirements (1 change), the schedules barely change at all during the workload execution, i.e., the rounded distance value is usually 0 or 1. However, if the requirements change more often (50, 500 or 1,000 changes), the distance values and their occurrence frequencies increase considerably.
Thus, the expected behavior has been confirmed: As long as the user requirements remain the same, consecutive schedules and the solutions for the knapsack problem, respectively, also remain stable. Heavy changes in the user requirements, however, result in very dif-




(a) Distances over Simulation Time

(b) Distance Occurences

Figure 9: Severity of change under different workloads ($qos_{q_i} = 0 \leftrightarrow 1$)

ferent schedules. Thus, the pareto-efficient scheduling is applicable for the online case as well (see Section 4).

## 6 Related Work

The problems addressed in this paper can be grouped into three larger categories: 1) the consideration of the quality-of-service and quality-of-data criteria as prerequisites for the definition of suitable metrics and objectives, 2) the definition of so-called multi-objective optimization problems and their potential solutions, and 3) the need for optimization of real-time databases and data warehouses regarding various objectives.

**Quality of Service / Data** There exists a variety of work on the definition of suitable QoS metrics in diverse application scenarios, such as multimedia applications, wireless data networks and data stream management. In the field of distributed data management, we particularly mention the Mariposa project [SAL+96] and the work of [BKK03]. The necessity to integrate service quality guarantees in information systems is addressed by [Wei99]. Similarly, there is a lot of work on QoD aspects of databases. Various proposals for data quality metrics can be found in [MR96, FP04] and in [VBQ99] with a special focus on data warehouses.
In our paper, we made use of existing metrics and proposed methods of how to map them to our optimization problem (see Sections 2.2.1 and 2.2.2)

**Multi-Objective Optimizations** The requirement to consider two or more criteria during scheduling processes is widely recognized in the scientific community. [Smi56] was the first to address this problem, focusing on the job completion time and the number of tardy jobs as objectives. Various approaches dealing with multi-objective optimization simplify the problem by tracing it back to a single-objective problem, i.e., they assign

weights to the individual objectives and map them to a common scale [Hug05]. However, this only succeeds if two objectives can be compared with each other, which is not the case in our scenario with the mean response time and the data quality as our objectives. Instead, we either have to compute all pareto-efficient schedules [NU69] and leave it to the user to select the appropriate schedule, or we allow the user to restrict a certain objective to compute the pareto-efficient schedule closest to that user-given bound, which is true for our scenario.
Most publications deal exclusively with the compution of pareto-efficient solutions for the static scheduling problem, where all jobs and their processing information are known a priori. To the best of our knowledge, there is no work on pareto-efficient scheduling in the dynamic case. The evolution of individual pareto-efficient schedules over the course of the simulation time has been analyzed in this paper (see Sections 4 and 5.4).

**Real-Time Databases and Data Warehouses** In order to resolve the conflict between many writing and long-running reading transactions in real-time data warehouses, the approach of isolated external caches or real-time partitions is used rather often [TPL08]. Updates write their modifications into the external cache to avoid the update-query contention problem in the data warehouse. Queries that require the real-time information are partially or completely redirected to the external cache. However, since the majority of the queries increasingly tend to exhibit certain real-time requirements, it is very likely that the real-time partition quickly shows the same query-update contention like the data warehouse.
The subject of scheduling algorithms focusing on one optimization criterion has been discussed extensively in the research community and thus, a variety of works exist; a representative paper is [LKA04]. Scheduling algorithms are often classified as online or offline and as preemptive or non-preemptive algorithms. In this paper, we focus on online and non-preemptive scheduling.
Our update prioritization shares some similarities with the transaction scheduling techniques in real-time database systems [Kan04, KSSA02, HCL93, HJC93]. Such approaches often work with deadline or utility semantics, where a transaction only adds value to the system if it finishes before its deadline expires. Real-time, in our context, refers to the insertion of updates that happens as quickly as possible (close to the change in the real world) or as quickly as needed, respectively, depending on the user requirements.
The data warehouse maintenance process, i.e., the propagation of updates, can be split into two phases: 1) The external maintenance phase denotes the maintenance process between the information sources and the data warehouse or its base tables. 2) The external maintenance phase refers to the process of maintaining materialized views with the base tables used as foundation. In this paper, we focus on phase 1) and assume a model with a sinqle queue and a single thread. That is to say, updates are inserted sequentially and in order of their importance for the query side. The maintenance of materialized views and the various aspects of this discipline, such as incremental maintenance or concurrent updates, are not in the center of attention of this paper.

## 7 Conclusion

Real-time data warehouses have to manage continuous flows of updates and queries and must comply with conflicting requirements, such as short response times versus high data quality. In this paper, we proposed a new approach for the combination of both objectives under given user preferences. First, we raised the objectives to a more abstract level and formulated separate maximization and minimization problems. Based on that, we developed a multi-objective scheduling algorithm that provides the optimal schedule with regard to the user requirements. We evaluated the stability of pareto-efficient schedules under dynamic aspects. The results demonstrated the usability of our approach for online scheduling. Furthermore, we confirmed the time and memory efficiency as well as the adaptability of the proposed scheduling with regard to changing user requirements.
To summarize, we believe that the real-time aspect, recently introduced for data warehouses, implicates an extended user model that describes the varying user demands. Facing a multitude of queries with different or even conflicting demands, we proposed a new approach to schedule the appropriate transactions according to the user requirements.

## References


[BKK03] R. Braumandl, A. Kemper, and D. Kossmann. Quality of service in an information economy. *ACM Trans. Interet Technol.*, 3(4):291–333, 2003.

[BScU05] Jürgen Branke, Erdem Salihoğlu, and Şima Uyar. Towards an analysis of dynamic environments. In *GECCO*, pages 1433–1440, New York, NY, USA, 2005. ACM.

[DG95] Diane L. Davison and Goetz Graefe. Dynamic resource brokering for multi-user query execution. *SIGMOD Rec.*, 24(2):281–292, 1995.

[DK99] Benedict G. C. Dellaert and Barbara E. Kahn. How Tolerable is Delay? Consumers Evaluations of Internet Web Sites after Waiting. *Journal of Interactive Marketing*, 13:41–54, 1999.

[FP04] Chiara Francalanci and Barbara Pernici. Data quality assessment from the user's perspective. In *IQIS*, pages 68–73, New York, NY, USA, 2004. ACM.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[HCL93] Jayant R. Haritsa, Michael J. Carey, and Miron Livny. Value-Based Scheduling in Real-Time Database Systems. *The VLDB Journal*, 2(2):117–152, 1993.

[HJC93] D. Hong, Theodore Johnson, and Sharma Chakravarthy. Real-Time Transaction Scheduling: A Cost Conscious Approach. In Peter Buneman and Sushil Jajodia, editors, *SIGMOD*, pages 197–206. ACM Press, 1993.

[Hug05] Evan J. Hughes. Evolutionary many-objective optimisation: many once or one many? In *Congress on Evolutionary Computation*, pages 222–227, 2005.

[Kan04] Kyoung-Don Kang. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *TKDE*, 16(10):1200–1216, 2004. Senior Member-Sang H. Son and Fellow-John A. Stankovic.

[Kar72] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations. New York: Plenum*, 1972.

[KDKK07] Stefan Krompass, Umeshwar Dayal, Harumi A. Kuno, and Alfons Kemper. Dynamic Workload Management for Very Large Data Warehouses: Juggling Feathers and Bowling Balls. In *VLDB*, pages 1105–1115, 2007.

[KSSA02] Kyoung-Don Kang, Sang Hyuk Son, John A. Stankovic, and Tarek F. Abdelzaher. A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases. In *ECRTS*, pages 203–212, 2002.

[LKA04] Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[MR96] Amihai Motro and Igor Rakov. Estimating the Quality of Data in Relational Databases. In *In Proceedings of the 1996 Conference on Information Quality*, pages 94–106. MIT, 1996.

[Nau78] Robert M. Nauss. The 0-1 knapsack problem with multiple choice constraints. *European Journal of Operational Research*, 2(2):125–131, March 1978.

[NU69] G. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15:494–505, 1969.

[SAL+96] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996.

[Sch68] L. E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:678690, 1968.

[SHBIN06] Bianca Schroeder, Mor Harchol-Balter, Arun Iyengar, and Erich Nahum. Achieving Class-Based QoS for Transactional Workloads. In *ICDE*, page 153, Washington, DC, USA, 2006. IEEE Computer Society.

[SM66] L. E. Schrage and L. W. Miller. The queue M/G/1 with the shortest remaining processing time discipline. *Operations Research*, 14:670684, 1966.

[Smi56] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[TFL07] Maik Thiele, Ulrike Fischer, and Wolfgang Lehner. Partition-based workload scheduling in living data warehouse environments. In *DOLAP*, pages 57–64, New York, NY, USA, 2007. ACM.

[TFL08] Maik Thiele, Ulrike Fischer, and Wolfgang Lehner. Partition-based Workload Scheduling in Living Data Warehouse Environments. *Information Systems*, 34:1–5, 2008.

[Tot80] Paolo Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25:29–45, 1980.

[TPL08] Christian Thomsen, Torben Bach Pedersen, and Wolfgang Lehner. RiTE: Providing On-Demand Data for Right-Time Data Warehousing. In *ICDE*, pages 456–465, 2008.

[VBQ99] Panos Vassiliadis, Mokrane Bouzeghoub, and Christoph Quix. Towards quality-oriented data warehouse usage and evolution. In *CAiSE*, pages 164–179. Springer, 1999.

[Wei99] Gerhard Weikum. Towards guaranteed quality and dependability of information systems. In *Proceedings of the Conference Datenbanksysteme in Buro, Technik und Wissenschaft*, pages 379–409. Springer Verlag, 1999.

[ZZ96] M. Zhou and L. Zhou. How does waiting duration information influence customers' reactions to waiting for services. *J. of Applied Social Psychology*, 26:1702–1717, 1996.

# Formalizing ETL Jobs for Incremental Loading of Data Warehouses

Thomas Jörg and Stefan Dessloch
University of Kaiserslautern,
67653 Kaiserslautern, Germany
{joerg|dessloch}@informatik.uni-kl.de

**Abstract:** Extract-transform-load (ETL) tools are primarily designed for data warehouse loading, i.e. to perform physical data integration. When the operational data sources happen to change, the data warehouse gets stale. To ensure data timeliness, the data warehouse is refreshed on a periodical basis. The naive approach of simply reloading the data warehouse is obviously inefficient. Typically, only a small fraction of source data is changed during loading cycles. It is therefore desirable to capture these changes at the operational data sources and refresh the data warehouse incrementally. This approach is known as *incremental loading*. Dedicated ETL jobs are required to perform incremental loading. We are not aware of any ETL tool that helps to automate this task. In fact, incremental load jobs are handcrafted by ETL programmers so far. The development is thus costly and error-prone.

In this paper we present an approach to the automated derivation of incremental load jobs based on equational reasoning. We review existing Change Data Capture techniques and discuss limitations of different approaches. We further review existing loading facilities for data warehouse refreshment. We then provide *transformation rules* for the derivation of incremental load jobs. We stress that the derived jobs rely on existing Change Data Capture techniques, existing loading facilities, and existing ETL execution platforms.

## 1 Introduction

The Extract-Transform-Load (ETL) system is the foundation of any data warehouse [KR02, KC04, LN07]. The objective of the ETL system is extracting data from multiple, heterogeneous data sources, transforming and cleansing data, and finally loading data into the data warehouse where it is accessible to business intelligence applications.

The very first population of a data warehouse is referred to as *initial load*. During an initial load, data is typically extracted exhaustively from the sources and delivered to the data warehouse. As source data changes over time, the data warehouse gets stale, and hence, needs to be refreshed. Data warehouse refreshment is typically performed in batch mode on a periodical basis. The naive approach to data warehouse refreshment is referred to as *full reloading*. The idea is to simply rerun the initial load job, collect the resulting data, and compare it to the data warehouse content. In this way, the required changes for data warehouse refreshment can be retrieved. Note that it is impractical to drop and recreate the data warehouse since historic data has to be maintained. Full reloading is

obviously inefficient considering that most often only a small fraction of source data is changed during loading cycles. It is rather desirable to capture source data changes and propagate the mere changes to the data warehouse. This approach is known as *incremental loading*.

In general, incremental loading can be assumed to be more efficient than full reloading. However, the efficiency gain comes at the cost of additional development effort. While ETL jobs for initial loading can easily be reused for reloading the data warehouse, they cannot be applied for incremental loading. In fact, dedicated ETL jobs are required for this purpose, which tend to be far more complex. That is, ETL programmers need to create separate jobs for both, initial and incremental loading. Little advice on the design of incremental load jobs is found in literature and we are not aware of any ETL tool that helps to automate this task.

In this paper we explore the following problem: Given an ETL job that performs initial loading, how can an ETL job be derived that performs incremental loading, is executable on existing ETL platforms, utilizes existing Change Data Capture technologies, and relies on existing loading facilities? Our proposed approach to tackle this problem is based on previous work of ours [JD08]. We elaborate on this approach and provide formal models for ETL data and ETL data transformations. We further provide formal transformation rules that facilitate the derivation of incremental load jobs by equational reasoning.

The problem addressed in this paper is clearly related to maintenance of materialized views. Work in this area, however, is partly based on assumptions that do not hold in data warehouse environments and cannot directly be transferred to this domain. We highlight the differences in Section 2 on related work. We review existing techniques with relevance to the ETL system in Section 3. We then introduce a formal model for the description of common ETL transformation capabilities in Section 4. Based on this model we describe the derivation of incremental load jobs from initial load jobs based on equivalence preserving transformation rules in Section 5 and conclude in Section 6.

## 2 Related Work

ETL has received considerable attention in the data integration market; numerous commercial ETL tools are available today [DS, OWB, DWE, IPC]. According to Ralph Kimball seventy percent of the resources needed for the implementation and maintenance of a data warehouse are typically consumed by the ETL system [KC04]. However, the database research community did not give ETL the attention that it received from commercial vendors so far. We present academic efforts in this area in Section 2.1. We then discuss work on the maintenance of materialized views. This problem is clearly related to the problem of loading data warehouses incrementally. However, approaches to the maintenance of materialized views are based on assumptions that do not hold in data warehouse environments. We discuss the differences in Section 2.2.

### 2.1 Related Work on ETL

An extensive study on the modeling of ETL jobs is by Simitsis, Vassiliadis, et al. [VSS02, SVTS05, Sim05, SVS05]. The authors propose both, a conceptual model and a logical model for the representation of ETL jobs. The conceptual model maps attributes of the data sources to the attributes of the data warehouse tables. The logical model describes the data flow from the sources towards the data warehouse. In [Sim05] a method for the translation of conceptual models to logical models is presented. In [SVS05] the optimization of logical model instances with regard to their execution cost is discussed. However, data warehouse refreshment and, in particular, incremental loading has not been studied.

In [AN08] the vision of a generic approach for ETL management is presented. The work is inspired by research on generic model management. The authors introduce a set of high-level operators for ETL management tasks and recognize the need for a platform- and tool-independent model for ETL jobs. However, no details of such a model are provided.

Most relevant to our work is the Orchid project [DHW+08]. The Orchid system facilitates the conversion from schema mappings to executable ETL jobs and vice versa. Schema mappings capture correspondences between source and target schema items in an abstract manner. Business users well understand these correspondences but usually do not have the technical skills to design appropriate ETL jobs.

Orchid translates schema mappings into executable ETL jobs in a two-step process. First, the schema mappings are translated into an intermediate representation referred to as Operator Hub Model (OHM) instance. The OHM captures the transformation semantics common to both, schema mappings and ETL jobs. Second, the OHM instance is translated into an ETL job tailored for the ETL tool of choice. However, jobs generated by Orchid are suited for initial loading (or full reloading) only. We adopt Orchid's OHM to describe ETL jobs in an abstract manner and contribute an approach to derive incremental load jobs from initial load jobs. By doing so, we can reuse the Orchid system for the deployment of ETL jobs. We thus complement Orchid with the ability to create ETL jobs for incremental loading.

### 2.2 Related Work on Maintenance of Materialized Views

Incremental loading of data warehouses is related to incremental maintenance of materialized views, because, in either case, physically integrated data is updated incrementally. Approaches for maintaining materialized views are, however, not directly applicable to data warehouse environments for several reasons.

- Approaches for maintaining materialized views construct maintenance expressions in response to changes at transaction commit time [GL95, QW91, GMS93]. Maintenance expressions assume access to the unchanged state of all base relations and the net changes of the committing transactions. ETL jobs for incremental loading, in contrast, operate in batch mode for efficiency reasons and are repeated on a peri-

odical basis. Hence, any possible change has to be anticipated. More importantly, source data is only available in its most current state unless the former state is explicitly preserved.

- A materialized view and its source relations are managed by the same database system. In consequence, full information about changes to source relations is available. In data warehouse environments Change Data Capture techniques are applied that may suffer from limitations and miss certain changes as stated in Section 3.2. In Section 4 we will introduce a notion of *partial change data* to cope with this situation. We are not aware of any other approach that considers limited access to change data.
- In literature, a data warehouse is sometimes regarded as a set of materialized views defined on distributed data sources [AASY97, ZGMHW95, Yu06, QGMW96]. We argue that this notion disregards an important aspect of data warehousing since the data warehouse keeps a history of data changes. Materialized views, in contrast, reflect the most current state of their base relations only. Change propagation approaches in the context of materialized views typically distinguish two types of data modifications, i.e. insertions and deletions [AASY97, GL95, GMS93, QW91, ZGMHW95, BLT86, CW91]. Updates are treated as a combination of both, insertions and deletions. The initial state of updated tuples is propagated as deletion while the current state of updated tuples is propagated as insertion. Materialized views are maintained by at first performing deletions and subsequently performing insertions that have been propagated from the sources. Data warehouse dimensions, however, keep a history of data changes. In consequence, deleting and reinserting tuples will not lead to the same result as updating tuples in place, in terms of the data history. Therefore, our approach to incremental data warehouse loading handles updates separately from insertions and deletions.

# 3 The ETL Environment

In this Section we review existing techniques that shape the environment of the ETL system and are thus relevant for ETL job design. We first introduce the dimensional modeling methodology that dictates data warehouse design. We then discuss techniques with relevance to incremental loading. First, we review so called Change Data Capture techniques that allow for detecting source data changes. Second, we describe loading facilities for data warehouses. In absence of an established term, we refer to these techniques as Change Data Application techniques.

## 3.1 Dimensional Modeling

Dimensional modeling is an established methodology for data warehouse design and is widely used in practice [KR02]. The dimensional modeling methodology dictates both,

the logical schema design of the data warehouse and the strategy for keeping the history of data changes. Both parts are highly relevant to the design of ETL jobs.

A database schema designed according to the rules of dimensional modeling is referred to as a *star schema* [KR02]. A star schema is made up of so called fact tables and dimension tables. Fact tables store measures of business processes that are referred to as *facts*. Facts are usually numeric values that can be aggregated. Dimension tables contain rich textual descriptions of the business entities. Taking a retail sales scenario as an example, facts may represent sales transactions and provide measures like the sales quantity and dollar sales amount. Dimensions may describe the product being sold, the retail store where it was purchased, and the date of the sales transaction. Data warehouse queries typically use dimension attributes to select, group, and aggregate facts of interest. We emphasize that star schemas typically are not in third normal form. In fact, dimensions often represent multiple hierarchical relationships in a single table. Products roll up into brands and then into categories, for instance. Information about products, brands, and categories is typically stored within the same dimension table of a star schema. That is, dimension tables are highly denormalized. The design goals are query performance, user understandability, and resilience to changes that come at the cost of data redundancy.

In addition to schema design, the dimensional modeling methodology dictates techniques for keeping the history of data changes. These techniques go by the name of *Slowly Changing Dimensions* [KR02, KC04]. The basic idea is to add a so called *surrogate key column* to dimension tables. Facts reference dimensions using the surrogate key to establish a foreign key relationship. Surrogate keys are exclusively controlled by the data warehouse. Their sole purpose is making dimension tuples uniquely identifiable while there value is meaningless by definition.

Operational data sources typically manage primary keys referred to as *business keys*. It is common to assign a unique number to each product in stock, for instance. Business keys are not replaced by surrogate keys. In fact, both, the business key and the surrogate key are included in the dimension table. In this way, dimension tuples can easily be traced back to the operational sources. This ability is known as *data lineage*.

When a new tuple is inserted into the dimension table a fresh surrogate key is assigned. The more interesting case occurs when a dimension tuple is updated. Different actions may be taken depending on the particular columns that have been modified. Changing a product name, for example, could be considered as an error correction; hence the corresponding name in the data warehouse is simply overwritten. This case is referred to as Slowly Changing Dimensions Type I. Increasing the retail price, in contrast, is likely considered to be a normal business activity and a history of retail prices is kept in the data warehouses. This case is referred to as Slowly Changing Dimensions Type II. The basic idea is to leave the outdated tuple in place and create a new one with the current data. The warehouse assigns a fresh surrogate key to the created tuple to distinguish it from its expired versions. Note that the complete history can be retrieved by means of the business key that remains constant throughout time. Besides the surrogate key there are further "special purpose" columns potentially involved in the update process. For instance, an effective timestamp and an expiration timestamp may be assigned, tuples may hold a

reference to their preceding version, and a flag may be used to indicate the most current version.

In summary, the dimensional modeling methodology impacts the ETL job design in two ways. First, it dictates the shape of the target schema, and more importantly it requires the ETL job to handle "special purpose" columns in the correct manner. We emphasize that the choice for Slowly Changing Dimensions Type I or II requires knowledge of the initial state and the current state of updated tuples.

### 3.2 Change Data Capture

Change Data Capture (CDC) is a generic term for techniques that monitor operational data sources with the objective of detecting and capturing data changes of interest [KC04, BT98]. CDC is of particular importance for data warehouse maintenance. With CDC techniques in place, the data warehouse can be maintained by propagating changes captured at the sources. CDC techniques applied in practice roughly follow three main approaches, namely log-based CDC, utilization of audit columns, and calculation of snapshot differentials [KC04, BT98, LGM96].

*Log-based* CDC techniques parse system logs and retrieve changes of interest. These techniques are typically employed in conjunction with database systems. Virtually all database systems record changes in transaction logs. This information can be leveraged for CDC. Alternatively, changes may be explicitly recorded using database triggers or application logic for instance.

Operational data sources often employ so called *audit columns*. Audit columns are appended to each tuple and indicate the time at which the tuple was modified for the last time. Usually timestamps or version numbers are used. Audit columns serve as the selection criteria to extract changes that occurred since the last incremental load process. Note that deletions remain undetected.

The *snapshot differential* technique is most appropriate for data that resides in unsophisticated data sources such as flat files or legacy applications. The latter typically offer mechanisms for dumping data into files but lack advanced query capabilities. In this case, changes can be inferred by comparing a current source snapshot with a snapshot taken at a previous point in time. A major drawback of the snapshot differential approach is the need for frequent extractions of large data volumes. However, it is applicable to virtually any type of data source.

The above mentioned CDC approaches differ not only in their technical realization but also in their ability to detect changes. We refer to the inability to detect certain types of changes as *CDC limitation*. As mentioned before deletions cannot be detected by means of audit columns. Often a single audit column is used to record the time of both, record creation and modification. In this case insertions and updates are indistinguishable with respect to CDC. Another limitation of the audit columns approach is the inability to retrieve the initial state of records that have been updated. Interestingly, existing snapshot differential implementations usually have the same limitation. They do not provide the initial state

of updated records while this would be feasible in principle, since the required data is available in the snapshot taken during the previous run. Log-based CDC approaches in practice typically capture all types of changes, i.e. insertions, deletions, and the initial and current state of updated records.

### 3.3 Change Data Application

We use the term *Change Data Application* to refer to any technique appropriate for updating the data warehouse content. Typically a database management system is used to host the data warehouse. Hence, available CDA techniques are DML statements issued via the SQL interface, proprietary bulk load utilities, or loading facilities provided by ETL tools. Note that CDA techniques have different requirements with regard to their input data.

Consider the SQL MERGE statement[1]. The MERGE statement inserts or updates tuples; the choice depends on the tuples existing in the target table, i.e. if there is no existing tuple with equal primary key values the new tuple is inserted; if such a tuple is found it is updated. There are bulk load utilities and ETL loading facilities that are equally able to decide whether a tuple is to be inserted or updated depending on its primary key. These techniques ping the target table to determine the appropriate operation. Note that the input to any of these techniques is a single dataset that contains tuples to be inserted and updated in a joint manner.

From a performance perspective, ETL jobs should explicitly separate data that is to be updated from data that is to be inserted to eliminate the need for frequent table lookups [KC04]. The SQL INSERT and UPDATE statements do not incur this overhead. Again, there are bulk load utilities and ETL loading facilities with similar properties. Note that these CDA techniques differ in their requirements from the ones mentioned before. They expect separate datasets for insertions and updates. However, deletions have to be separated in either case to be applied to the target dataset using SQL DELETE statements, for instance.

In Section 3.1 we introduced the Slowly Changing Dimensions (SCD) approach that is the technique of choice to keep a history of data changes in the data warehouse. Recall, that the decision whether to apply SCD strategy type I or type II depends on the columns that have been updated within a tuple. That is, one has to consider both, the initial state and the current state of updated tuples. Hence, SCD techniques need to be provided with both datasets unless the costs for frequent dimension lookups are acceptable.

In summary, CDA techniques differ in their *requirements* with regard to input change data very much like CDC techniques face varying limitations. While some CDA techniques are able to process insertions and updates provided in an indistinguishable manner, others demand for separated data sets. Sophisticated CDA techniques such as SCD demand for both, the initial state and the current state of updates tuples.

[1] The MERGE statement (also known as upsert) has been introduced with the SQL:2003 standard.

## 4 Modeling ETL Jobs

To tackle the problem of deriving incremental load jobs we introduce a model for ETL jobs in this section. We first specify a model for data and data changes and afterwards provide a model for data transformations. We adopt the relational data model here since the vast majority of operational systems organize data in a structured manner[2]. In the following we use the term *relation* to refer to any structured datasets. We do not restrict this term to relational database tables but include other structured datasets such as flat files, for instance.

Formally, relations are defined as follows. Let $\mathcal{R}$ be a set of relation names, $\mathcal{A}$ a set of attribute names, and $\mathcal{D}$ a set of domains, i.e. sets of atomic values. A relation is defined by a relation name $R \in \mathcal{R}$ along with a relation schema. A relation schema is a list of attribute names and denoted by $sch(R) = (A_1, A_2, \ldots, A_n)$ with $A_i \in \mathcal{A}$. We use the function $dom : \mathcal{A} \rightarrow \mathcal{D}$ to map an attribute name to its domains. The domain of a relation $R$ is defined as the Cartesian product of its attributes' domains and denoted by $dom(R) := dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$ with $A_i \in sch(R)$. The data content of a relation is referred to as the relation's state. The state $r$ of a relation $R$ is a subset of its domain, i.e. $r \subseteq dom(R)$. Note that the state of a relation may change over time as data is modified. In the following we use $r_{\text{new}}$ and $r_{\text{old}}$ to denote the current state of a relation and the state at the time of the previous incremental load, respectively.

During incremental loading the data changes that occurred at operational data sources are propagated towards the data warehouse. We introduce a formal model for the description of data changes referred to as *change data*. Change data specifies how the state of an operational data source has changed during one loading cycle. Change data is captured at tuple granularity and consists of four sets of tuples namely, the set of inserted tuples (insert), the set of deleted tuples (delete), the set of updated tuples in their current state (update new), and the set of updated tuples in their initial state (update old). We stress that change data serves as both, the model for the output of CDC techniques and the model for the input of CDA techniques. Below, we provide a formal definition of change data. We make use of the relational algebra projection operator denoted by $\pi$.

**Definition 4.1.** Given relation $R$, the current state $r_{\text{new}}$ of $R$, the previous state $r_{\text{old}}$ of $R$, the list of attribute names $S := sch(R)$, and the list of primary key attribute names $K \subseteq sch(R)$, *change data* is a four-tuple $(r_{\text{ins}}, r_{\text{del}}, r_{\text{un}}, r_{\text{uo}})$ such that

$$r_{\text{ins}} := \{s \mid s \in r_{\text{new}} \wedge t \in r_{\text{old}} \wedge \pi_K(s) \neq \pi_K(t)\} \tag{4.1}$$

$$r_{\text{del}} := \{s \mid s \in r_{\text{old}} \wedge t \in r_{\text{new}} \wedge \pi_K(s) \neq \pi_K(t)\} \tag{4.2}$$

$$r_{\text{un}} := \{s \mid s \in r_{\text{new}} \wedge t \in r_{\text{old}} \wedge (\pi_K(s) = \pi_K(t) \rightarrow \pi_{S \setminus K}(s) \neq \pi_{S \setminus K}(t))\} \tag{4.3}$$

$$r_{\text{uo}} := \{s \mid s \in r_{\text{old}} \wedge t \in r_{\text{new}} \wedge (\pi_K(s) = \pi_K(t) \rightarrow \pi_{S \setminus K}(s) \neq \pi_{S \setminus K}(t))\}. \tag{4.4}$$

[2]Data in the semi-structured XML format with relevance to data warehousing is typically data-centric, i.e. data is organized in repeating tree structures, mixed content is avoided, the document order does not contribute to the semantics, and a schema definition is available. The conversion of data-centric XML into a structured representation is usually straightforward.

In order to express changes of relations we introduce two partial set operators, namely the disjoint union and the contained difference.

- Let $R$ and $S$ be relations. The *disjoint union* $R \oplus S$ is equal to the set union $R \cup S$ if $R$ and $S$ are disjoint ($R \cap S = \varnothing$). Otherwise it is not defined.
- The *contained difference* $R \ominus S$ is equal to the set difference $R \setminus S$ if $S$ is a subset of $R$. Otherwise it is not defined.

In an ETL environment the state of a relation at the time of the previous incremental load is typically not available. This state can however be calculated given the current state and change data.

**Theorem 4.2.** *Given relation $R$, the current state $r_{\text{new}}$ of $R$, the previous state $r_{\text{old}}$ of $R$, and change data $r_{\text{ins}}$, $r_{\text{del}}$, $r_{\text{un}}$, $r_{\text{uo}}$, the following equations hold*

$$r_{\text{new}} \ominus r_{\text{ins}} \ominus r_{\text{un}} = r_{\text{new}} \cap r_{\text{old}} \tag{4.5}$$

$$r_{\text{new}} \cap r_{\text{old}} \oplus r_{\text{del}} \oplus r_{\text{uo}} = r_{\text{old}} \tag{4.6}$$

$$r_{\text{new}} = r_{\text{old}} \ominus r_{\text{del}} \ominus r_{\text{uo}} \oplus r_{\text{ins}} \oplus r_{\text{un}}. \tag{4.7}$$

*Proof.* We show the correctness of 4.5. Equation 4.6 can be shown in a similarly way. Equation 4.7 follows from 4.5 and 4.6.

$$\begin{aligned}
&r_{\text{new}} \ominus r_{\text{ins}} \ominus r_{\text{un}} = \\
&r_{\text{new}} \ominus (r_{\text{ins}} \oplus r_{\text{un}}) = \\
&r_{\text{new}} \ominus \left\{ s \mid s \in r_{\text{new}} \wedge t \in r_{\text{old}} \wedge \left( \pi_K(s) \neq \pi_K(t) \vee \pi_{S \setminus K}(s) \neq \pi_{S \setminus K}(t) \right) \right\} = \\
&r_{\text{new}} \ominus \{ s \mid s \in r_{\text{new}} \wedge t \in r_{\text{old}} \wedge s \neq t \} \\
&r_{\text{new}} \ominus (r_{\text{new}} \setminus r_{\text{old}}) = r_{\text{new}} \cap r_{\text{old}}
\end{aligned}$$

□

The survey of CDC in Section 3.2 revealed that CDC techniques may suffer from limitations. While Log-based CDC techniques provide complete change data, both the snapshot differential and audit column approaches are unable to capture certain types of changes. We refer to the resulting change data as *partial* change data. Existing snapshot differential implementations often do not capture the initial state of updated tuples (update old). Consequently, this data is unavailable for change propagation; with regard to relation $R$ we say that $r_{\text{uo}}$ is unavailable. The audit column approach misses deletions and the initial state of updated records, i.e. $r_{\text{del}}$ and $r_{\text{uo}}$ are unavailable. In case a single audit column is used to record the time of tuple insertions and subsequent updates, insertions and updates cannot be distinguished. Thus, the best the CDC technique can provide is $r_{\text{ups}} := r_{\text{ins}} \oplus r_{\text{un}}$. Tuples that have been inserted or updated since the previous incremental load are jointly provided within a single dataset (upsert). Note that neither $r_{\text{ins}}$ nor $r_{\text{un}}$ are available though.

We further stressed in Section 3.3 that CDA techniques differ in their requirements with regard to change data. There are CDA techniques capable of consuming upsert sets, for

example. These techniques ping the data warehouse to decide whether to perform insertions or updates. Other techniques need to be provided with separated sets of insertions und updates but work more efficiently. Advanced techniques may additionally require updated tuples in their initial state. In summary, the notion of partial change data is key to the description of both, the output of CDC techniques and the input of CDA techniques. Below, we provide a definition of partial change data.

**Definition 4.3.** Given a relation $R$ we refer to change data as *partial change data* if at least one component $r_{\text{ins}}$, $r_{\text{del}}$, $r_{\text{un}}$, or $r_{\text{uo}}$ is unavailable.

Having established a model for data and change data we are still in need of a model for data transformations. While all major database management systems adhere to the SQL standard, no comparable standard exists in the area of ETL. In fact, ETL tools make use of proprietary scripting languages or visual user interfaces. We adopt OHM [DHW+08] to describe the transformational part of ETL jobs in a platform-independent manner. The OHM is based on a thorough analysis of ETL tools and captures common transformation capabilities. Roughly speaking, OHM operators are generalizations of relational algebra operators. We consider a subset of OHM operators, namely projection, selection, union, and join, which are described in-depth in Section 5.1, Section 5.2, Section 5.3, and Section 5.4, respectively.

**Definition 4.4.** An *ETL transformation expression* $\mathcal{E}$ (or *ETL expression* for short) is generated by the grammar $G$ with the following production rules.

$$
\begin{array}{rr}
\mathcal{E} ::= R & \text{Relation name from } \mathcal{R} \\
\mid \sigma_p(\mathcal{E}) & \text{Selection} \\
\mid \pi_A(\mathcal{E}) & \text{Projection} \\
\mid \mathcal{E} \oplus \mathcal{E} & \text{Disjoint union} \\
\mid \mathcal{E} \ominus \mathcal{E} & \text{Contained difference} \\
\mid \mathcal{E} \bowtie \mathcal{E} & \text{Join}
\end{array}
$$

We impose restrictions on the projection and the union operator, i.e. we consider key-preserving projections and key-disjoint unions only. We show that these restrictions are justifiable in the section on the respective operator. In contrast to [DHW+08] we favor an equational representation of ETL transformations over a graph-based representation. Advanced OHM operators for aggregation and restructuring of data are ignored for the moment and left for future work.

## 5 Incremental Loading

The basic idea of incremental loading is to infer changes required to refresh the data warehouse from changes captured at the data sources. The benefits of incremental loading as

compared to full reloading are twofold. First, the volume of changed data at the sources is typically very small compared to the overall data volume, i.e. less data needs to be extracted. Second, the vast majority of data within the warehouse remains untouched during incremental loading, since changes are only applied where necessary.

We propose a change propagation approach to incremental loading. That is, we construct an ETL job that inputs change data captured at the sources, transforms the change data, and ultimately outputs change data that specifies the changes required within the data warehouse. We refer to such an ETL job as *change data propagation job* or CDP job for short. We stress that CDP jobs are essentially conventional ETL jobs in the sense that they are executable on existing ETL platforms. Furthermore existing CDC techniques are used to provide the input change data and existing CDA techniques are used to refresh the data warehouse.

Recall that CDC techniques may have limitations and hence may provide partial change data (see Section 3.2). Similarly, CDA techniques have varying requirements with regard to change data they can consume (see Section 3.3). The input of CDP jobs is change data provided by CDC techniques; the output is again change data that is fed into CDA techniques. Thus, CDP jobs need to cope with CDC limitations and satisfy CDA requirements at the same time. This is, however, not always possible. Two questions arise. Given CDA requirements, what CDC limitations are acceptable? Or the other way round, given CDC limitations, what CDA requirements are satisfiable?

In the remainder of this section we describe our approach to derive CDP jobs from ETL jobs for initial loading. We assume that the initial load job is given as an ETL expression $\mathcal{E}$ as defined in 4.4. We are thus interested in deriving ETL expressions $\mathcal{E}_{\mathrm{ins}}$, $\mathcal{E}_{\mathrm{del}}$, $\mathcal{E}_{\mathrm{un}}$, and $\mathcal{E}_{\mathrm{uo}}$ that propagate insertions, deletions, the current state of updated tuples, and the initial state of updated tuples, respectively. We refer to these expressions as CDP expressions. In combination, CDP expressions form a CDP job[3]. Note that $\mathcal{E}_{\mathrm{ins}}$, $\mathcal{E}_{\mathrm{del}}$, $\mathcal{E}_{\mathrm{un}}$, or $\mathcal{E}_{\mathrm{uo}}$ may depend on source change data that is unavailable due to CDC limitations. That is, the propagation of insertions, deletions, updated tuples in their current state, or updated tuples in their initial state may not be possible, thus, the overall CDP job may output *partial* change data. In this situation one may choose a CDA technique suitable for partial change data, migrate to more powerful CDC techniques, or simply refrain from propagating the unavailable changes. The dimensional modeling methodology, for instance, proposes to leave dimension tuples in place after the corresponding source tuples have been deleted. Hence, deletions can safely be ignored with regard to change propagation in such an environment.

For the derivation of CDP expressions we define a set of functions $f_{\mathrm{ins}} : L(G) \rightarrow L(G)$, $f_{\mathrm{del}} : L(G) \rightarrow L(G)$, $f_{\mathrm{un}} : L(G) \rightarrow L(G)$, and $f_{\mathrm{uo}} : L(G) \rightarrow L(G)$ that map an ETL expression $\mathcal{E}$ to the CDP expressions $\mathcal{E}_{\mathrm{ins}}$, $\mathcal{E}_{\mathrm{del}}$, $\mathcal{E}_{\mathrm{un}}$, and $\mathcal{E}_{\mathrm{uo}}$, respectively. These functions can however not directly be evaluated. Instead, we define equivalence preserving transformation rules that "push" these functions into the ETL expression and thereby transform the expression appropriately. After repeatedly applying applicable transformation rules, these functions will eventually take a relation name as input argument instead

[3]CDP expressions typically share common subexpressions, hence it is desirable to combine them into a single ETL job.

of a complex ETL expression. At this point, the function simply denotes the output of the relation's CDC system.

We provide transformation rules for projection, selection, union, and join in the subsequent sections and explain the expression derivation process by an example in Section 5.5.

### 5.1 Key-preserving Projection

Our notion of an ETL projection operator generalizes the classical relational projection. Besides dropping columns, we allow for adding and renaming columns, assigning constant values, and performing value conversions. The latter may include sophisticated transformations, such as parsing and splitting up free-form address fields. Our only assumptions are that any value transformation function processes a single tuple at a time and computes its output in a deterministic manner.

We further restrict our considerations to *key-preserving* projections since dropping key columns is generally unwanted during ETL processing [KR02, KC04]. We highlighted the importance of business keys in Section 3.1 on dimensional modeling. Business keys are the primary means of maintaining data lineage. Moreover, in the absence of business keys, it is impractical to keep a history of data changes at the data warehouse. Hence, business keys must not be dropped by the ETL job. Whenever the ETL job joins data from multiple sources, the business key is composed of (a subset of) the key columns of the source datasets. Consider that the key columns of at least one dataset have been dropped. In this case, the join result will just as well lack key columns. That is, the propagation of business keys renders impossible if at any step of the ETL job a dataset without key columns is produced. We thus exclude any non-key-preserving projection from our considerations. For this reason implicit duplicate elimination is never performed by the projection operator.

In view of the above considerations we formulate the following transformation rules. The proofs are straightforward and therefore omitted.

$$f_{\text{ins}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{ins}}\left(\mathcal{E}\right)\right) \tag{5.1}$$

$$f_{\text{del}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{del}}\left(\mathcal{E}\right)\right) \tag{5.2}$$

$$f_{\text{un}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{un}}\left(\mathcal{E}\right)\right) \tag{5.3}$$

$$f_{\text{uo}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{uo}}\left(\mathcal{E}\right)\right) \tag{5.4}$$

$$f_{\text{ups}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{ups}}\left(\mathcal{E}\right)\right) \tag{5.5}$$

$$f_{\text{new}}\left(\pi_A\left(\mathcal{E}\right)\right) \rightsquigarrow \pi_A\left(f_{\text{new}}\left(\mathcal{E}\right)\right) \tag{5.6}$$

### 5.2 Selection

The selection operator filters those tuples for which a given boolean predicate $p$ holds and discards all others. In the light of change propagation three cases are to be distinguished.

- Inserts that satisfy the filter predicate are propagated while inserts that do not satisfy the filter predicate are dropped.
- Deletions that satisfy the filter predicate are propagated since the tuple to delete has been propagated towards the data sink before. Deletions that do not satisfy the filter predicate lack this counterpart and are dropped.
- Update pairs, i.e. the initial and the current value of updated tuples, are propagated if both values satisfy the filter predicate and dropped if neither of them does so. In case only the current value satisfies the filter predicate while the initial value does not, the update turns into an insertion. The updated tuple has not been propagated towards the data sink before and hence is to be created. Similarly, an update turns into a deletion if the initial value satisfies the filter predicate while the current value fails to do so.

  If the initial value of updated records is unavailable due to partial change data it cannot be checked against the filter predicate. Thus, it is not possible to conclude whether the updated source tuple has been propagated to the data sinks before. It is thus unclear whether an insert or an update is to be issued, provided that the current value of the updated tuple satisfies the filter predicate. Hence, inserts and updates can only be propagated in a joint manner in this situation (see 5.11).

We formulate the following transformation rules for the select operator. We make use of the semijoin operator denoted by $\ltimes$ and the antijoin operator denoted by $\overline{\ltimes}$ known from relational algebra.

$$f_{\text{ins}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{ins}}\left(\mathcal{E}\right)\right) \oplus \left[\sigma_p\left(f_{\text{un}}\left(\mathcal{E}\right)\right) \overline{\ltimes} \sigma_p\left(f_{\text{uo}}\left(\mathcal{E}\right)\right)\right] \tag{5.7}$$

$$f_{\text{del}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{del}}\left(\mathcal{E}\right)\right) \oplus \left[\sigma_p\left(f_{\text{uo}}\left(\mathcal{E}\right)\right) \overline{\ltimes} \sigma_p\left(f_{\text{un}}\left(\mathcal{E}\right)\right)\right] \tag{5.8}$$

$$f_{\text{un}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{un}}\left(\mathcal{E}\right)\right) \ltimes \sigma_p\left(f_{\text{uo}}\left(\mathcal{E}\right)\right) \tag{5.9}$$

$$f_{\text{uo}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{uo}}\left(\mathcal{E}\right)\right) \ltimes \sigma_p\left(f_{\text{un}}\left(\mathcal{E}\right)\right) \tag{5.10}$$

$$f_{\text{ups}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{ups}}\left(\mathcal{E}\right)\right) \tag{5.11}$$

$$f_{\text{new}}\left(\sigma_p\left(\mathcal{E}\right)\right) \rightsquigarrow \sigma_p\left(f_{\text{new}}\left(\mathcal{E}\right)\right) \tag{5.12}$$

*Proof.* We show the correctness of (5.7). We make use of the fact that $\pi_K\left(\mathcal{E}_{\text{ins}}\right)$ and $\pi_K\left(\mathcal{E}_{\text{old}}\right)$ are disjoint and $\mathcal{E}_{\text{un}}$ shares common keys with the updated tuples $\mathcal{E}_{\text{uo}} \subseteq \mathcal{E}_{\text{old}}$ only. Equation (5.8), (5.9), and (5.10) can be shown in a similar way.

$$\begin{aligned}
&f_{\text{ins}}\left(\sigma_p\left(\mathcal{E}\right)\right) \overset{4.1}{=} \\
&\left\{r \mid r \in \sigma_p\left(\mathcal{E}_{\text{new}}\right) \wedge s \in \sigma_p\left(\mathcal{E}_{\text{old}}\right) \wedge \pi_K\left(r\right) \neq \pi_K\left(s\right)\right\} \overset{4.7}{=} \\
&\left\{r \mid r \in \sigma_p\left(\mathcal{E}_{\text{old}} \ominus \mathcal{E}_{\text{del}} \ominus \mathcal{E}_{\text{uo}} \oplus \mathcal{E}_{\text{ins}} \oplus \mathcal{E}_{\text{un}}\right) \wedge s \in \sigma_p\left(\mathcal{E}_{\text{old}}\right) \wedge \pi_K\left(r\right) \neq \pi_K\left(s\right)\right\} = \\
&\left\{r \mid r \in \sigma_p\left(\mathcal{E}_{\text{old}} \ominus \mathcal{E}_{\text{del}} \ominus \mathcal{E}_{\text{uo}}\right) \wedge s \in \sigma_p\left(\mathcal{E}_{\text{old}}\right) \wedge \pi_K\left(r\right) \neq \pi_K\left(s\right)\right\} \oplus \\
&\left\{r \mid r \in \sigma_p\left(\mathcal{E}_{\text{ins}}\right) \wedge s \in \sigma_p\left(\mathcal{E}_{\text{old}}\right) \wedge \pi_K\left(r\right) \neq \pi_K\left(s\right)\right\} \oplus \\
&\left\{r \mid r \in \sigma_p\left(\mathcal{E}_{\text{un}}\right) \wedge s \in \sigma_p\left(\mathcal{E}_{\text{old}}\right) \wedge \pi_K\left(r\right) \neq \pi_K\left(s\right)\right\} = \\
&\varnothing \oplus \sigma_p\left(\mathcal{E}_{\text{ins}}\right) \oplus \left(\sigma_p\left(\mathcal{E}_{\text{un}}\right) \overline{\ltimes} \sigma_p\left(\mathcal{E}_{\text{uo}}\right)\right)
\end{aligned}$$

□

We omit proofs for (5.11) and (5.12).

### 5.3 Key-disjoint Union

We already emphasized the importance of business keys for maintaining data lineage and keeping a history of data changes at the data warehouse. Business keys must "survive" the ETL process and we therefore restrict our considerations to key-preserving unions. That is, the result of a union operation must include unique key values. In consequence we require the source relations to be disjoint with regard to their key values. This property can be achieved by prefixing each key value with a unique identifier of its respective data source. The key-disjoint union is a special case of the disjoint union.

$$f_{\text{ins}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{ins}}(\mathcal{E}) \oplus f_{\text{ins}}(\mathcal{F}) \tag{5.13}$$
$$f_{\text{del}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{del}}(\mathcal{E}) \oplus f_{\text{del}}(\mathcal{F}) \tag{5.14}$$
$$f_{\text{un}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{un}}(\mathcal{E}) \oplus f_{\text{un}}(\mathcal{F}) \tag{5.15}$$
$$f_{\text{uo}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{uo}}(\mathcal{E}) \oplus f_{\text{uo}}(\mathcal{F}) \tag{5.16}$$
$$f_{\text{ups}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{ups}}(\mathcal{E}) \oplus f_{\text{ups}}(\mathcal{F}) \tag{5.17}$$
$$f_{\text{new}}(\mathcal{E} \oplus \mathcal{F}) \rightsquigarrow f_{\text{new}}(\mathcal{E}) \oplus f_{\text{new}}(\mathcal{F}) \tag{5.18}$$

### 5.4 Join

The majority of transformation rules seen so far map input change data to output change data of the same type. Exceptions are (5.7), (5.8), (5.9), (5.10) on the selection operator. Here updates may give rise to insertions and deletions depending on the evaluation of the filter predicate. The situation is somewhat more complex for the join operator. We need to distinguish between one-to-many and many-to-many joins. Consider two (derived) relations $\mathcal{E}$ and $\mathcal{F}$ involved in a foreign key relationship where $\mathcal{E}$ is the referencing relation and $\mathcal{F}$ is the referenced relation. Say, $\mathcal{E}$ and $\mathcal{F}$ are joined using the foreign key of $\mathcal{E}$ and the primary key of $\mathcal{F}$. Then tuples in $\mathcal{E} \bowtie \mathcal{F}$ are functionally dependent on $\mathcal{E}$'s primary key. In particular, for each tuple in $\mathcal{E}_{\text{un}}$ at most one join partner is found in $\mathcal{F}$ even if the foreign key column has been updated. It is therefore appropriate to propagate updates. The situation is different for many-to-many joins, i.e. no column in the join predicate is unique. Here, multiple new join partners may be found in response to an update of a column in the join predicate. Moreover, multiple former join partners may be lost at the same time. In consequence, multiple insertions are propagated, one for each new join partner and multiple deletions are propagated, one for each lost join partner. The number of insertions and deletions may differ.

| | $E_{new} \cap E_{old}$ | $E_{\text{ins}}$ | $E_{\text{del}}$ | $E_{\text{un}}$ | $E_{\text{uo}}$ |
|---|---|---|---|---|---|
| $F_{new} \cap F_{old}$ | | ins | del | un | uo |
| $F_{\text{ins}}$ | | ins | | un | |
| $F_{\text{del}}$ | | | del | | uo |
| $F_{\text{un}}$ | un | ins | | un | |
| $F_{\text{uo}}$ | uo | | del | | uo |

Figure 1: Matrix Representation 1-to-n Joins

The matrix shown in Figure 1 depicts the interdependencies between change data of the referencing relation $\mathcal{E}$ and the referenced relation $\mathcal{F}$ assuming referential integrity. The matrix is filled in as follows. The column headers and row headers denote change data of relation $\mathcal{E}$ and $\mathcal{F}$, respectivelly. From left to right, the column headers denote the unmodified tuples of $\mathcal{E}$ ($\mathcal{E}_{\text{new}} \cap \mathcal{E}_{\text{old}}$), insertions $\mathcal{E}_{\text{ins}}$, deletions $\mathcal{E}_{\text{del}}$, updated records in their current state $\mathcal{E}_{\text{un}}$, and updated records in their initial state $\mathcal{E}_{\text{uo}}$. The row headers are organized in a similar way form top to bottom. Columns and rows intersect in cells. Each cell represents the join of two datasets indicated by the row and column headers. The caption of a cell shows whether the join contributes to either $f_{\text{ins}}(\mathcal{E} \bowtie \mathcal{F})$, $f_{\text{del}}(\mathcal{E} \bowtie \mathcal{F})$, $f_{\text{un}}(\mathcal{E} \bowtie \mathcal{F})$, $f_{\text{uo}}(\mathcal{E} \bowtie \mathcal{F})$, or to none of them (empty cell). Take the $\mathcal{E}_{\text{ins}}$ column as an example. The column contains five cells out of which three are labeled as insertions and two are empty. That means the three joins $\mathcal{E}_{\text{ins}} \bowtie (\mathcal{F}_{\text{new}} \cap \mathcal{F}_{\text{old}})$, $\mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{ins}}$, and $\mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{un}}$ contribute to $f_{\text{ins}}(\mathcal{E} \bowtie \mathcal{F})$. Since no other cells are labeled as insertions $f_{\text{ins}}(\mathcal{E} \bowtie \mathcal{F})$ is given by the union of these joins.

$$\begin{aligned} f_{\text{ins}}(\mathcal{E} \bowtie \mathcal{F}) =& \mathcal{E}_{\text{ins}} \bowtie (\mathcal{F}_{\text{new}} \cap \mathcal{F}_{\text{old}}) \oplus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{ins}} \oplus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{un}} \overset{4.5}{=} \\ & \mathcal{E}_{\text{ins}} \bowtie (\mathcal{F}_{\text{new}} \ominus \mathcal{F}_{\text{ins}} \ominus \mathcal{F}_{\text{un}}) \oplus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{ins}} \oplus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{un}} = \\ & \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{new}} \end{aligned}$$

Note, that the above considerations lead to the transformation rule 5.19. The other transformation rules are derived in a similar way.

$$f_{\text{ins}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{new}} \tag{5.19}$$

$$\begin{aligned} f_{\text{del}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow & \mathcal{E}_{\text{del}} \bowtie \mathcal{F}_{\text{new}} \oplus \mathcal{E}_{\text{del}} \bowtie \mathcal{F}_{\text{del}} \oplus \mathcal{E}_{\text{del}} \bowtie \mathcal{F}_{\text{uo}} \ominus \\ & \mathcal{E}_{\text{del}} \bowtie \mathcal{F}_{\text{ins}} \ominus \mathcal{E}_{\text{del}} \bowtie \mathcal{F}_{\text{un}} \end{aligned} \tag{5.20}$$

$$f_{\text{un}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow \mathcal{E}_{\text{new}} \bowtie \mathcal{F}_{\text{un}} \oplus \mathcal{E}_{\text{un}} \bowtie \mathcal{F}_{\text{new}} \ominus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{un}} \ominus \mathcal{E}_{\text{un}} \bowtie \mathcal{F}_{\text{un}} \tag{5.21}$$

$$\begin{aligned} f_{\text{uo}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow & \mathcal{E}_{\text{new}} \bowtie \mathcal{F}_{\text{uo}} \ominus \mathcal{E}_{\text{ins}} \bowtie \mathcal{F}_{\text{uo}} \ominus \mathcal{E}_{\text{un}} \bowtie \mathcal{F}_{\text{uo}} \oplus \mathcal{E}_{\text{uo}} \bowtie \mathcal{F}_{\text{new}} \ominus \\ & \mathcal{E}_{\text{uo}} \bowtie \mathcal{F}_{\text{ins}} \ominus \mathcal{E}_{\text{uo}} \bowtie \mathcal{F}_{\text{un}} \oplus \mathcal{E}_{\text{uo}} \bowtie \mathcal{F}_{\text{del}} \oplus \mathcal{E}_{\text{uo}} \bowtie \mathcal{F}_{\text{uo}} \end{aligned} \tag{5.22}$$

$$f_{\text{ups}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow \mathcal{E}_{\text{new}} \bowtie \mathcal{F}_{\text{ups}} \oplus \mathcal{E}_{\text{ups}} \bowtie \mathcal{F}_{\text{new}} \ominus \mathcal{E}_{\text{ups}} \bowtie \mathcal{F}_{\text{ups}} \tag{5.23}$$

$$f_{\text{new}}(\mathcal{E} \bowtie \mathcal{F}) \rightsquigarrow \mathcal{E}_{\text{new}} \bowtie \mathcal{F}_{\text{new}} \tag{5.24}$$

Though the right-hand sides of the above transformation rules look rather complex they can still be efficiently evaluated by ETL tools. It can be assumed that change data is

| | | Join predicate unaffected | | | Join predicate affected | | |
|---|---|---|---|---|---|---|---|
| | $E_{new} \cap E_{old}$ | $E_{ins}$ | $E_{del}$ | $E_{un}$ | $E_{uo}$ | $E_{un}$ | $E_{uo}$ |
| $F_{new} \cap F_{old}$ | | ins | del | un | uo | ins | del |
| $F_{ins}$ | ins | ins | | ins | | ins | |
| $F_{del}$ | del | | del | | del | | del |
| Join predicate unaffected: $F_{un}$ | un | ins | | un | | ins | |
| Join predicate unaffected: $F_{uo}$ | uo | | del | | uo | | del |
| Join predicate affected: $F_{un}$ | ins | ins | | ins | | ins | |
| Join predicate affected: $F_{uo}$ | del | | del | | del | | del |

Figure 2: Matrix Representation n-to-m Joins

considerable smaller in volume than base relations. Furthermore ETL tools are able to process the joins involved in parallel.

For the case of many-to-many joins we again provide a matrix representation shown in Figure 2. Note that a distinction of update operation is necessary here. Updates that affect columns used in the join predicate have to be separated from updates that do not affect those columns. The reason is that the former may cause tuples to find new join partners or loose current ones while the latter may not. Consequently, updates that affect the join predicate give rise to insertions and deletions while updates that do not affect the join predicate can often be propagated as updates. For space limitations we omit the transformation rules for many-to-many joins.

### 5.5 Example

We exemplify the derivation process of incremental load jobs by means of an example. Assume that there are two operational data sources $A$ and $B$ that store customer information. Further assume that there is a dataset $C$ that contains information about countries and regions together with a unique identifier, say ISO country codes. The country codes for each customer are available in relations $A$ and $B$. All these information shall be integrated into the customer dimension of the data warehouse. That is, customer data extracted from relations $A$ and $B$ is transformed to conform to an integrated schema according to the expressions $a$ and $b$. This step may involve converting date formats or standardizing address information, for instance. Furthermore a unique source identifier is appended to maintain data lineage. The resulting data is then merged (union) and joined to the country relation $C$. For the sake of the example, assume that customer data from $A$ is filtered by means of some predicate $p$ after being extracted, for example, to discard inactive customers. The ETL expression $\mathcal{E}$ to describe the initial load is given below.

$$\mathcal{E} = [\pi_a (A) \oplus \pi_b (\sigma_p (B))] \bowtie C$$

CDP expressions are derived from $\mathcal{E}$ step-by-step, by applying suitable transformation rules. The process is exemplified for the case of insertions below.

$$
\begin{aligned}
& f_{\text{ins}}(\mathcal{E}) = \\
& f_{\text{ins}}([\pi_a(A) \oplus \pi_b(\sigma_p(B))] \ltimes C) \stackrel{5.19}{=} \\
& f_{\text{ins}}([\pi_a(A) \oplus \pi_b(\sigma_p(B))]) \ltimes f_{\text{new}}(C) \stackrel{5.13}{=} \\
& [f_{\text{ins}}(\pi_a(A)) \oplus f_{\text{ins}}(\pi_b(\sigma_p(B)))] \ltimes f_{\text{new}}(C) \stackrel{5.1}{=} \\
& [\pi_a(f_{\text{ins}}(A)) \oplus \pi_b(f_{\text{ins}}(\sigma_p(B)))] \ltimes f_{\text{new}}(C) \stackrel{5.7}{=} \\
& [\pi_a(f_{\text{ins}}(A)) \oplus \pi_b(\sigma_p(f_{\text{ins}}(B)) \oplus (\sigma_p(f_{\text{un}}(B)) \,\overline{\ltimes}\, \sigma_p(f_{\text{uo}}(B))))] \ltimes f_{\text{new}}(C) = \\
& [\pi_a(A_{\text{ins}}) \oplus \pi_b(\sigma_p(B_{\text{ins}}) \oplus (\sigma_p(B_{\text{un}}) \,\overline{\ltimes}\, \sigma_p(B_{\text{uo}})))] \ltimes C_{\text{new}}
\end{aligned}
$$

CDP expressions can be translated into an executable ETL job as shown in [DHW+08]. The transformation rules are designed to "push" the function $f_{\text{ins}}$, $f_{\text{del}}$, $f_{\text{un}}$, or $f_{\text{uo}}$ into the ETL expression. Eventually these functions are directly applied to source relations and thus denote the output of CDC techniques. At this point no further transformation rule is applicable and the derivation process terminates. The resulting CDP expression shows that the propagation of insertions is possible if $A$'s CDC technique is capable of capturing insertions and $B$'s CDC technique is capable of capturing insertions and updates in their initial state and in their current state. In case the available CDC techniques do not meet these requirements the propagation of insertions is impractical. However, it is possible to propagate upserts in this situation as the evaluation of $f_{\text{ups}}(\mathcal{E})$ would show.

### 5.6 Experimental Results

We provide some experimental results to demonstrate the advantage of incremental loading over full reloading. We stress that the measurement is exemplary. Nevertheless, the results clearly suggest performance benefits. Our experiment is based on the sample scenario described in the previous section. We chose the cardinality of the customer relation and the country relation to be 600,000 tuples and 300 tuples, respectively. During the experiment, we stepwise increased the number of changes to the source relations. At each step, equal numbers of tuples were inserted, deleted, and updated. We employed ETL jobs for initial loading, full reloading, and incremental loading. The jobs for initial loading and full reloading differ in the sense that the former expects the target relation to be empty while the latter does not. Instead, it performs lookups to decide whether tuples need to be inserted, updated, or remain unchanged. The incremental load job computes three separate datasets, i.e. tuples to be inserted and tuples to be updated in both, their initial state and their current state. Deletions are not propagated since historical data is kept in the data warehouse.

We measured the time to compute change data for data warehouse refreshment, i.e. we focused on CDP and excluded CDC and CDA from our considerations. The cost of CDC

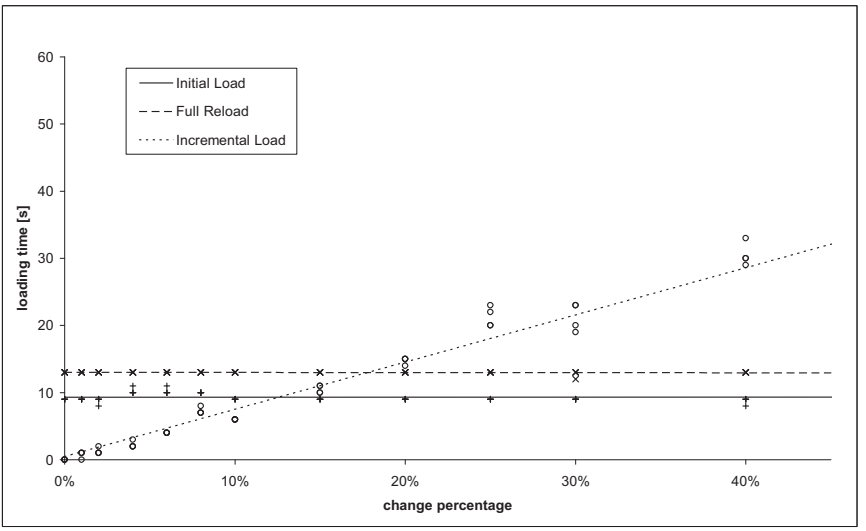



Figure 3: Change Data Propagation Time Comparisons

depends on the CDC technique used and the architecture of the CDC system. However, none of the CDC techniques described in Section 3.2 has a stronger impact on the data source than a full extraction required for full reloading. The cost of CDA again depends on the loading facility used. However, the CDA cost is the same for both, full reloading and incremental loading.

The results of the experiment are provided in Figure 3. Expectedly, the time for the initial load and the full reload are constant, i.e. the number of source data changes does not impact the runtime of these jobs. The full reload is considerably slower than the initial load. The reason is the overhead incurred by the frequent lookups. Incremental loading clearly outperforms full reloading unless the source relations happen to change dramatically.

## 6 Conclusion

In this paper we addressed the issue of data warehouse refreshment. We argued that incremental loading is more efficient than full reloading unless the operational data sources happen to change dramatically. Thus, incremental loading is generally preferable. However, the development of ETL jobs for incremental loading is ill-supported by existing ETL tools. In fact, separate ETL jobs for initial loading and incremental loading have to be created by ETL programmers so far. Since incremental load jobs are considerably more complex their development is more costly and error-prone.

To overcome this obstacle we proposed an approach to derive incremental load jobs from given initial load jobs based on equational reasoning. We therefore reviewed existing Change Data Capture (CDC) techniques that provide the input for incremental loading.

We further reviewed existing loading facilities to update the data warehouse incrementally, i.e. to perform the final step of incremental loading referred to as Change Data Application (CDA). Based on our analysis we introduced a formal model for *change data* to characterize both, the output of CDC techniques and the input of CDA techniques. Depending on the technical realization, CDC techniques suffer from limitations and are unable to detect certain changes. For this reason we introduced a notion of *partial* change data.

Our main contribution is a set of equivalence preserving transformation rules that allow for deriving incremental load jobs from initial load jobs. We emphasize that our approach works nicely in the presence of *partial* change data. The derived expressions immediately reveal the impact of partial input change data on the overall change propagation. That is, the interdependencies between tolerable CDC limitations and satisfiable CDA requirements become apparent. We are not aware of any other change propagation approach that considers limited knowledge of change data. Thus, we are confident that our work contributes to the improvement of ETL development tools.

Future work will focus on advanced transformation operators such as aggregation, outer joins, and data restructuring such as pivoting. We further plan to investigate the usage of the staging area, i.e. allow ETL jobs to persist data that serves as additional input for the subsequent runs. By utilizing the staging area, CDC limitations can be compensated to some extent, i.e. partial change data can be complemented while being propagated. Moreover, we expect performance improvements form persisting intermediary results.

## References


[AASY97] Divyakant Agrawal, Amr El Abbadi, Ambuj K. Singh, and Tolga Yurek. Efficient View Maintenance at Data Warehouses. In *SIGMOD Conference*, pages 417–427, 1997.

[AN08] Alexander Albrecht and Felix Naumann. Managing ETL Processes. In *NTII*, pages 12–15, 2008.

[BLT86] José A. Blakeley, Per-Åke Larson, and Frank Wm. Tompa. Efficiently Updating Materialized Views. In *SIGMOD Conference*, pages 61–71, 1986.

[BT98] Michele Bokun and Carmen Taglienti. Incremental Data Warehouse Updates. *DM Review Magazine*, May 1998.

[CW91] Stefano Ceri and Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB*, pages 577–589, 1991.

[DHW+08] Stefan Dessloch, Mauricio A. Hernández, Ryan Wisnesky, Ahmed Radwan, and Jindan Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316, 2008.

[DS] IBM WebSphere DataStage. `http://www-306.ibm.com/software/data/integration/datastage/`.

[DWE] IBM DB2 Data Warehouse Enterprise Edition. `www.ibm.com/software/data/db2/dwe/`.

[GL95] Timothy Griffin and Leonid Libkin. Incremental Maintenance of Views with Duplicates. In *SIGMOD Conference*, pages 328–339, 1995.

[GMS93] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *SIGMOD Conference*, pages 157–166, 1993.

[IPC] Informatica PowerCenter. `http://www.informatica.com/products_services/powercenter/`.

[JD08] Thomas Jörg and Stefan Deßloch. Towards Generating ETL Processes for Incremental Loading. In *IDEAS*, pages 101–110, 2008.

[KC04] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, Inc., 2004.

[KR02] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[LGM96] Wilburt Labio and Hector Garcia-Molina. Efficient Snapshot Differential Algorithms for Data Warehousing. In *VLDB*, pages 63–74, 1996.

[LN07] Ulf Leser and Felix Naumann. *Informationsintegration*. dpunkt.verlag, 2007.

[OWB] Oracle Warehouse Builder. `http://www.oracle.com/technology/products/warehouse/index.html`.

[QGMW96] Dallan Quass, Ashish Gupta, Inderpal Singh Mumick, and Jennifer Widom. Making Views Self-Maintainable for Data Warehousing. In *PDIS*, pages 158–169, 1996.

[QW91] Xiaolei Qian and Gio Wiederhold. Incremental Recomputation of Active Relational Expressions. *IEEE Trans. Knowl. Data Eng.*, 3(3):337–341, 1991.

[Sim05] Alkis Simitsis. Mapping conceptual to logical models for ETL processes. In *DOLAP*, pages 67–76, 2005.

[SVS05] Alkis Simitsis, Panos Vassiliadis, and Timos K. Sellis. Optimizing ETL Processes in Data Warehouses. In *ICDE*, pages 564–575, 2005.

[SVTS05] Alkis Simitsis, Panos Vassiliadis, Manolis Terrovitis, and Spiros Skiadopoulos. Graph-Based Modeling of ETL Activities with Multi-level Transformations and Updates. In *DaWaK*, pages 43–52, 2005.

[VSS02] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. Conceptual modeling for ETL processes. In *DOLAP*, pages 14–21, 2002.

[Yu06] Tsae-Feng Yu. A Materialized View-based Approach to Integrating ETL Process and Data Warehouse Applications. In *IKE*, pages 257–263, 2006.

[ZGMHW95] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View Maintenance in a Warehousing Environment. In *SIGMOD Conference*, pages 316–327, 1995.

# Selektives Laden und Entladen von Prädikatsextensionen beim Constraint-basierten Datenbank-Caching

Joachim Klein, Susanne Braun, Gustavo Machado

AG Datenbanken und Informationssysteme, Fachbereich Informatik
Technische Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
{jklein, s_braun, machado}@informatik.uni-kl.de

**Zusammenfassung:** Um die Antwortzeit von Anfragen an Datenbanksysteme zu verringern und die Skalierbarkeit zu erhöhen, werden beim Datenbank-Caching Teilmengen von Daten in der Nähe von Anwendungen vorgehalten. Im Gegensatz zu anderen Ansätzen wird hierbei eine deklarative Anfragebearbeitung durch das Cache-System angestrebt, welche die Auswertung einzelner Prädikate unterstützt, die in häufig auszuwertenden Anfragen auftreten. Beim Constraint-basierten Datenbank-Caching werden hierzu Bedingungen definiert, die eine korrekte Anfrageauswertung garantieren und zudem leicht überprüfbar sind. Dabei beschreiben die Constraints einen Abhängigkeitsgraphen, der das Laden und Entladen von Cache-Inhalten beeinflusst. Um dennoch eine bestmögliche Anfragebearbeitung zu gewährleisten, ist es wichtig, leistungsstarke Methoden zu entwickeln, die ein selektives Laden und Entladen zu verwaltender Einheiten (so genannter Cache Units) ermöglicht. Dieser Aufsatz beschreibt die bisherigen Ansätze und evaluiert erstmals explizit deren Performance. Die neu eingeführten Begriffe Cache-Unit und Cache-Unit-Differenz helfen dabei, die Größenverhältnisse der zu verwaltenden Einheiten zu beschreiben. Darüber hinaus werden neue Umsetzungen vorgestellt, die ein effizienteres Laden und Entladen als bisher ermöglichen und die Adaptivität des Gesamtsystems steigern.

## 1 Motivation

Heutzutage gibt es kaum noch Anwendungen, die nicht auf entfernt gespeicherte Daten angewiesen sind. Immer mehr Anwendungen und Dienste werden bereits direkt über das Internet abgerufen. So kann man heute schon auf komplette Office-Anwendungen im Web (so genannte Web-Anwendungen) zugreifen. Sie alle sind auf Daten aus leistungsstarken Datenspeichern angewiesen. Im Zuge dieser Entwicklung gewinnen Datenbanken, zu denen vor allem große relationale Datenbanksysteme gehören, zunehmend an Bedeutung. Immer mehr Daten, von nunmehr weltweit agierenden Benutzern, werden gespeichert und verarbeitet. Zusätzlich werden die Systeme inzwischen immer mehr durch Analysen (z. B. durch Data-Mining-Techniken) belastet.

Um die Zugriffszeit auf weit entfernte Datenbanken (oder auch ganze Datenbank-Cluster) zu verringern, versucht das Datenbank-Caching, ähnlich wie das Web-Caching, häufig zugegriffene Daten in der Nähe der Anwendung vorzuhalten. Gleichzeitig entlas-

ten die Cache-Instanzen so den zentralen Datenbank-Server (Backend). Die Existenz eines Cache-Systems ist hierbei für Anwendungen stets transparent, wodurch sie wie gewöhnlich auf die Daten, wie sie im Backend-Schema definiert sind, zugreifen können.

Die einfachste Art, einen Datenbank-Cache zu realisieren, besteht darin, komplette Tabellen repliziert vorzuhalten, was auch als Full-Table Caching [Ora08] bezeichnet wird. Da solche Ansätze keine dynamische Anpassung des Caches-Inhaltes an die tatsächliche Anfragelast zulassen, wurden mehrere weiterführende Konzepte entwickelt. Viele dieser Ansätze benutzen materialisierte Sichten (bzw. Anpassungen davon) [APTP03, BDD+98, GL01, LMSS95, LGZ04, LGGZ04]. Da alle diese Ansätze auf die Auswertung der durch materialisierte Sichten definierten Tabellen beschränkt sind, versuchen Konkurrenzansätze PSJ-Anfragen[1] über mehrere Cache-Tabellen zu unterstützen [ABK+03, The02]. Zu diesen Ansätzen gehört auch das Constraint-basierte Datenbank-Caching (CbDBC). Auch in der Industrie haben Datenbankhersteller die Ansätze übernommen, um ihre Produkte um einen entsprechenden Datenbank-Cache zu erweitern [ABK+03, LGZ04, The02].

Der Grundgedanke des CbDBC besteht darin, vollständige Extensionen (also Satzmengen) einzelner Prädikate vorzuhalten, die hierdurch für die Beantwortung unterschiedlicher Anfragen verwendbar sind. Dabei wird die Vollständigkeit dieser Prädikatsextensionen durch einfache Bedingungen (Constraints) sichergestellt (vgl. Abschnitt 2).

Da diese Constraints zu jeder Zeit vom Cache-Inhalt erfüllt sein müssen, hat die Cache-Wartung mengenorientiert unter Beachtung der definierten Constraints zu erfolgen. Das Laden von Satzmengen in den Cache und das Entladen dieser ist beim CbDBC von besonderer Bedeutung, da sich der Caching-Effekt erhöht, je schneller die Daten aktuell referenzierter Prädikate geladen und veraltete entfernt werden können [BK07]. Die typischerweise hohe Latenz zwischen Backend und Cache erschwert das Laden hierbei zusätzlich. Darüber hinaus entstehen durch die definierten Constraints Abhängigkeitsketten (typischerweise zwischen Tabellen, die häufig Verbundpartner sind), deren Beachtung beim Laden und Entladen erhebliche Komplexität in sich birgt. Vor allem zyklische Abhängigkeiten führen große Probleme ein (vgl. Abschnitt 3.2).

In [BHM06] wurden bereits die konzeptionellen Grundlagen des Ladens sowie eine erste Implementierung vorgestellt. Gezielte Untersuchungen der Performance bezüglich grundlegender Cache-Strukturen (vgl. Abschnitt 4.1) bestätigen die bereits erwarteten Performance-Einbrüche bei längeren Abhängigkeitsketten. Dieser Aufsatz stellt zwei neue Implementierungskonzepte vor, das *indirekte Laden* (im Abschnitt 4.2) und das *vorbereitete Laden* (in Abschnitt 4.3), welche die aufgetretenen Probleme lösen. Dabei werden die Vor- und Nachteile der verschiedenen Konzepte gegenübergestellt und durch Messungen überprüft.

Bezüglich des Entladens wurde bisher vorgeschlagen, den kompletten Cache-Inhalt zu bestimmten Zeitpunkten zu löschen, da ein selektives Löschen in Zyklen zu gesonderten Problemen führt [HB07] (vgl. auch Abschnitt 3.2). Wir stellen in Abschnitt 5 eine neue Methode vor, die es erlaubt, das selektive Entladen performant durchzuführen. Zunächst werden jedoch in Abschnitt 2 die grundlegenden Definitionen des CbDBC (aus [HB07]) zum besseren Verständnis des Aufsatzes wiederholt.

[1] Projektion-Selektion-Join-Anfragen

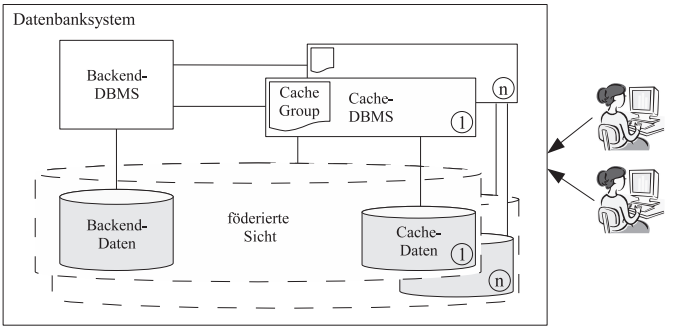



Abbildung 1: Der grundlegende Aufbau eines CbDBC-Systems

## 2 Constraint-basiertes Datenbank-Caching

Das CbDBC hält Satzmengen häufig angefragter Prädikate in Cache-Instanzen vor und beschleunigt so den lesenden Zugriff. Die Daten werden in Cache-Tabellen gespeichert und erfüllen dabei stets die aktuell gültige Menge von Constraints. Das Cache-Verwaltungssystem verwendet die Constraints, um entscheiden zu können, ob eine Anfrage oder Teilprädikate davon beantwortet werden können.

Die grundlegende Struktur des Cache-Systems wird durch Abbildung 1 verdeutlicht. Jede Cache-Instanz bedient sich einer föderierten Sicht, um in Anfragen sowohl auf Cache-Tabellen als auch auf Backend-Tabellen zugreifen zu können. Das komplette Datenbanksystem besteht nun aus einem Backend-DBS und eventuell mehreren Cache-Instanzen. Die Menge an Cache-Tabellen und die Menge aller Constraints, die eine Cache-Instanz verwaltet, werden in einer so genannten *Cache Group* zusammengefasst.

### 2.1 Elemente einer Cache Group

Cache-Tabellen sind logische Teilkopien entsprechender Backend-Tabellen, da sie stets nur eine (häufig benötigte) Teilmenge ihrer Daten enthalten. Dabei entspricht die Schema-Definition einer Cache-Tabelle der ihr zugeordneten Backend-Tabelle bis auf Fremdschlüssel, die nicht übernommen werden. Primärschlüsseldefinitionen und Unique-Constraints bleiben jedoch erhalten. Die zu einer Backend-Tabelle $T$ gehörende Cache-Tabelle bezeichnen wir fortan als $T_C$.

**Constraints.** Die einzigen in einer Cache Group verwendeten Constraints sind *Füllspalten* (filling column, FC) und *referentielle Cache-Constraints* (referential cache constraint, RCC)[2]. Über die Inhalte einer Füllspalte (vgl. Abbildung 2a) entscheidet das Cache-System, ob Werte in den Cache zu laden sind. Ein Ladevorgang wird immer dann ausgeführt, wenn eine Anfrage einen Wert der Füllspalte explizit referenziert, z. B. durch die

[2]Inwiefern ein RCC von den bekannten Konzepten einer Primär-Fremdschlüsselbeziehung abweicht, wird auch in [HB07] ausführlich erläutert.

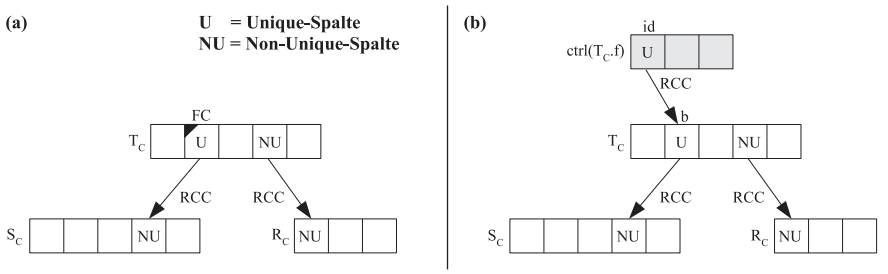



Abbildung 2: Konzeptionelle (a) und interne (b) Darstellung einer Cache Group

Anfrage $\sigma_{id=10}T$. Ist der referenzierte Wert in einer zuvor definierten Kandidatenmenge enthalten, so wird er, sofern noch nicht vorhanden, *wertvollständig* in den Cache geladen.

**Definition 2.1 (Wertvollständigkeit)** *Ein Wert einer Spalte wird als wertvollständig bezeichnet (oder kurz vollständig), wenn alle Sätze der zugehörigen Backend-Tabelle, die den gleichen Spaltenwert enthalten, auf dem Cache verfügbar sind. Somit ist der Wert v vollständig in $T_C.a$ genau dann, wenn alle Sätze $\sigma_{a=v}T$ im Cache verfügbar sind.*

Mit einem RCC lassen sich zwei Spalten aus Cache-Tabellen miteinander verbinden, die den gleichen Wertebereich haben. Er wird als gerichtete Kante zwischen den Spalten dargestellt (vgl. Abbildung 2). Ein RCC verlangt, dass alle Werte, die in der Quellspalte eines RCC enthalten sind, in der Zielspalte wertvollständig vorliegen. Es lässt sich nun bereits erkennen, dass alle über RCCs abhängigen Sätze (nach Beendigung eines Ladevorgangs) im Cache vorliegen müssen.

**Definition 2.2 (Füllspalte)** *Eine Cache-Spalte $T_C.b$, die als Füllspalte deklariert ist, lädt alle Werte $v \in K$, die explizit durch eine Anfrage $\sigma_{b=v}T$ referenziert werden, wertvollständig in den Cache. Hierbei beschreibt K die Menge aller ladbaren Füllspaltenwerte (Kandidatenmenge).*

**Definition 2.3 (RCC)** *Ein referenzieller Cache-Constraint $T_C.b \rightarrow S_C.a$ zwischen einer Quellspalte $T_C.b$ und einer Zielspalte $S_C.a$ ist genau dann erfüllt, wenn alle Werte v aus $T_C.b$ wertvollständig in $S_C.a$ sind.*

Da die Zusicherungen der beiden Constraints hauptsächlich auf der zugrundeliegenden Wertvollständigkeit basieren, lässt sich das Konzept noch weiter vereinfachen. Zunächst bezeichnen wir jede RCC-Quellspalte als *Kontrollspalte*, da sie für alle in ihr enthaltenen Werte die Wertvollständigkeit in der RCC-Zielspalte erzwingt. Um die Wertvollständigkeit einer Füllspalte (etwa $T_C.f$ aus Abbildung 2b) zu garantieren, wird jeweils eine interne (nur auf dem Cache verwaltete) Tabelle hinzugefügt. Diese wird *Kontrolltabelle* $\text{ctrl}(T_C.f)$ genannt und speichert in einer Kontrollspalte $\text{ctrl}(T_C.f).id$ alle Werte, die in $T_C.f$ wertvollständig geladen sind. Ein in die Spalte $\text{ctrl}(T_C.f).id$ eingelagerter Wert wird als *Füllwert* bezeichnet. Ein RCC realisiert die Eigenschaften der Füllspalte und ermöglicht es fortan, alle Funktionalitäten (wie auch das Laden und Entladen) nur noch mit Hilfe von RCCs und Cache-Tabellen zu beschreiben.

# 3 Grundlagen des Ladens und Entladens

Wodurch und wann das Laden von Sätzen in den Cache ausgelöst wird, wurde bereits im vorangegangenen Abschnitt beschrieben. In diesem Abschnitt wenden wir uns nun eingehend den speziellen Eigenschaften zu und den Problemen, die beim Laden und Entladen auftreten. Zunächst schaffen wir einige theoretische Grundlagen, um die Menge der Sätze, die zu laden und zu entladen ist, besser beschreiben zu können. Die Menge an Sätzen, die aufgrund eines Kontrollwertes $v$ in $T_C.a$ auf dem Cache vorliegen müssen, wird als *RCC-Hülle* bezeichnet.

**Definition 3.1 (RCC-Hülle)** *Sei $T_C(a_1,\ldots,a_n)$ eine Cache-Tabelle mit einem ausgehenden RCC $T_C.a_i \rightarrow R_C.b$ und $s = (w_1,\ldots,w_n) \in T_C$ ein Satz mit $w_i = v$. Die Menge aller Sätze, die aufgrund von $s \in T_C$ mit $w_i = v$ im Cache vorliegen müssen, wird als RCC-Hülle von $v$ in $T_C.a_i$ (oder kurz: $RH_{T_C.a_i}(v)$) bezeichnet.*

Für das Laden und Entladen von Sätzen sind die RCC-Hüllen der Füllwerte besonders wichtig, da durch sie die Mengen aller Sätze, die auf dem Cache vorliegen müssen, festgelegt sind. Aus diesem Grund bezeichnen wir die RCC-Hülle eines Füllwertes als *Cache Unit*.

**Definition 3.2 (Cache Unit)** *Sei $T_C.f$ eine Füllspalte. Die RCC-Hülle eines Füllwertes $v$ aus* ctrl$(T_C.f)$*.id bezeichnen wir als Cache Unit von $v$ in $T_C.f$ (oder kurz als $CU_{T_C.f}(v)$).*

Ein Satz, der in den Cache geladen wurde, kann zu mehreren Cache Units gehören. Die Anzahl der Sätze, die während eines Ladevorgangs tatsächlich geladen werden müssen bzw. während des Entladens gelöscht werden können, kann daher sehr viel kleiner sein als die Menge der Sätze in der Cache Unit. Die Menge der zu ladenden bzw. zu verdrängenden Sätze hängt also stark von den bereits geladenen Cache Units ab und wird daher erstmals als *Cache-Unit-Differenz* konkret definiert.

**Definition 3.3 (Cache-Unit-Differenz)** *Sei $CU_{T_C.f}(v)$ eine Cache Unit, die zu laden bzw. zu entladen ist, und $I$ die Vereinigung aller übrigen Cache Units, die unverändert im Cache bleiben. Die Menge $CU_{T_C.f}(v) - I$ aller Sätze, die tatsächlich geladen bzw. verdrängt werden müssen, wird als Cache-Unit-Differenz von $v$ in $T_C.f$ (oder kurz: CU-Diff$_{T_C.f}(v)$) bezeichnet.*[3]

Durch die vorangegangenen Betrachtungen wird deutlich, dass es aufgrund der Struktur einer Cache Group evtl. schwer sein kann, die Sätze zu finden, die zur jeweiligen Cache-Unit-Differenz gehören. Um ein besseres Verständnis für diese Schwierigkeiten zu erlangen, isolieren wir zunächst die Zyklen innerhalb einer Cache Group.

## 3.1 Atomare Zonen

Die Struktur einer Cache Group, bestehend aus Cache-Tabellen und RCCs, lässt sich als gerichteter Graph mit Tabellen als Knoten und RCCs als Kanten betrachten. Zyklische

[3] Auf eine noch formalere Definition wurde bewusst verzichtet, da sie in der Folge nicht benötigt wird.

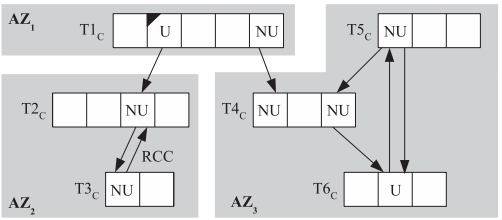



Abbildung 3: Die Aufspaltung einer Cache Group in ihre atomaren Zonen

Abhängigkeiten verursachen in diesem Graph die größten Probleme und werden daher separat in Abschnitt 3.2 betrachtet. Um einen azyklischen Graph zu bilden, werden die Zyklen in *atomaren Zonen (AZ)* zusammengefasst (vgl. auch [BHM06]), die sich isoliert betrachten lassen. Auch Cache-Tabellen, die nicht Teil eines Zyklus sind, stellen atomare Zonen dar. Wir nennen diese *triviale atomare Zonen*. Cache-Tabellen, die zu einem Zyklus gehören, werden in einer so genannten *nicht-trivialen atomaren Zone* zusammengefasst. Abbildung 3 zeigt ein Beispiel, in dem die Aufspaltung einer Cache Group in ihre Zonen verdeutlicht wird. Dabei wird ein RCC, der zwischen zwei Zonen verläuft, als *externer RCC* bezeichnet, ein RCC, der innerhalb einer atomaren Zone verläuft, als *interner RCC*.

Betrachten wir die atomaren Zonen als Knoten und die externen RCCs als Kanten, so entsteht ein gerichteter azyklischer Graph, der es uns ermöglicht, das Laden und Entladen von Werten auf einer höheren Abstraktionsebene zu betrachten, ohne Zyklen zu berücksichtigen.

Wir betrachten erneut Abbildung 3: Bereits in [BHM06] wurde diskutiert, in welcher Reihenfolge die Sätze in die Cache-Tabellen einzufügen sind (top-down oder bottom-up). Dabei wurde festgestellt, dass das Einladen der Daten von unten nach oben den enormen Vorteil bietet, pro atomarer Zone eine Fülltransaktion bilden zu können, ohne RCCs zu verletzen. Dadurch ist es z. B. möglich, die neu geladenen Sätze der RCCs in $AZ_3$ bereits vor der Beendigung des kompletten Ladevorgangs für anstehende Anfragen zu nutzen. Würden die Sätze von oben nach unten eingefüllt, dürfte das Commit der Ladetransaktion erst nach dem Laden aller atomaren Zonen ausgeführt werden, um die Konsistenz der Cache-Inhalte und eine korrekte Anfrageauswertung zu gewährleisten.

Das Entladen von Cache Units bietet die gleichen Vorteile, wenn die atomaren Zonen von oben nach unten entladen werden und dabei die betroffenen Sätze jeweils transaktionsgeschützt entfernt werden. Deshalb ist zuerst der Füllwert aus der Kontrolltabelle zu löschen. Danach sind die Daten aus $AZ_1$, dann aus $AZ_2$ und zuletzt aus $AZ_3$ zu löschen. Auch hier können die Transaktionen nach Bearbeitung einer atomaren Zone abgeschlossen werden.

Um jeweils die richtige Ausführungsreihenfolge zu finden, wird ein einfacher Algorithmus [CLRS82] eingesetzt, der die Zonen topologisch sortiert und somit eine totale Ordnung bestimmt.

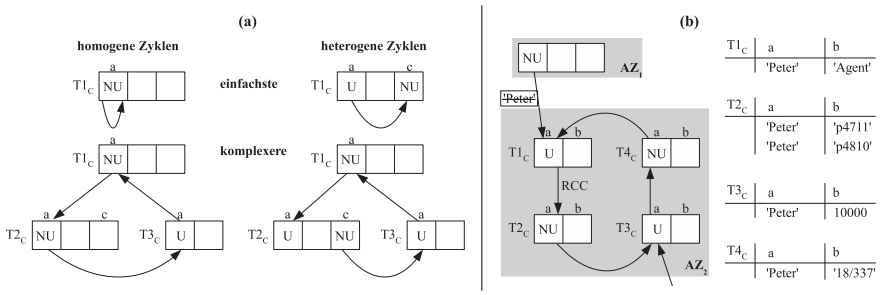



Abbildung 4: (a) Beispiele für homogene und heterogene Zyklen, (b) Wertebeispiel

### 3.2 Probleme in Zyklen

Nachdem wir bisher durch die Einführung der atomaren Zonen von Zyklen abstrahiert haben, werden wir uns nun mit den Problemen beschäftigen, die innerhalb nicht-trivialer Zonen auftreten. Die Zyklen werden nochmals unterschieden in *homogene Zyklen* und *heterogene Zyklen*. Abbildung 4a zeigt einige Beispiele von homogenen und heterogenen Zyklen. Wir nennen einen Zyklus aus RCCs homogen, wenn nur eine Spalte pro Tabelle Teil des Zyklus ist. Dies ist z. B. im Zyklus $T1_C.a \rightarrow T2_C.a \rightarrow T3_C.a \rightarrow T1_C.a$ der Fall. Im Gegensatz dazu nennen wir einen Zyklus heterogen, wenn mehr als eine Spalte pro Tabelle beteiligt ist [ABK+03].

Während des Ladens treten die Hauptprobleme in heterogenen Zyklen auf, weil dabei ein *rekursives Laden* ausgelöst werden kann. Bei einem Zyklusdurchlauf können nämlich Sätze eingelagert werden, bei denen Werte aus mehreren Spalten die Fortsetzung des Ladevorgangs bestimmen und somit erneute Zyklusdurchläufe erzwingen. Dieses Problem versucht Beispiel 3.1 zu verdeutlichen; es ist ausführlich in [HB07] beschrieben. Das gleiche Problem entsteht auch beim Entladen, wobei zu große Cache-Unit-Differenzen entstehen können, die dann zu entladen wären.

Homogene Zyklen sind hingegen gutartig. In ihnen stoppt der Ladevorgang stets spätestens nach einem Zyklusdurchlauf, weil in den bestimmenden Spalten beim Einfügen keine neuen Werte auftreten. Beim Entladen in homogenen Zyklen ist die Situation jedoch ungleich komplizierter. Die zyklische Abhängigkeit unter den Werten muss erkannt und aufgelöst werden. Wir erläutern dieses Problem ausführlich in Beispiel 3.2.

*Beispiel 3.1.* [Rekursives Laden/Entladen]
Wir betrachten den einfachsten heterogenen Zyklus aus Abbildung 4. Sobald ein Wert $w_1$ in Spalte $T1_C.a$ geladen wird, müssen auch alle abhängigen Sätze mit den Werten $T1_C.c = w_1$ in die Tabelle $T1_C$ geladen werden. Dabei können wiederum neue Werte $w_2, ..., w_k$, die zuvor nicht in der Tabelle vorhanden waren, in der Spalte $T1_C.a$ auftreten. Das Laden setzt sich rekursiv fort, bis keine neuen Werte mehr auftauchen.

Im schlimmsten Fall werden alle Daten des Backends in die Cache-Tabellen geladen. Um Situationen zu vermeiden, in denen ein rekursives Laden bzw. Entladen möglich ist, werden heterogene Zyklen beim Design einer Cache Group ausgeschlossen. In homogenen

Zyklen stoppt der Ladeprozess nach dem ersten Zyklusdurchlauf, da keine neuen Kontrollwerte mehr auftauchen können. Beim Entladen ist es jedoch schwer, die Abhängigkeiten innerhalb des Zyklus richtig zu erkennen, wie das folgende Beispiel belegt.

*Beispiel 3.2.* [Abhängigkeiten in homogenen Zyklen]
Abbildung 4b zeigt einen homogenen Zyklus, in dem der Wert *'Peter'* bereits aus $AZ_1$ gelöscht wurde. Versuchen wir nun herauszufinden, ob der Wert *'Peter'* auch innerhalb von $AZ_2$ gelöscht werden kann, so muss zunächst die zyklische Abhängigkeit $T1_C.a \rightarrow T2_C.a \rightarrow T3_C.a \rightarrow T4_C.a \rightarrow T1_C.a$ innerhalb der atomaren Zone erkannt werden. Der Wert *'Peter'* kann aber auch von externen RCCs abhängig sein, wie es z. B. durch Tabelle $T3_C$ möglich wäre. Referenziert der externe RCC den Wert, so kann *'Peter'* aus keiner Tabelle entfernt werden. Tut er dies nicht, so sind alle in Abbildung 4b gezeigten Sätze löschbar. Darüber hinaus ist stets zu beachten, dass die Abhängigkeitskette innerhalb eines Zyklus unterbrochen sein kann. Dies ist der Fall, wenn zu einem Kontrollwert in der Zieltabelle des RCC kein Satz mit dem entsprechenden Wert existiert. Nehmen wir an, dass in Tabelle $T3_C$ oder $T4_C$ kein Satz mit dem Wert *'Peter'* existiert, so können zumindest die entsprechenden Sätze der Tabellen $T1_C$ und $T2_C$ gelöscht werden.

Aufgrund dieser Probleme wurde in [HB07] als vorläufige Lösung vorgeschlagen, den kompletten Inhalt des Cache zu bestimmten Zeitpunkten zu löschen. Selbst wenn man einen günstigen Zeitpunkt für einen solchen Leerstart abwartet, so ist der zu erwartende Aufwand des Nachladens, bis der Cache wieder gefüllt ist, beträchtlich. Ein *selektives Entladen* einzelner Cache Units, wie es in Abschnitt 5 beschrieben wird, vermeidet das Löschen des gesamten Inhalts.

### 3.3 Messungen

Die in den folgenden Abschnitten diskutierten Messungen wurden mit dem Ziel durchgeführt, den Aufwand für das Laden und Entladen grundlegender Cache-Group-Strukturen abschätzen zu können. In diesen Experimenten wurde bewusst nicht die Performance des Gesamtsystems vermessen, damit gezielte Aussagen über die hier betrachteten Hauptfunktionen möglich sind. Darüber hinaus wurde zur Vereinfachung die Latenz zwischen Backend und Cache nicht variiert. In allen Messungen wird die Anzahl der zu ladenden bzw. zu löschenden Sätze schrittweise erhöht. Die dabei angegebene Anzahl der zu ladenden Sätze entspricht der Größe der Cache-Unit-Differenz des jeweils geladenen oder entladenen Füllwertes. Dabei wurden die Sätze möglichst gleichmäßig über die Menge der Cache-Tabellen in der Cache-Group-Struktur verteilt.

**Wichtige Cache-Group-Strukturen.** Wie die vorangegangenen Ausführungen verdeutlichen, sind vor allem unterschiedlich lange RCC-Ketten und homogene Zyklen für die Messungen interessant. Wir haben dieser Menge grundlegender Strukturen noch Bäume hinzugefügt, bei denen jede innere Cache-Tabelle zwei ausgehende RCCs aufweist. In Abbildung 5 sind einige Beispiele dieser Strukturen aufgeführt, wobei die jeweilige Kontrolltabelle mit angegeben ist. Die einfachste Struktur, die vermessen wurde, bestand aus

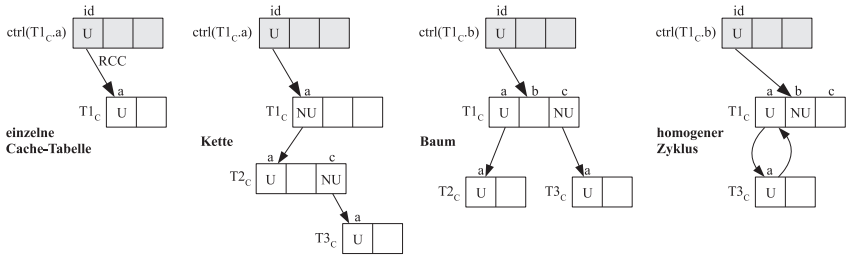



Abbildung 5: Beispiele wichtiger Cache-Group-Strukturen

einer einzelnen Cache-Tabelle mit einer Füllspalte, sodass eine Kette mit einem RCC (zwischen Kontrolltabelle und Cache-Tabelle) entstand.

**Datengenerator.** Ein eigens entwickelter Datengenerator analysiert vor jeder Messung die ihm übergebene Cache Group und versucht, eine Cache-Unit-Differenz bestimmter Größe (z. B. mit 2000 Sätzen) zu generieren, die er auf alle Cache-Tabellen gleich verteilt[4]. Dies bedeutet für die Vermessung einer RCC-Kette mit drei Cache-Tabellen, dass im Beispiel 666 Sätze für die erste Cache-Tabelle, 666 für die zweite und 667 für die dritte Tabelle generiert werden. Die so generierten Sätze werden im Backend hinterlegt.

Alle verwendeten Tabellen besitzen sieben vollbesetzte Spalten. Sind diese Teil einer RCC-Definition, wurde als Datentyp `INTEGER` definiert, für alle anderen Spalten wurde der Datentyp `VARCHAR(300)` verwendet.

**Messaufbau.** Als Cache-System wurde der Prototyp des ACCache-Projekts [Mer05, BHM06] eingesetzt. Die Funktionalität des Systems wurde durch die in Abschnitt 4 und 5 beschriebenen Methoden entsprechend erweitert.

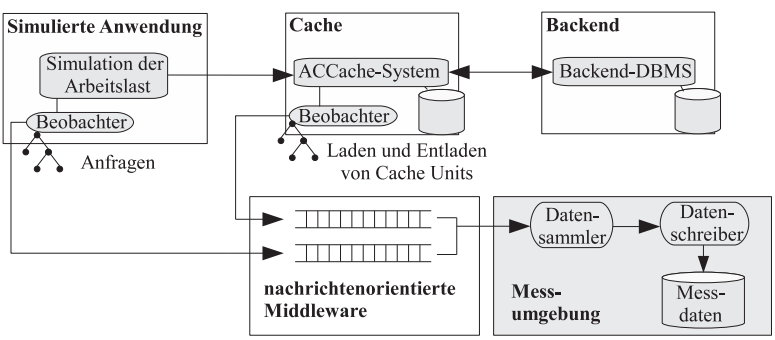



Abbildung 6: Aufbau der Messungen

[4]Es gibt Cache-Group-Strukturen, bei denen eine Gleichverteilung nicht gewährleistet werden kann. Diese leisten jedoch für eine Aufwandsabschätzung keinen gesonderten Beitrag.

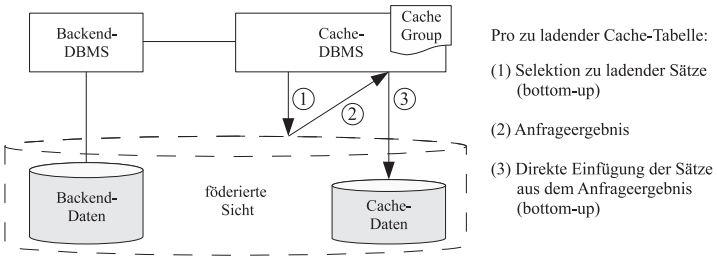



Abbildung 7: Ablaufdiagramm für das direkte Laden

Die verschiedenen Messungen wurden mit einer von uns selbst entwickelten Messumgebung durchgeführt, wobei jeder Messlauf sechsmal ausgeführt wurde. Vor jedem Durchlauf wurden die Daten der Cache Units neu generiert, sodass die Anzahl der Sätze in der Cache-Unit-Differenz zwar gleich blieb, die Daten sich aber jedes mal änderten. Alle Anwendungen wurden durch die Messumgebung jeweils gestoppt und wieder neu gestartet. Die in den folgenden Abschnitten gezeigten Grafiken zeigen jeweils den Mittelwert der Messergebnisse aus den sechs zusammengehörigen Durchläufen.

Die Messumgebung beschreibt die an einer Messung teilnehmenden Anwendungen mittels *Arbeitsknoten*. In allen Messungen wurden drei Arbeitsknoten verwendet: die simulierte Client-Anwendung, die durch entsprechende Anfragen das Laden der Cache Units auslöst, das ACCache-System und das Backend-DBS (vgl. Abbildung 6).

# 4 Laden von Cache Units

## 4.1 Direktes Laden

Wie bereits in Abschnitt 3.1 erwähnt, betrachten wir zuerst das in [BHM06] vorgestellte Konzept, wobei die Sätze über die atomaren Zonen von unten nach oben, *direkt* in den Cache eingefügt werden. Wir bezeichnen diese Methode daher als *direktes Laden*. Abbildung 7 zeigt den Ablauf des Ladevorgangs.

Für jede Cache-Tabelle wird vom Cache-System ein `INSERT`-Statement an die Datenbank gesendet, welches die zu ladenden Sätze auswählt und direkt in die Tabelle einfügt. Das Cache-System greift dabei über die föderierte Sicht auf die Datenbank zu, um die ausgewählten Sätze der Cache Unit mit den bereits geladenen vergleichen zu können (vgl. auch [BHM06]).

**Messergebnisse.** Abbildung 8 zeigt die Messergebnisse für die in Abschnitt 3.3 besprochenen Cache-Group-Strukturen. Um die übrigen Messungen besser einordnen zu können, betrachten wir zunächst das Laden in die einzelne Cache-Tabelle. Die verbleibenden Messungen lassen sich somit relativ zu dieser bewerten. Die in den Messungen gewählte maximale Anzahl von 3000 zu ladenden Sätzen ist recht gering, sodass es keinen erhöhten Aufwand darstellt, diese auszuwählen und einzufügen. Das auf der Cache-Instanz verwen-

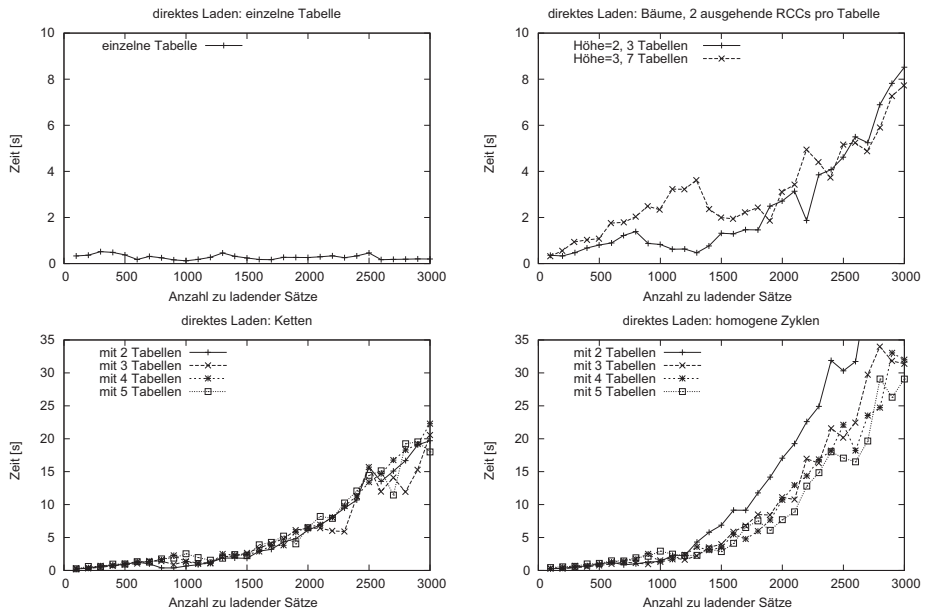



Abbildung 8: Ergebnisse des direkten Ladens: einzelne Knoten, Ketten, Bäume, homogene Zyklen

dete Datenbanksystem erfüllt die Aufgabe stets in einer Geschwindigkeit, die unter 500 ms liegt.

Bereits bei der Auswertung einer Kette mit nur zwei Cache-Tabellen ergibt sich ein inakzeptabel erhöhter Aufwand von bis zu 23 Sekunden, der auch bei der Vermessung längerer Ketten stabil bleibt. Dieses Verhalten gründet sich auf zwei gegenläufige Effekte beim Auswerten der für das Laden erforderlichen Anfragen: Zum einen erhöht sich der Aufwand durch die Hinzunahme einer weiteren Cache-Tabelle, da für das Laden in die unterste Tabelle eine zusätzliche Anfrage benötigt wird, in der ein Verbund mit allen darüber liegenden Tabellen vorzunehmen ist. Zum anderen verringert sich die Anzahl der jeweils zu selektierenden Sätze pro Tabelle, da die Sätze der Cache Unit auf alle Tabellen gleichmäßig verteilt wurden. Hierdurch sinkt der Aufwand des jeweils auszuführenden Verbunds.

Innerhalb von Zyklen ist der Aufwand sogar so hoch, dass teilweise für das Laden einer Cache Unit bis zu 55 Sekunden notwendig waren. Da in Bäumen die Anzahl der notwendigen Verbunde von ihrer Höhe abhängt, ist der Aufwand hier deutlich geringer, jedoch immer noch sehr hoch (bis zu 9 Sekunden).

Für alle Strukturen ist klar erkennbar, dass der Aufwand stark ansteigt, sobald die Menge der zu ladenden Sätze signifikant hoch ist ($> 1000$ Sätze). Bei kleinen Mengen ($< 1000$ Sätze) liegt der Aufwand hingegen meist unter einer Sekunde.

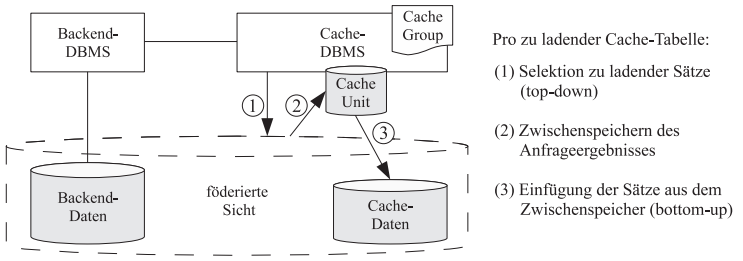



Abbildung 9: Ablauf des indirekten Ladens

## 4.2 Indirektes Laden

Der Ansatz des indirekten Ladens versucht den hohen Selektionsaufwand des direkten Ladens zu vermeiden, indem die Sätze einer Cache Unit zuvor in so genannte *Staging-Tabellen* geladen werden. Dies ermöglicht es, ausgehend vom Füllwert, die Sätze einer Cache Unit von oben nach unten (top-down) zu bestimmen. Dabei werden neu geladene Kontrollwerte in so genannte *Kontrollwerttabellen* geschrieben, anhand derer die zu ladenden Sätze nachfolgender Tabellen bestimmt werden. Sobald eine Cache Unit komplett zwischengespeichert wurde, kann sie wie bisher von unten nach oben geladen werden, wodurch keine RCCs verletzt werden. Abbildung 9 zeigt den schematischen Ablauf des Ladens.

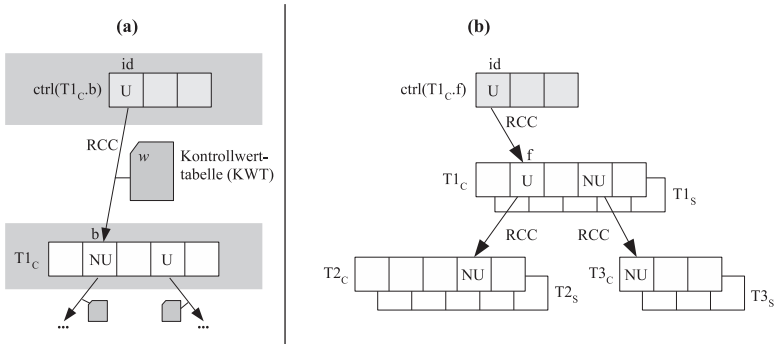



Abbildung 10: Die neuen Elemente: Kontrollwerttabellen (a), Staging-Tabellen (b)

**Zusammenstellen der Cache Unit.** Für jede Cache-Tabelle wird eine Staging-Tabelle mit identischen Spaltendefinitionen angelegt. Die zu einer Cache-Tabelle $T1_C$ gehörende Staging-Tabelle bezeichnen wir mit $T1_S$. Um alle Sätze einer Cache Unit aufzusammeln, wird zunächst der zu ladende Füllwert (z. B. *w*, vgl. Abbildung 10a) in die Kontrollwerttabelle (KWT) des ausgehenden RCCs eingefügt. Jeder RCC verfügt über eine solche Tabelle, um geladene oder gelöschte Kontrollwerte weiterzureichen. Das Zusammenstellen der Daten kann nun durch einen einfachen rekursiven Algorithmus realisiert werden,

der für jede Tabelle, die noch zu verarbeitende KWTs eingehender RCCs aufweist, fehlende Sätze anfragt und neu geladene Kontrollwerte weiterreicht. Besitzt keine KWT mehr zu verarbeitende Kontrollwerte, ist das Zusammenstellen der Cache Unit beendet. Diese Methode, ist unabhängig von der Struktur der Cache Group und würde auch das Laden in heterogenen Zyklen ermöglichen.

**Vergleich: Direktes/indirektes Laden.** In Abbildung 11 sind die Ergebnisse des Vergleichs zwischen direktem und indirektem Laden aufgeführt. Hierzu wurden genau die Cache-Group-Strukturen vermessen, bei denen die Performance-Einbrüche am deutlichsten waren, da das indirekte Laden primär dazu entwickelt wurde, diese Problemsituationen zu bewältigen. Das Ergebnis ist überdeutlich: In beiden Fällen bleibt der Aufwand für das indirekte Laden unter 3 Sekunden. Meistens wird sogar nur eine Sekunde benötigt.

Dieses Ergebnis kann eventuell noch verbessert werden. Das indirekte und auch das direkte Laden werden vom Cache allein durchgeführt. In beiden Fällen sind daher zum Zusammenstellen der Cache Unit mehrere Anfragen notwendig, die auch Backend-Tabellen enthalten. Ist die Latenz zwischen Backend und Cache hoch, wovon grundsätzlich ausgegangen wird, muss diese somit auch mehrfach in Kauf genommen werden. Das nachfolgend vorgestellte Konzept des *vorbereiteten Ladens*, versucht diesen Nachteil zu vermeiden.

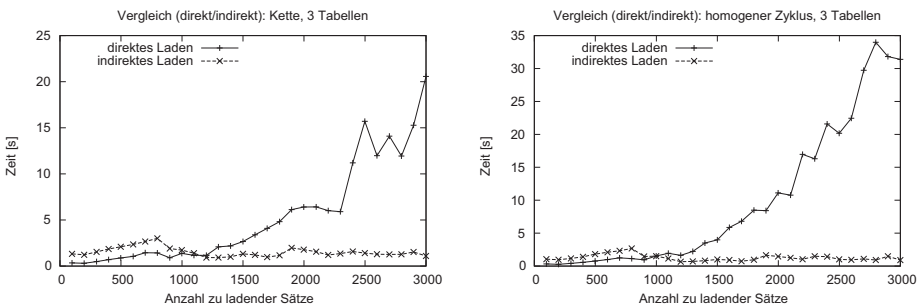



Abbildung 11: Vergleich des direkten und indirekten Ladens

## 4.3 Vorbereitetes Laden

In dieser Variante übernimmt das Backend-DBMS die Zusammenstellung der Cache Unit (vgl. Abbildung 12). Das Cache-DBMS fordert die Cache Unit an, indem es den zu ladenden Füllwert an das Backend-System übergibt. Das Auswählen der Sätze erfolgt dabei wie beim indirekten Laden. Damit dies möglich ist, muss das Backend-DBMS die Definition der Cache Group kennen. Es wird also durch die Selektion der Daten und durch die Wartung zusätzlicher Metadaten belastet. Da es jedoch für andere Funktionalitäten (z. B. die Synchronisation) sowieso notwendig ist, Metadaten der Cache-Instanzen auf dem Backend zu verwalten, wird der Zusatzaufwand für das Verwalten der Metainformationen relativiert.

Sobald die Cache Unit zusammengestellt ist, wird sie an den Cache gesendet, der die Sätze direkt von unten nach oben einfügt. Ein Zwischenlagern der Daten in Staging-Tabellen entfällt. Diese Methode kann vermutlich nur dann einen zusätzlichen Vorteil bieten, wenn zwischen Backend und Cache eine hohe Latenz herrscht. Eine Vermessung der Auswirkungen hoher Latenzen wurde im Rahmen dieser Arbeit noch nicht durchgeführt und sollte Bestandteil zukünftiger Aktivitäten sein.

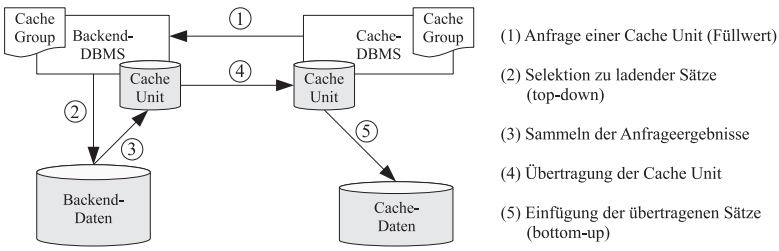



Abbildung 12: Ablauf des vorbereiteten Ladens

### 4.4 Zusammenfassung

Die durchgeführten Messungen zeigen, dass das direkte Laden nur schlecht skaliert und zum Einfügen größerer Cache-Unit-Differenzen nicht gut geeignet ist. Die schlechte Performance führt dazu, dass die zu ladenden Prädikatsextensionen im Cache erst viel zu spät nutzbar werden. Die Methode des indirekten Ladens beschleunigt das Laden der Sätze enorm und somit auch die Geschwindigkeit, in der die neu geladenen Prädikate für die Anfragebearbeitung nutzbar werden. Bei sehr kleinen Cache Units bietet das direkte Einladen jedoch Vorteile, da hierbei eine Indirektion (Zwischenspeichern der Daten) vermieden wird. Mit dem vorbereiteten Laden wurde eine weitere Methode entwickelt, die mehrere entfernte Anfragen auf Backend-Daten vermeidet und dadurch evtl. hohe Latenzzeiten reduzieren kann. Ein CbDBC-System, welches alle Methoden implementiert, kann leicht in die Lage versetzt werden, je nach Situation die richtige Variante dynamisch während der Laufzeit auszuwählen. Gleichzeitig zeigen die Messungen, dass hierfür die Größe der Cache Unit, die Struktur der Cache Group und die Höhe der Latenz wichtige Kenngrößen darstellen.

## 5 Entladen von Cache Units

In diesem Abschnitt stellen wir ein Konzept zum selektiven Entladen von Cache Units vor. Wie bereits in Abschnitt 3.1 erklärt, werden beim Entladen die atomaren Zonen der Cache Group von oben nach unten durchlaufen. Wir bezeichnen diese Vorgehensweise auch als *vorwärtsgerichtetes Entladen*, bei welchem die gelöschten Kontrollwerte wiederum mithilfe der in Abschnitt 4.2 vorgestellten Kontrollwerttabellen weitergereicht werden.

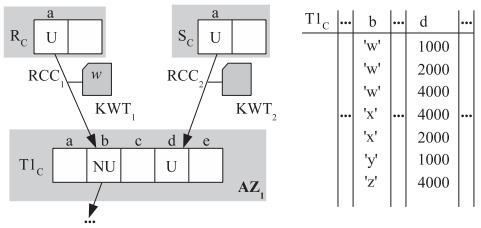



Abbildung 13: Löschen innerhalb einer trivialen Zone

Im Folgenden betrachten wir zunächst das vergleichsweise einfache Entfernen der Sätze aus einer trivialen atomaren Zone. Danach wird das Entladen aus einer nicht-trivialen atomaren Zone besprochen. Abschnitt 5.3 diskutiert die Ergebnisse der Messungen. Zuletzt wird in Abschnitt 5.4 die Verdrängungsstrategie des Cache-Systems vorgestellt.

### 5.1 Entladen in trivialen atomare Zonen

Zur besseren Veranschaulichung benutzen wir die in Abbildung 13 gezeigte Cache Group. Durch die vorberechnete Reihenfolge, in der die atomaren Zonen abzuarbeiteten sind, ist sichergestellt, dass alle KWTs eingehender externer RCCs mit den vollständig gelöschten Werten aus vorangegangenen Zonen gefüllt sind.

In Abbildung 13 wird der Löschvorgang durch den Wert $w$ in $KWT_1$ ausgelöst. $KWT_1$ ist somit *Ausgangspunkt* für den Löschvorgang, der in Tabelle $T1_C$ gestartet werden muss. Wir nennen die Zieltabelle eines externen RCCs, dessen KWT gefüllt ist, daher auch *Starttabelle*.

Um die löschbaren Sätze zu bestimmen, muss jeder Satz mit $T1_C.b = \text{'}w\text{'}$ überprüft werden. Im Beispiel können alle diejenigen Sätze entfernt werden, deren Anwesenheit nicht durch die Kontrollwerte des $RCC_2$ verlangt wird. Enthält die Quellspalte des $RCC_2$ z. B. nur den Wert 1000, so können alle Sätze $\sigma_{(b=\text{'}w\text{'} \wedge d \neq 1000)} T1_C$ gelöscht werden. Das Entladen in trivialen atomaren Zonen kann also durch die Ausführung einer einzigen `DELETE`-Anweisung vorgenommen werden. Das folgende, vereinfacht dargestellte Statement findet und löscht die Sätze der atomaren Zone im Beispiel, wobei alle KWTs eingehender RCCs (hier $KWT_1$ und $KWT_2$) parallel berücksichtigt werden.

```
delete from T1_C
where (b in (select KW from KWT_1) or
       d in (select KW from KWT_2))
  and (b not in (select distinct R_C.a from R_C) and
       d not in (select distinct S_C.a from S_C))
```

## 5.2 Entladen in nicht-trivialen atomaren Zonen

Wir beschäftigen uns nun detailliert mit den Problemen, die im Beispiel 3.2 aufgezeigt wurden. Dort wurde bereits festgestellt, dass es notwendig ist, interne und externe Abhängigkeiten zu unterscheiden. Anhand des in Abbildung 14 gezeigten homogenen Zyklus wird nun erklärt, wie diese Abhängigkeiten aufgelöst werden können.

Von besonderer Bedeutung sind die Werte der Tabellenspalten, die Teil des homogenen Zyklus sind. Wir nennen diese Werte *Zykluswerte*. Die entscheidende Idee, um die Abhängigkeiten innerhalb der atomaren Zone schnell aufzulösen, besteht darin, diejenigen Zykluswerte zu bestimmen, für deren Sätze es keinerlei externe Abhängigkeiten mehr gibt. In Abbildung 14 gilt dies nur für den Wert 1000, da die Werte 2000 und 3000 durch $R_C.a = \text{'}x\text{'}$ und $S_C.a = \text{'}j\text{'}$ extern referenziert werden. Löschen wir nun alle Sätze mit diesem Wert aus allen Cache-Tabellen der Zone (*globales Löschen*), so verbleiben in der atomaren Zone nur noch Sätze, die entweder gar keine Abhängigkeiten mehr haben oder die aufgrund bestehender Abhängigkeiten nicht gelöscht werden dürfen. Die verbleibenden, noch löschbaren Sätze lassen sich nun durch ein intern (nur innerhalb der atomaren Zone) ausgeführtes, vorwärtsgerichtetes Entladen ermitteln (*internes Löschen*). Im Beispiel werden dabei die Sätze mit dem Wert 3000 aus $T1_C$ und $T2_C$ gelöscht; der Satz in $T3_C$ muss jedoch, aufgrund seiner externen Abhängigkeit, erhalten bleiben.

Nachfolgend betrachten wir die einzelnen Schritte dieser Vorgehensweise im Detail. Sie sind für jeden externen RCC, der zu löschende Kontrollwerte übermittelt, durchzuführen, um alle Sätze der Zone zu entladen[5].

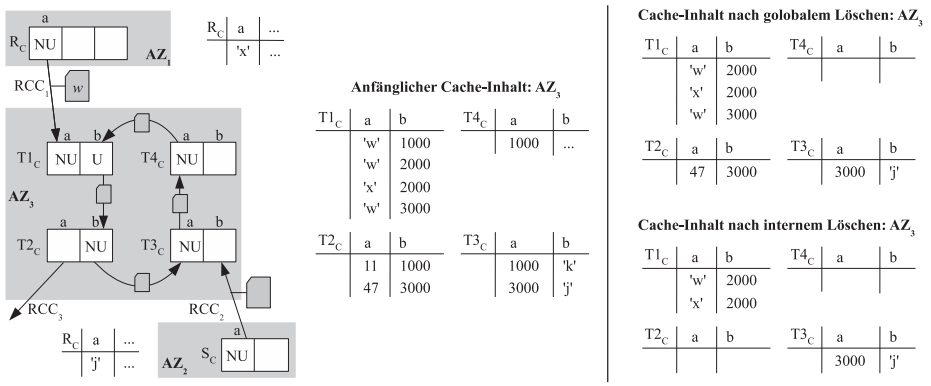



Abbildung 14: Löschen innerhalb nicht-trivialer Zonen

**Globales Löschen.** Wir betrachten nochmals Abbildung 14 und nehmen an, dass der Wert *w* wie angedeutet entladen werden soll. Um die Zykluswerte zu ermitteln, die global im Zyklus löschbar sind, genügt es, einen Verbund zwischen denjenigen Tabellen auszuführen, die einen externen eingehenden RCC aufweisen. Im Beispiel sind das die Tabellen

[5] Es ist möglich, den Algorithmus so zu konzipieren, dass er alle übermittelten Werte gleichzeitig betrachtet. Anhand dieser komplexen Variante lassen sich die Konzepte jedoch nicht gut erklären.

$T1_C$ und $T3_C$. Die folgende Anfrage bestimmt die zu löschenden Zykluswerte:

```
select T1_C.b from T1_C, T3_C
where  T1_C.b = T3_C.a
and    T1_C.a IN (select KW from KWT_1)
and    T1_C.a NOT IN (select distinct R_C.a from R_C)
and    T3_C.b NOT IN (select distinct S_C.a from S_C)
```

Es genügt also die Abhängigkeitsprüfung über die Quellspalten der externen RCCs vorzunehmen. Interne Abhängigkeiten (interne RCCs) müssen nicht überprüft werden, wodurch die Tatsache, dass in allen am Zyklus beteiligten Spalten die selben Werte miteinander verbunden werden, bestmöglich ausgenutzt wird. Im Beispiel wird auf diese Weise der Wert 1000 aus allen Tabellen innerhalb der atomaren Zone entfernt. Während jeder Löschung ist darauf zu achten, ob hierbei ein Kontrollwert vollständig entfernt wird, welcher somit in die entsprechende Kontrollwerttabelle ausgehender RCCs einzustellen ist. Dies ist im Beispiel für den Wert 11 der Fall, welcher vollständig aus $T2_C.a$ entfernt wurde und somit über die KWT von $RCC_3$ weitergeleitet wird.

**Internes Löschen.** Das interne Löschen kann nun auf die gleiche Weise vorgenommen werden wie das Löschen in trivialen atomaren Zonen. Für alle eingehenden RCCs (externe sowie interne) wird beginnend in der Starttabelle überprüft, ob es noch zu löschende Sätze gibt. Ist dies der Fall, wie im Beispiel aus Abbildung 14, so werden diese gelöscht, wobei die dabei vollständig gelöschten Kontrollwerte über **alle** ausgehenden RCCs (also externe wie interne) weitergeleitet werden. Der Löschvorgang wird innerhalb der atomaren Zone so lange fortgesetzt, bis keine interne KWT mehr zu verarbeitende Werte aufweist. Im Beispiel stoppt die Bearbeitung also in Tabelle $T3_C$, da der Wert 3000 aufgrund der externen Abhängigkeit nicht gelöscht werden kann.

Hätte es vor Beginn des internen Löschens einen Satz mit dem Wert 3000 in $T4_C$ gegeben, so wäre das interne Laden bereits sofort nach der ersten Überprüfung in der Starttabelle beendet. Weitere Prüfungen sind überflüssig, da jeder Zykluswert durch eine externe Abhängigkeit motiviert ist.

### 5.3 Messergebnisse

Abbildung 15 zeigt die Zeit, die benötigt wurde, um die beim Laden generierten Einheiten jeweils zu entladen. In allen Strukturen läuft der Entladeprozess dabei enorm schnell ab (meist unter 200 ms). Der anfänglich vergleichsweise hohe Aufwand (bei 100 Sätzen) entsteht durch die erstmalige Optimierung vorbereiteter Anfragen (Prepared Statements) durch die Datenbank. Dieser Aufwand ist auch in allen vorangegangenen Messungen enthalten, wo er aber aufgrund der hohen Aufwände in den Diagrammen nicht zu sehen ist.

Erfreulich ist vor allem das Ergebnis beim Löschen innerhalb homogener Zyklen. Das Löschen lief sogar noch etwas schneller ab als in vergleichbar langen Ketten. Denn für das globale Löschen, welches in unseren Tests die meisten Sätze löscht, wurden weniger Anfragen benötigt als beim Löschen über mehrere atomare Zonen hinweg. Wir konnten somit zeigen, dass ein selektives Entladen in Zyklen effizient durchführbar ist.

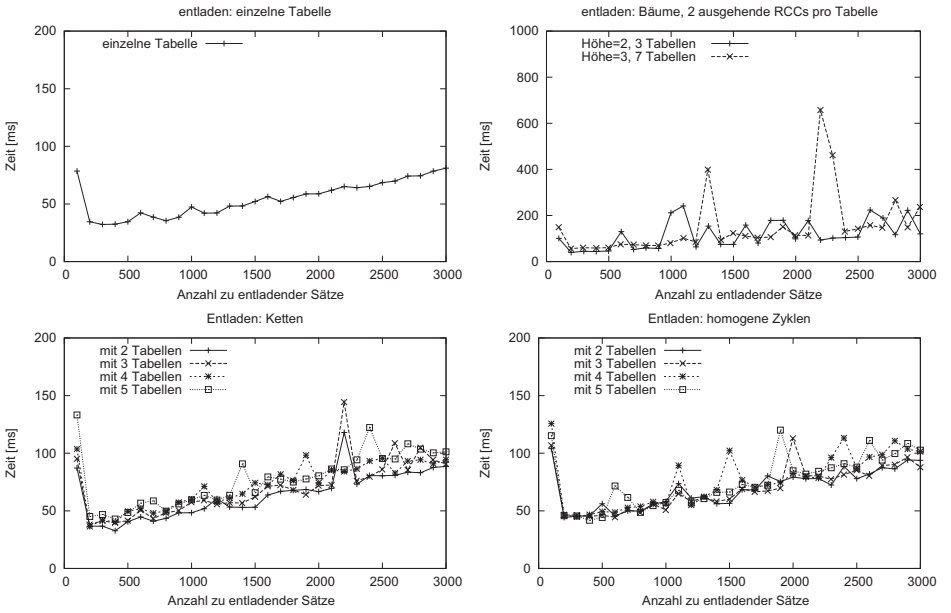



Abbildung 15: Entladen von Cache Units

Überraschend ist das Ergebnis beim Löschen in Bäumen. Beim Baum der Höhe drei, der sieben Tabellen und somit auch sieben atomare Zonen umfasst, steigt der Aufwand zweimal vergleichsweise stark an. In allen Messdurchläufen bei jeweils unterschiedlichen Daten gleichermaßen. Da für die Messungen ein so genannter zirkulärer Log-Schreibmodus verwendet wurde, mussten aufgrund der geringen Größe der Log-Dateien (von dreimal 4096 kB, Seitengröße 4 kB) zwischenzeitlich Seiten aus dem DB-Puffer verdrängt werden, um in den Log-Dateien vorangegangene Einträge überschreiben zu können. Dies wurde durch eine nachträglich vorgenommene Messung mit einem zehnmal größeren Log bestätigt und erklärt somit die zyklisch auftretenden Spitzen. Trotzdem bleibt auch in diesem Fall der Aufwand, um die Cache Units zu entladen, sehr gering.

## 5.4 Verdrängungsstrategie

Als Verdrängungsstrategie wird die allgemein bekannte LRU-k-Strategie (least recently used) eingesetzt. Zur Umsetzung wird jeweils der Zugriffszeitpunkt auf eine Cache Unit in der Kontrolltabelle für den entsprechenden Füllwert vermerkt (vgl. [BHM06]).

Die Entscheidung, wann das Entladen von Cache Units notwendig ist, wird anhand zweier Füllstandswerte getroffen. Der erste definiert die Maximalanzahl an Cache Units, die in eine Füllspalte eingelagert werden dürfen. Der zweite Wert definiert auf die gleiche Weise, bis zu welchem Füllstand entladen werden soll, wenn das Maximum erreicht ist. Der aktuelle Füllstand wird als Verhältnis zwischen der aktuellen Anzahl geladener

Cache Units und der Anzahl der Kandidatenwerte (der Anzahl aller ladbaren Cache Units) ermittelt. Auf diese Weise ist die Verdrängung von Cache Units pro Füllspalte einstellbar, was eine höhere Adaptivität gewährleistet als die Definition von Schranken für die ganze Cache Group. Es ist jedoch nur sehr schwer überprüfbar, ob diese Strategie zum Verdrängen der Cache Units ähnlich gut geeignet ist wie für den Einsatz im Datenbankpuffer. Der größte Unterschied ist, dass beim CbDBC ganze Prädikatsextensionen verdrängt werden und nicht einzelne Seiten. Auch das Ziel des Löschens ist unterschiedlich. Beim CbDBC steht nicht das Freigeben von Speicher im Vordergrund. Wir können davon ausgehen, das stets genug Speicher zur Verfügung gestellt werden kann, um alle möglichen Cache Units vorzuhalten. Es ist vielmehr wichtig, die Zahl der vorgehaltenen Cache Units einschränken zu können, um den Aufwand, z. B. für die Synchronisation von Replikaten möglichst gering zu halten und eine hohe Lokalität zu erreichen.

### 5.5 Zusammenfassung

Die vorgestellte Lösung ermöglicht es, einzelne Cache Units selektiv und effizient zu entladen. Sie ermöglicht jedoch nicht automatisch auch die Verarbeitung heterogener Zyklen, wie dies beim indirekten und vorbereiteten Laden der Fall ist. Das Entladen in homogenen Zyklen konnte so weit verbessert werden, dass nur noch ein sehr geringer Mehraufwand im Vergleich zu anderen Strukturen zu erwarten ist. Darüber hinaus beschränken die aufgezeigten Optimierungen den Aufwand beim Löschen in einfacheren homogenen Zyklen deutlich. Die Parameter der Verdrängungsstrategie lassen sich während der Laufzeit für jede Füllspalte separat anpassen. Hierdurch ist das Gesamtsystem prinzipiell in der Lage, mehr oder weniger Cache Units auf einer Cache-Instanz zuzulassen. Einzelne Cache Units können über die Kandidatenmenge von Füllspalten auch komplett ausgeschlossen werden.

## 6 Fazit

Das CbDBC versucht in seiner Konzeption, deklarative Anfragen auf einem Datenbank-Cache zu unterstützen. Die dazu eingeführten Constraints und die somit entstehenden Abhängigkeitsketten stellen jedoch eine Herausforderung für die Umsetzung der benötigten Funktionalitäten dar (alle Bedingungen müssen jederzeit erfüllt sein). Um weiterhin alle Vorteile einer relationalen Bearbeitung von Anfragen zu nutzen, ist es jedoch wichtig, die neuen Strukturen und Bedingungen auf relationaler Ebene durch leistungsstarke und einfache Methoden handhaben zu können. Mit den in diesem Aufsatz vorgestellten Konzepten und Algorithmen gelingt es, das selektive Laden von Cache Units deutlich zu verbessern und gleichzeitig das Entladen ebenfalls selektiv zu gestalten. Mit den neu eingeführten Definitionen Cache Unit und Cache-Unit-Differenz gelingt es nun, die Größenverhältnisse zu ladender bzw. zu entladender Einheiten genau zu beschreiben. Die gezeigten Verfahren erhöhen die Anpassungsfähigkeit des Cache-Systems, da es möglich ist, das beste Verfahren aus einer Menge von Alternativen auszuwählen. Darüber hinaus lässt sich das Füll- und Entladeverhalten mithilfe von Kandidatenwerten und verschiedener Füllgrade für die Füllspalten sehr gezielt beeinflussen.

## 7 Ausblick

Die vorgestellten Untersuchungen belegen zwar den Nutzen der neuen Verfahren, analysieren aber deren Verhalten immer noch nicht genau genug, um die Größenordnungen entscheidender Faktoren zu bestimmen. Ziel wäre es, z. B. anhand statistisch bzw. direkt erfassbarer Daten (wie die Größe der zu ladenden Cache Unit) entscheiden zu können, welche Ladestrategie zu wählen ist. Hierzu müsste vor allem der Einfluss verschiedener Latenzen noch genauer untersucht werden. Ebenso wurde nicht geklärt, inwieweit die Verbesserungen andere Funktionen beeinflussen. In der Zukunft planen wir daher, weitere Messverfahren zu entwickeln, die uns helfen, wichtige Entscheidungsrößen zu definieren.

## Literatur


[ABK+03] Mehmet Altinel, Christof Bornhövd, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh und Berthold Reinwald. Cache Tables: Paving the Way for an Adaptive Database Cache. In *VLDB*, Seiten 718–729, 2003.

[APTP03] Khalil Amiri, Sanghyun Park, Renu Tewari und Sriram Padmanabhan. DBProxy: A Dynamic Data Cache for Web Applications. In *ICDE*, Seiten 821–831, 2003.

[BDD+98] Randall G. Bello, Karl Dias, Alan Downing, James J. Feenan, Jr., James L. Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski und Mohamed Ziauddin. Materialized Views in Oracle. In *VLDB*, Seiten 659–664, 1998.

[BHM06] Andreas Bühmann, Theo Härder und Christian Merker. A Middleware-Based Approach to Database Caching. In Y. Manolopoulos, J. Pokorný und T. Sellis, Hrsg., *ADBIS 2006*, LNCS 4152, Seiten 182–199, Springer, 2006.

[BK07] Andreas Bühmann und Joachim Klein. Examining the Performance of a Constraint-Based Database Cache. In *IDEAS*, Seiten 290–295. IEEE Computer Society, 2007.

[CLRS82] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. *Introduction to Algorithms*, Kapitel 22.4, Seiten 549–552. The MIT Press, 2. Auflage, 1982.

[GL01] Jonathan Goldstein und Per-Åke Larson. Using Materialized Views: A Practical, Scalable Solution. In *SIGMOD*, Seiten 331–342, 2001.

[HB07] Theo Härder und Andreas Bühmann. Value Complete, Column Complete, Predicate Complete – Magic Words Driving the Design of Cache Groups. *VLDB Journal*, Seiten 805–826, 2007.

[LGGZ04] Per-Åke Larson, Jonathan Goldstein, Hongfei Guo und Jingren Zhou. MTCache: Mid-Tier Database Caching for SQL Server. *Data Engineering Bulletin*, 27(2):35–40, Juni 2004.

[LGZ04] Per-Åke Larson, Jonathan Goldstein und Jingren Zhou. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In *ICDE*, Seiten 177–189. IEEE Computer Society, 2004.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv und Divesh Srivastava. Answering Queries Using Views. In *PODS*, Seiten 95–104, 1995.

[Mer05] Christian Merker. Konzeption und Realisierung eines Constraint-basierten Datenbank-Cache. Diplomarbeit, TU Kaiserslautern, 2005.

[Ora08] Oracle Corporation. Internet Application Server Documentation Library, 2008.

[The02] The TimesTen Team. Mid-tier Caching: The TimesTen Approach. In *SIGMOD*, Seiten 588–593, 2002.

# In-Network Detection of Anomaly Regions in Sensor Networks with Obstacles*

Conny Franke[1] Marcel Karnstedt[2]
Daniel Klan[2] Michael Gertz[3] Kai-Uwe Sattler[2] Wolfram Kattanek[4]
[1]Department of Computer Science, University of California, Davis, U.S.A.
[2]Databases and Information Systems Group, Ilmenau University of Technology, Germany
[3]Institute of Computer Science, University of Heidelberg, Germany
[4]Institute of Microelectronics- and Mechatronics Systems, Ilmenau, Germany

**Abstract:** In the past couple of years, sensor networks have evolved to a powerful infrastructure component for monitoring and tracking events and phenomena in many application domains. An important task in processing streams of sensor data is the detection of anomalies, e.g., outliers or bursts, and in particular the computation of the location and spatial extent of such anomalies in a sensor network. In this paper, we present an approach that facilitates the efficient computation of such anomaly regions from sensor readings. We propose an algorithm to derive spatial regions from individual anomalous sensor readings, with a particular focus on obstacles present in the sensor network. We improve this approach by proposing a distributed in-network processing technique where the region detection is performed at the sensor nodes. We demonstrate the advantages of this strategy over a centralized processing strategy by utilizing a cost model for real sensors and sensor networks.

## 1 Introduction

Driven by major advancements in sensor technology, several sensor networks have been and are being deployed in various application domains such as the monitoring of traffic, buildings, rivers, and the environment in general. Typical examples for environmental monitoring include precision agriculture (e.g., observing the humidity of the soil) and monitoring particles in urban areas to react to changes in air quality measures. An important objective in processing sensor data is the detection of anomalies that occur, e.g., in the form of outliers or bursts. This kind of data processing and analysis not only reduces the volume of data reaching end user applications but it also simplifies the further processing and interpretation of the sensor data.

By analyzing individual and aggregated sensor measurements, one can obtain useful information about the locations where anomalous events and phenomena occur. Such location information then can be visualized on a map and interpreted for individual sensors. In particular it can be used to derive *anomaly regions*. Such regions are composed of neighboring

*This work was in part supported by the National Science Foundation under Award No. ATM-0619139 and by the BMBF under grant 03WKBD2B.

sensors that show anomalous readings and are combined to describe a polygonal anomaly region. Compared to information about only individual (anomalous) sensors, providing users with such region information, including their spatial extent, has several advantages.

- It represents a natural way of event aggregation and correlation as needed in many monitoring applications, such as impact analysis.
- The location information associated with sensor data and regions allows for a direct processing of the results, e.g., for tracking regions.
- By approximating the regions in the unobserved space between sensors exhibiting normal and anomalous readings, one can determine region boundaries that more closely reflect the true boundaries of an event detected by a group of sensors. For this, one can also take propagation characteristics of detected events as well as natural and artificial obstacles occurring in the sensor network region into account.

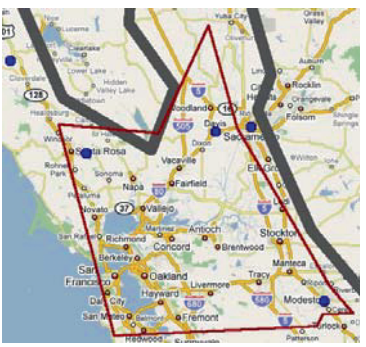



Figure 1: Example of anomaly region

For example, in the context of environmental monitoring this then not only allows for the detection of anomalies as measurement points on a map but also for determining regions where related anomalous values occur. Furthermore, obstacles such as buildings, ridges, rivers, and valleys are taken into account for predicting the propagation of anomaly regions. This aspect is illustrated in Fig. 1 where wind speed values are measured by the CIMIS sensor network [cim] in California. Based on the sensor locations (indicated by the blue dots) and the obstacles (here ridges indicated by thick gray lines) the marked region (read polygon) can be derived and placed on a map.

It should be noted, however, that the benefit of detecting anomaly regions and the underlying spatial aggregation of anomalous sensor readings can only be exploited if the event aggregation is performed locally at the affected sensor nodes or their close neighborhood, respectively. Particularly for wireless and battery-powered sensors, the expected reduction of expensive radio communication might improve the lifetime significantly when such a processing of sensor data is performed locally and intermediate results from groups of sensors are propagated in a hierarchical fashion. In this paper, we present such an approach for the distributed detection of anomalous spatial regions in sensor networks. In particular, the main contributions of our work are as follows:

1. We discuss a framework for anomaly detection that isolates the threshold-based region detection from the actual anomaly detection and, therefore, is orthogonal to the event detection that could be triggered by outliers or bursts.
2. In our approach for determining anomaly regions, we consider natural and man-made obstacles that might damp the effect of an event and thus need to be considered appropriately in determining the spread of (potential) anomaly regions.

3. We present a distribution strategy for the in-network detection and processing of anomalous sensor readings and deriving anomaly regions. This strategy can lead to significant savings in power consumption. We demonstrate the capabilities of the in-network detection approach using an evaluation based on real sensor network characteristics.

This paper is organized as follows: In Section 2, we introduce the scenario and goals of this paper. We also present our framework for the detection of anomaly regions. Section 3 summarizes related work in the areas of anomaly detection, region detection, obstacle handling, and in-network processing. In Section 4, we present our algorithm for detecting anomaly regions in the presence of obstacles. We discuss the benefits of the in-network computation of anomalies and anomaly regions in Section 5. The corresponding evaluation and experimental results are presented in Section 6. Section 7 concludes the paper.

## 2 Background and Setup

We assume a sensor network $S$ comprised of $m$ stationary sensors, $\mathcal{S} = \{s_1, \ldots, s_m\}$. Each sensor $s \in \mathcal{S}$ has a *spatial attribute*, $\langle x_s, y_s \rangle$, which defines its location in 2D space. Our approach is also applicable to a 3D setting, where nodes in the network are given by their $x_s$, $y_s$, and $z_s$ coordinate to account for different elevations. For ease of presentation, we focus on 2D scenarios. The sensors are distributed non-uniformly in the network and monitor the *same* environmental *variable* such as temperature, humidity, or wind speed.

For a sensor $s$, a measurement of a variable is denoted $r_{s,t}$, with the timestamp $t$ indicating when the variable reading was obtained. The network in our setting is *synchronized*, i.e., a set of $m$ new measurements is processed in the network each time period. Synchronous processing is not a strict requirement for our method, but eases the processing of measurements as well as explaining the functionality of our technique.

Based on the spatial attribute of sensors a *spatial neighborhood* $N_f(s_i) \subseteq \mathcal{S}$ can be defined for each sensor $s_i \in \mathcal{S}$. A suitable neighborhood function $f$ allows for different metrics, such as distance based neighbors (given a maximum distance $r$) or k-nearest neighbors.

### 2.1 Degree-Based Anomalies

Anomaly detection is a broad field that comprises areas like outlier detection, deviation detection, and burst detection. Anomalies of any kind are, by definition, data points that appear anomalous when compared to other data points in a data set or stream. For example, bursts are characterized as "abnormal aggregates in data streams" by Zhu et al. [ZS03]. An outlier is described as "a data point that is significantly different from the rest of the data points" in [BM07].

In *threshold-based* approaches, a threshold is used to separate two categories of data points, anomalous and normal ones. Some algorithms in the field of outlier detection use the notion of *degree-based outliers*, e.g., [FG08, WCD+07], to better capture the intensity

of the observed anomaly. In this context, an *anomaly degree*, $AD \in [0, 1]$, is determined for each data point. By using an $AD$ value to describe a data point, it is taken into account that some data points are more clearly anomalous than others. When analyzing a data stream, each sensor $s$ and measurement $r_{s,t}$, respectively, is assigned a value $AD \in [0, 1]$, which can change with each new measurement the sensor obtains. An $AD$ value of 0 indicates that the measurement obtained by $s$ at time $t$ is normal.

A *reference* is necessary to answer the question "$r_{s,t}$ is anomalous with respect to which other measurements?". In a spatial setting, it is common to use the spatial neighborhood $N_f(s)$ as reference. If only previous values of $s$ are used to determine the $AD$ of sensor $s$ at time $t$, then $N_f(s) = \emptyset$. The other extreme is to set $N_f(s) = S$. Then measurements from all nodes in the network are used as reference. Between these two extremes, other definitions of $N_f(s)$ are possible, as mentioned above.

At time $t$, an anomaly detection algorithm is applied to each of the $m$ new measurements. The output of the anomaly detection algorithm is a stream of tuples $(s_i, t, AD)$, i.e., at time $t$ sensor $s_i$ has the anomaly degree $AD$.

In the following, we use two different approaches for anomaly detection, a degree-based outlier detection algorithm [FG08] and a burst detection algorithm [KKPS08]. Both algorithms determine the $AD$ value of a measurement with the help of two threshold parameters $k_{low}$ and $k_{high}$. If the measurement is between the two thresholds $k_{low}$ and $k_{high}$, its $AD$ value is computed based on its distance to $k_{low}$, i.e., the farther from $k_{low}$ the measurement is, the higher is its assigned $AD$ value. Otherwise the measurement is assigned $AD = 0$ or $AD = 1$ depending on whether it is above or below both thresholds.

### 2.2 Anomaly Regions and Obstacles

*Anomaly regions* are time-variant spatial regions in a sensor field where unusual phenomena or events are taking place at some point in time. Detecting event regions and their boundaries has been studied in, e.g., [FG08, KZ06], but so far obstacles in the sensor field have not been taken into account when constructing such regions.

For anomaly region detection we use the *TWISI* (Triangulated WIreframe Surface Intersection) approach proposed in [FG08], where polygonal anomaly regions are constructed with respect to an intensity threshold $\varphi$. At each point in time, the currently detected anomalous sensors are used for region construction. A user specified value $\varphi \in [0, 1]$ is used to select a subset of all detected anomalous sensors, i.e., only those sensors having $AD \geq \varphi$ should be included in an anomaly region. A region's boundary is placed in the unobserved space between anomalous and normal sensors. It is placed in such a way that we assume a measurement taken at a location next to the boundary would have an $AD$ value close to $\varphi$. In Section 4 we briefly outline how region detection using the TWISI approach works.

We use the TWISI approach as the basis for our anomaly region detection because TWISI's boundary placement is very accurate. To illustrate this, we use the Intel lab sensor data [Int], which provide temperature measurements from 54 sensors deployed in the Intel Berkeley Research lab. Figure 2 shows a section from the region detected by TWISI. The black lines are part of the region boundary, and each of the sensors is labeled with its

current $AD$ value. The gray sensor in the figure is a control point that does not contribute to the region boundary detection. It is used to check if the boundary placement is accurate. When setting the intensity threshold $\varphi$ to $0.25$, it can be seen that the control point having $AD = 0.27$ is located fairly close to the region boundary and inside the anomaly region. This shows that the boundary placement is meaningful with respect to the values that could be measured by new sensors, like the gray sensor in Figure 2, that are placed in the unobserved space between existing sensors, like the sensors having $AD$ values $0.0$ and $0.32$ in Figure 2.

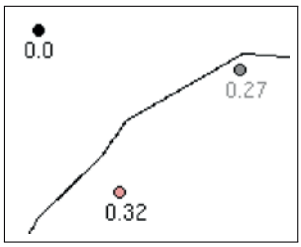



Figure 2: Accuracy of boundary placement

*Obstacles* in a sensor field are typically physical barriers like walls, buildings, rivers, or mountains. In 2D, obstacles are commonly modelled as simple polygons (see, e.g., [THH01]). Obstacles might damp the effects of a phenomenon, but do not necessarily stop its spread completely. A wall in a building will damp the effect of a cold room on the adjacent rooms, but the adjacent rooms' temperature will nevertheless be affected. In contrast, a draft in one room will not spread through walls to adjacent rooms. Thus, obstacles provide different damping factors for different phenomena. As the damping factor of large obstacles can vary, e.g., a mountain does not provide the same damping everywhere, we assign a damping factor $df(s_i, s_j) \in [0, 1]$ to each pair of sensor nodes, according to the obstacle(s) between the two sensors. Obstacles do not necessarily have to be physical barriers, as the air between two sensor locations can act as an obstacle as well, thereby damping the effect of an event due to the distance. Our approach is not limited to symmetric damping factors between two sensors, i.e., it is possible to define $df(s_i, s_j) \neq df(s_j, s_i)$.

By taking obstacles into account, we select a subset of all anomalous sensors detected at time $t$ to be included in the anomaly region. This step uses the stream of anomalies as input, and works on a jumping window such that the most recent $AD$ values of all $m$ sensors are considered. The main purpose of detecting anomaly regions is to indicate the spread of events. We therefore use information about obstacles to extend the regions by also including anomalies having $AD < \varphi$. An anomalous sensor $s$ with $AD_s < \varphi$ is included in a region if there is an obstacle between the source of an event and sensor $s$ that damped the effect of this event. Assume $\varphi = 0.45$ and two sensors $s_1$ and $s_2$ with $AD_{s_1} = 0.29$ and $AD_{s_2} = 0.51$. Also assume an obstacle between $s_1$ and $s_2$ that incurs a damping factor of $df(s_1, s_2) = 0.2$. Sensor $s_2$ is clearly included in the anomaly region, as $AD_{s_2} \geq 0.4$. The event spreads from $s_2$ to $s_1$, but is damped by the obstacle. We therefore expect the $AD$ value of $s_2$ to be lower than it would be without the obstacle, and decrease the threshold $\varphi$ for including $s_2$ in the region by the damping factor. This step is called *threshold propagation*. By doing so, $s_2$ is now only required to have $AD \geq \varphi - df(s_1, s_2) = 0.45 - 0.2 = 0.25$ in order to be included in the anomaly region. As $AD_{s_1} = 0.29 \geq 0.25$, the detected region includes $s_1$ and $s_2$ This example is illustrated in Figure 4(b).

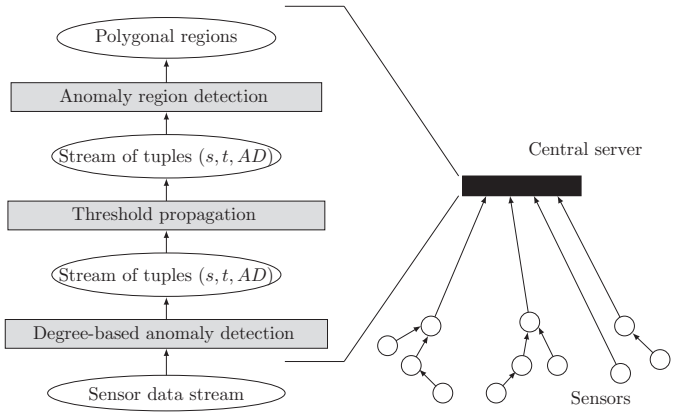



Figure 3: Conceptual and physical architecture of our framework

### 2.3 Three Tier Framework

All three steps, anomaly detection, threshold propagation, and anomaly region detection are combined into a three tier framework, as illustrated on the left hand side of Figure 3. Within this framework, the incoming stream of sensor measurements is piped through the different algorithms, which in the end output a stream of anomaly regions over time. Note that this framework comprises three modular processing steps, and therefore each of the three components can be replaced independently. For example, anomaly detection can be done using the burst detection or outlier detection approach mentioned above. Also, in an obstacle-free sensor field the second tier, threshold propagation, can be omitted without any changes to the remaining framework. The left hand side of Figure 3 shows the conceptual architecture of our framework, whereas the physical architecture is depicted on the right hand side. The latter consists of a hierarchically organized sensor network and a central server. Details about the physical architecture are presented in Section 5.

## 3 Related Work

**Anomaly Detection** Wu et al. [WCD+07] propose a degree-based outlier detection algorithm for static data sets. Franke et al. [FG08] do the same for data streams. The output of such algorithms is the basis for the threshold propagation and region detection we propose in this paper. Other anomaly detection methods can be used as well, for example the burst detection algorithm proposed by Klan et al. in [KKPS08]. Their approach can be easily modified to detect degree-based anomalies by adding a second threshold $k_{high}$ and computing $AD$ values as described in Section 2.1. Similar modifications can be applied to other anomaly detection algorithms, e.g., [SPP+06, ZS06].

**Region Detection** Other region or boundary detection algorithms in sensor fields, e.g., [DCXC05, KZ06], place the region boundary right next to the sensors that are on the edge of a region having distinct properties. Some papers, e.g., [DCXC05], define the region boundary as the set of sensors that are in the interior of a region but close to sensors outside the region. In contrast, our boundary placement is more considerate. We place the boundary between anomalous and normal sensors in a meaningful way, and its exact location depends in the intensity of an event at different locations.

Using spatial clustering algorithms, e.g., those mentioned in [HKT01], to partition the sensor field in anomalous and normal regions would not result in an accurate boundary placement either. This is because clustering aims at finding distinct groups of sensors rather than the exact location of the boundary between each two groups.

**Obstacles** Many publications deal with various data mining techniques in the presence of obstacles, e.g., [THH01, ZPMZ04]. However, in these methods obstacles are considered impenetrable objects that need to be bypassed, for example, to compute the distance between two objects as done in [ZPMZ04]. In contrast, we consider obstacles to be permeable albeit having different properties than their surroundings. We achieve this by defining a damping factor for pairs of sensors that are separated by one or more obstacles. This way, our definition subsumes existing definitions of obstacles, as a damping factor of 1 results in a impenetrable obstacle, providing absolute damping.

**In-Network Computation** TinyDB [GM04] and Cougar [YG02] are two well established query processing systems for sensor networks. Both systems support in-network processing with respect to data quality and sensor node life time. The essential difference is the used aggregation strategy. In TinyDB all sensor nodes are of the same type, whereas Cougar distinguishes three classes of nodes: sources nodes, intermediate nodes for data processing like aggregations, and gateway nodes, which connect the user. In order to decrease energy-consumption, both systems build aggregation trees to aggregate sensor data in nodes at higher levels within the routing tree. Building an optimal aggregation tree is NP-Hard. In [KEW02] the authors investigated the performance of aggregation in sensor networks and presented some heuristics to generate suboptimal aggregation trees.

Sensor placement in the network can have a significant impact on the communication costs of in-network processing. Dhillon et al. [DC03] propose an algorithm that places sensors in the network with the goal of effective coverage of the area. The sensor placement generated by the pSPIEL algorithm by Krause et al. [KGGK06] aims at minimizing communication cost between sensors and placing sensors at the most informative locations. In both papers obstacles are taken into account when finding the optimal sensor placement.

## 4 Detecting Anomaly Regions

The basis for our anomaly region detection is the TWISI method proposed in [FG08]. The TWISI approach assumes a barrier-free network, where events spread unhindered between

nodes. However, obstacles like buildings or mountains can obstruct the direct spread of temperature, wind, fine particles, etc. We therefore extend the TWISI approach to take obstacles into account. In the next paragraph, we describe the original TWISI method as proposed in [FG08], and then introduce our extensions.

The first step in TWISI is to construct a Delaunay triangulation of the sensor network using sensors as nodes in the triangulation. Then, a third dimension is added to represent the $AD$ values of sensors, i.e., nodes are assigned a height according to their $AD$ value. This results in a 3D surface, called triangulated wirefame surface or TWS for short, where outlier regions stand out as "hills". The height of each node is updated periodically when new measurements are obtained by the sensor and consequently its $AD$ value is recomputed. To detect anomalous regions, a plane parallel to the x/y plane is intersected with the TWS at height $\varphi$, yielding a set of line segments where the plane intersects the different triangles of the triangulation. The projection of these line segments onto the x/y plane represents the boundaries of anomaly regions, which are polygons. The TWISI approach includes all anomalous sensors with $AD \geq \varphi$ in the generated regions.

Now, we show how to extend the TWISI approach to take obstacles into account. The goal is to propagate the original intensity threshold $\varphi$ through the network such that also sensors having an $AD < \varphi$ might be included in the final anomaly region. This is motivated by the fact that the effect of an event might be damped by the obstacles in the network. By taking this damping factor betwen pairs of sensors into account, the anomaly region is extended such that we can observe the spread of a phenomenon taking the effect of obstacles into account. When propagating the threshold $\varphi$ from $s_1$ to $s_2$, its value is lowered according to the damping factor between both nodes.

After the threshold is propagated through the entire network, the TWISI approach is applied. Due to the lowered threshold at some of the nodes, not one plane is used to intersect the TWS, but several planes at different heights, according to the threshold propagated to each of the sensor nodes. The resulting anomaly region is still a polygon, constructed from the line segments generated by the intersection of the planes at different heights with the TWS.

**Propagation Algorithm** The propagation algorithm works as described by Algorithm 1, and is iterative. All nodes that will be included in the anomaly region and their respective thresholds are stored in the data structure $\mathcal{O}$. In the initial iteration 0, we identify anomalies having $AD \geq \varphi$, add them to $\mathcal{O}$, and mark these sensors as visited by adding them to $S_{marked}$. We call these "level 0 anomalies", and their threshold is set to $\varphi$ (lines 1 – 4). Then, in each subsequent iteration $i$ the threshold is propagated from each node $o \in \mathcal{O}$ of the current level $i$ to its direct neighbors, denoted $Neigh(o)$, i.e., all nodes that are connected to $o$ by an edge in the triangulation of the network (line 7). This is done as long as new nodes are added to $\mathcal{O}$ in one iteration (line 5). If the neighbor $n$ is an anomaly and has not been marked yet (line 8), the damping factor $df$ between $o$ and $n$ is determined (line 10). The propagated threshold $\Delta$ of $n$ is computed by subtracting the damping factor from $o$'s threshold, i.e., $n.\Delta = o.\Delta - df$. If $n$ is a direct neighbor of more than one level $i$ anomaly, then we choose the largest of the propagated thresholds to prevent over-damping (lines 11 – 13). If $n$ is not in $\mathcal{O}$ yet, i.e., it is not a direct neighbor of any of the level

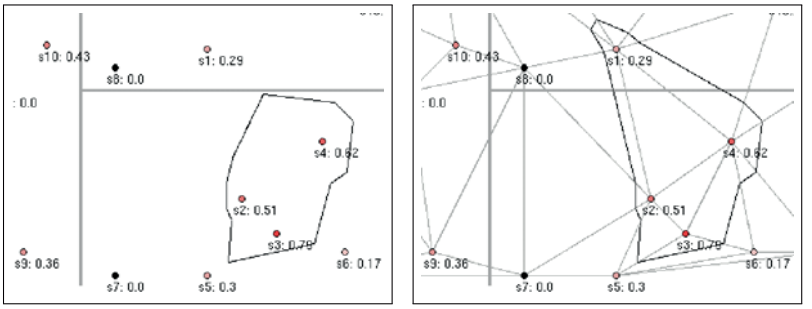
$i$ anomalies checked so far, then $n$ is added to $\mathcal{O}$ at level $i+1$ (lines 14 – 15). After checking all direct neighbors of all level $i$ anomalies, we remove nodes $o$ from $\mathcal{O}$ where the $AD$ value is less than their propagated threshold $\Delta$ (line 16). This way, only nodes with $AD$ value above the propagated threshold are included in the final anomaly regions.

**Input**: $\varphi$
**Output**: set of polygons
1 $\mathcal{I}$ = get-anomalies(); /* $i \in \mathcal{I}$ of the form $[SID, AD]$ */
2 $\mathcal{O} = \{[o.SID, o.AD, \varphi, 0] | o \in \mathcal{I} \wedge o.AD \geq \varphi\}$; /* $o \in \mathcal{O}$ of the form $[SID, AD, \Delta, lvl]$ */
3 $S_{marked} = \{o.SID | o \in \mathcal{I} \wedge o.AD \geq \varphi\}$;
4 $level = 0$;
5 **while** $\exists o \in \mathcal{O} : o.lvl = level$ **do**
6 $S_{checked} = \emptyset$;
7 **foreach** $o \in \mathcal{O} : o.lvl = level$ **do**
8 **foreach** $n \in Neigh(o) \cap \mathcal{I} : n.SID \notin S_{marked}$ **do**
9 $S_{checked} = S_{checked} \cup \{n.SID\}$;
10 $df$ = get-damping-factor($o.SID, n.SID$);
11 **if** $\exists o_n \in \mathcal{O} : o_n.SID = n.SID$ **then**
12 **if** $o_n.\Delta < o.\Delta - df$ **then**
13 $o_n.\Delta = o.\Delta - df$;
14 **else**
15 $\mathcal{O} = \mathcal{O} \cup \{[n.SID, n.AD, o.\Delta - df, level + 1]\}$;
16 $\mathcal{O} = \mathcal{O} \setminus \{o \in \mathcal{O} | o.AD < o.\Delta\}$;
17 $S_{marked} = S_{marked} \cup S_{checked}$;
18 $level = level + 1$;
19 **return** *get-and-combine-line-segments*($\mathcal{O}$);

**Algorithm 1**: Centralized threshold propagation and region detection algorithm

Marking visited sensors after each iteration prevents cycles, where the threshold of a node would initially be set in iteration $i$ and then overwritten in iteration $j > i$ because of a chain of direct neighbors being included in $\mathcal{O}$. In combination with the iterative approach, marking visited sensors causes the threshold to be propagated to each node in only one iteration, and this iteration corresponds to the minimum number of hops from the level $0$ anomalies. That is, each node is visited "as soon as possible", starting at the nodes that are initially above the threshold $\varphi$, and the propagated threshold for each node can not be overwritten in later iterations.

Figure 4 illustrates the effects of threshold propagation, using the example we already discussed in Section 2.2. The intensity threshold is set to $\varphi = 0.45$ in both figures. Each sensor is labeled with its sensor id and $AD$ value. The triangulation of the nodes is shown in Figure 4(b) by the thin gray lines. The thick gray lines mark obstacles between sensors, which induce damping factors of $0.2$ between each pair of sensors that is connected by an edge in the triangulation. Figure 4(a) depicts the anomaly region that was detected without threshold propagation. Sensor $s_1$ is not included in the region, although it is anomalous and fairly close to sensors that are inside the region, i.e., it is a direct neighbor of sensors $s_2$ and $s_4$, which are included in the anomaly region. Due to this proximity we would like to include $s_1$ in the region if its $AD$ value, considering the damping factors to $s_2$ and $s_4$ respectively, is sufficiently high. This will be determined using threshold propagation.




(a) Without threshold propagation (b) With threshold propagation

Figure 4: Anomaly regions without and with threshold propagation

In Figure 4(b) threshold propagation was applied before constructing the anomaly region. The region in Figure 4(b) spreads to the area above the obstacle and includes the anomalous sensor $s_1$ there. This is what we wanted to achieve, as it provides us with additional information about the phenomenon we detected in the area below the obstacle. That is, the phenomenon spreads to sensors in the proximity of affected sensors in the lower area, i.e., to $s_1$, although $s_1$ is shielded from the phenomenon by an obstacle. In contrast, the region and thus the phenomenon does not spread to the area on the left of the obstacles, because the sensors $s_7$ and $s_8$ that are in the close proximity of the anomalies $s_9$ and $s_{10}$ are normal. The phenomenon in the lower area cannot spread through normal sensors to the anomalous sensors. Technically speaking, sensors $s_9$ and $s_{10}$ were not included in the region because they do not have a direct neighbor that has been added to the data structure $\mathcal{O}$ and thus could have propagated the threshold.

## 5 Distributed Approach

In wireless networks, sending and receiving messages is much more energy consuming than local processing. As energy consumption (measured in Joule $J$) is a crucial (if not the most crucial) cost factor in wireless networks, the number of messages should be minimized. In the centralized approach proposed up to here, all data sources, i.e., the sensors in the network, periodically send their data to a central server where it is analyzed and processed. Thus, a promising idea is to distribute the processing costs and by this hopefully lower the number of messages needed. This can be achieved by pushing (parts of) the processing steps down into the network, which is called *in-network processing*. Actually, there is a choice on the degree of distribution. As an opposite to the centralized processing, all processing steps are completely delegated to the sources and only detected anomalies are signalized to a central sever. We assume a multi-hop network having a hierarchical organization, similar to the one used in [SPP+06]. The idea is to partition the network using virtual grids. The network has several levels: at the lowest level sensors in a local area are combined in one grid cell, and cells at higher levels subsume multiple cells from lower levels. At the highest level is one cell (the central server) representing the entire network. Each cell at each level (except the root cell) has a leader node which can be either cho-

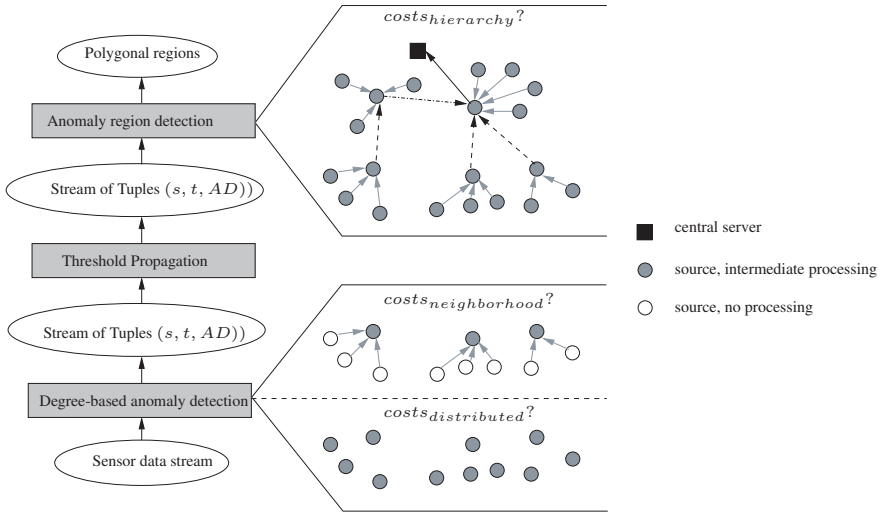



Figure 5: Possibilities of in-network processing

sen from the nodes in the network or it can be a virtual node. Like this, the hierarchy of nodes can be illustrated as a tree. In our setup we assume the same tree for both, multi-hop message passing and in-network processing (which is rather intuitive). This implies that each node can reach its parent node in one single hop. In [KGGK06] Krause et al. give an algorithm that can be used to partition the sensor network into grid cells and choose leader nodes. The resulting partition takes obstacles in the network into account, and thus it is unlikely that sensors within one cell are separated by one or more obstacles. Such an obstacle-aware partitioning of the network is desirable for our distributed algorithms.

Figure 5 illustratively summarizes the focus of the following section. For anomaly detection, we have three choices:

1. send all data to a central server for processing
2. choose leader nodes that collect data from all peers in their neighborhood and process the data
3. detect anomalies at each source separately

Option 3 is only practicable if anomalies are independent from neighboring sources, because otherwise a full exchange between all sources in a neighborhood is needed.

Threshold propagation and anomaly region detection cannot be processed on the individual sources, i.e., on the sensors, or for each neighborhood independently, because we also have to detect regions crossing neighborhoods. Thus, we only have the options:

1. send all data to a central server for processing
2. use a hierarchy between chosen leader nodes that exchange data accordingly

Obviously, threshold propagation and region detection can only be processed in-network if anomaly detection is done in-network as well. As all properties and statements made in the following equally apply to threshold propagation and region detection, from now on we use only region detection when referring to both methods, threshold propagation and region detection.

The choice on the degree of distribution depends on the trade-off between processing and transmission costs. For making the right decision on this, we will discuss an appropriate cost model. The crucial part is the energy consumption observed at the data sources and hierarchy peers. Thus, the factors influencing the total costs $C$ (in $\frac{\mu J}{s} = W$ (Watt)) are:

- $c_{msg}$: constant costs for a single message (header etc.) in $\mu J$
- $c_{byte}$: additional costs for each byte in a message in $\mu J$
- $c_{cpu}(op)$: costs for processing operation $op$ on a node in $\mu J$
- $r_m$: the rate of taking measurements in $\frac{1}{s}$
- $r_a$: the rate of events, i.e., the average rate an anomaly is detected, in $\frac{1}{s}$
- $m$: number of sources contained in the network
- $m_l$: the number of leader nodes (the number of separated neighborhoods, respectively)
- $h$: average number of hops from a source to the central server (correlating with shortest paths in the node hierarchy)

In the following, we will develop general cost formulas for the different options of in-network processing. In Section 6 we will use concrete cost values in order to analytically evaluate the different choices.

### 5.1 Distributed Anomaly Detection

In the centralized approach, we consider the costs for transmitting data. The costs for processing at the central server are not the focus of this work, because we assume a powerful machine with external power supply for that. Usually, not every node is in radio range to the central server. Thus, messages are routed in a multi-hop manner using the hierarchy of nodes. Sending a message always results in a constant overhead $c_{msg}$ due to header information etc. Additionally, costs depend on the size of the data contained, measured in bytes ($c_{byte}$ for each byte). Receiving a message results in energy consumption as well. In our experiments, we observed that this is about the same costs as sending a message. See Section 6 for more details on this. A single measurement can be expressed using 2 bytes. Thus, we obtain the following costs for data transmission in the centralized approach:

$$C_{centr} = \underbrace{h \cdot r_m \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot m}_{\text{send measurements}} + \underbrace{(h-1) \cdot r_m \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot m}_{\text{receive measurements}}$$

Even if there exist techniques for collision prevention (based on time slots or ready/clear signals), there is a small probability of colliding messages. For convenience we omit this in our cost function, as it would only result in a small fraction of resent messages. Further, we assume all sources have the same periodicity, i.e., all sensors produce new measurements at the same frequency, and that messages are forwarded directly without collecting them at intermediate peers.

Several detection algorithms can be directly mapped to the data sources (of course, assuming that respective processing capabilities exist on the sensors). This holds, for instance,

for the introduced burst detection algorithm and the outlier detection if no information about the neighborhood is involved. For anomaly detection on the individual sensors, we obtain the following costs:

$$C_{anomaly} = \underbrace{r_m \cdot c_{cpu}(update) \cdot m}_{\text{update at sources}} + \underbrace{h \cdot r_a \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot m}_{\text{send anomalies}} + \underbrace{(h-1) \cdot r_a \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot m}_{\text{receive anomalies}}$$

Obviously, this can only help reducing energy consumption if $r_a$ is significantly lower than $r_m$, which should be the usual case, as we are dealing with anomalies rather than normal situations. For popular sensors, $c_{cpu}$ is orders of magnitude lower than $c_{msg}$.

If we take information about neighboring sources into account when determining anomalies, we make leader nodes responsible for detecting anomalies in each neighborhood. Sensors send messages containing single measurements to the leader nodes of their neighborhood and processing is done there. Then, we have

$$C_{lead} = \underbrace{r_m \cdot 2 \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot (m - m_l)}_{\text{send and receive measurements}} + \underbrace{r_m \cdot c_{cpu}(update) \cdot m}_{\text{update at leader nodes}} + \underbrace{(h-1) \cdot r_a \cdot (c_{msg} \cdot m_l + 2 \cdot c_{byte} \cdot m)}_{\text{send anomalies}} + \underbrace{(h-2) \cdot r_a \cdot (c_{msg} \cdot m_l + 2 \cdot c_{byte} \cdot m)}_{\text{receive anomalies}}$$

As this assumes uniform distribution of all $r_a \cdot m$ anomalies over the $m_l$ leader nodes, the costs $C_{lead}$ represent an upper bound on the cost of anomaly detection at leader nodes.

The approaches for anomaly detection introduced in this work have no requirements regarding how much processing should be pushed into the in-network hierarchy. In fact, the processing can be totally distributed or is done at the individual leader nodes in case we have to handle neighborhoods. This does not hold for distributed region detection, where communication between leader nodes is mandatory. Depending on the structure and extent of a detected anomaly region, this can result in completely traversing a hierarchy of leader nodes potentially up to the central server on the very top of it. We discuss this approach of distribution and the corresponding costs in the following subsection.

### 5.2 Distributed Region Detection

At time $t$, the $AD$ values of all sensors in one cell are collected at the cell's leader node. Then, threshold propagation is conducted as shown in Algorithm 2. At leader nodes of the lowest level, this algorithm works very similar to the centralized approach described in Algorithm 1, as can be seen in the comments below line 1 and line 22 as well as in lines 23–25 of Algorithm 2. Here, $\mathcal{I}$ contains only the outliers contained in the cell, not all outliers in the sensor network. All nodes that are direct neighbors of nodes in $\mathcal{O}$ but

**Input**: $\varphi$ [, sets $\mathcal{O}, \mathcal{P}, S_{marked}, \mathcal{I}$ from sub-cells]
**Output**: set of line segments, $\perp$ if delegated to next level

1 **if** $\mathcal{P} = \emptyset$ **then** /* only possible at leader nodes lowest in hierarchy */
  /* fill $\mathcal{I}, \mathcal{O}$ and $S_{marked}$ as in lines 1-3 of Algorithm 1 */
2  $\mathcal{P} = \emptyset$; /* $p \in \mathcal{P}$ := $[SID_1, SID_2, \Delta, lvl]$ */
3 **else**
4  merge sets $\mathcal{O}, \mathcal{P}, S_{marked}$ and $\mathcal{I}$ from all sub-cells;
5 $level = 0$;
6 **while** $\exists o \in \mathcal{O} \cup \mathcal{P} : o.lvl \geq level$ **do**
7  $S_{checked} = \emptyset$;
8  **foreach** $p \in \mathcal{P} : p.lvl = level$ **do**
    /* only possible at intermediate nodes at higher hierarchy levels */
9  **if** $p.SID_2 \notin S_{marked}$ **then**
10  **if** $\nexists s \in \mathcal{I} : s.SID = p.SID_2$ **then**
11  **if** $p.SID_2 \in LocalCell$ **then**
12  request $\mathcal{I}$ and $S_{marked}$ from corresponding sub-cell and merge locally;
13  **else**
14  continue ; /* $p$ is kept in $\mathcal{P} \rightarrow$ one level up */
15  $df$ = get-damping-factor($p.SID_1, p.SID_2$);
16  **if** $\exists o_p \in \mathcal{O} : o_p.SID = p.SID_2$ **then**
17  **if** $o_p.\Delta < p.\Delta - df$ **then**
18  $o_p.\Delta = p.\Delta - df$;
19  **else**
20  $\mathcal{O} = \mathcal{O} \cup \{[p.SID_2, p.SID_2.AD, p.\Delta - df, level]\}$;
21  $S_{checked} = S_{checked} \cup \{p.SID_2\}$;
22  $\mathcal{P} = \mathcal{P} \setminus \{p\}$;
  /* expand current level as in lines 7-15 of Algorithm 1 */
  /* all nodes not in $LocalCell$ go into $\mathcal{P}$: $[o.SID, n.SID, o.\Delta, level + 1]$ */
23  $\mathcal{O} = \mathcal{O} \setminus \{o \in \mathcal{O} | o.AD < o.\Delta\}$;
24  $S_{marked} = S_{marked} \cup S_{checked}$;
25  $level = level + 1$;
26 **if** $P = \emptyset$ **then**
27  **return** *get-line-segments*($\mathcal{O}$);
28 **else**
29  delegate $\varphi, \mathcal{O}, \mathcal{P}, S_{marked}, \mathcal{I}$ hierarchy upwards;
30  **return** $\perp$;

**Algorithm 2**: Distributed threshold propagation and region detection algorithm

are not in the local cell are collected in the set $\mathcal{P}$, which is later propagated upwards in the network hierarchy and the next higher leader node will attempt to determine the threshold for these nodes.

If $\mathcal{P}$ in the output of Algorithm 2 is empty, i.e., $\mathcal{P} = \emptyset$, propagation terminates and anomaly regions can be detected on this level of the network hierarchy. Otherwise, $\varphi$, $\mathcal{O}$, $\mathcal{P}$, $S_{marked}$, and $\mathcal{I}$ of the current leader node are sent upwards to the leader node of the next higher level. There, the incoming data sets from all sub-cells are merged (line 4). Then, all nodes that have been collected in $\mathcal{P}$ on lower levels are considered for insertion into $\mathcal{O}$. It is possible that the $AD$ value of a node $p \in \mathcal{P}$ is not known to the leader node,

because either $p$ is in a different cell on this level (line 14), or the sub-cell containing $p$ did not send any data upwards. In the latter case, information about $p$ is requested from the corresponding sub-cell (line 12). Generally, nodes in $\mathcal{P}$ are only considered for insertion into $\mathcal{O}$ if they have not been previously considered, i.e., if they are not in $S_{marked}$. In the distributed algorithm, this property results in a feature we call "neighborhood preserving". It means that if a node has been checked by a leader node on a lower level already, and is thus in $S_{marked}$, it will not be checked again at higher levels, even if this node appears in $P$ with a lower level than in $S_{marked}$. This way, decisions made by sub-cells, i.e., the closer neighborhood of this node, about this node are not overwritten at higher levels. Insertion of nodes from $\mathcal{P}$ into $\mathcal{O}$ is similar to what happens in the centralized approach (lines 15–22 in Algorithm 2). As we stored the potential level for each node in $\mathcal{P}$, the nodes can be inserted at the appropriate level in $\mathcal{O}$. To propagate the threshold from nodes that have been newly inserted into $\mathcal{O}$ from $\mathcal{P}$, all nodes in the current level of $\mathcal{O}$ are checked again (comment below line 22).

For approximating the costs of in-network region detection, we have to introduce some more cost factors:

- $L$: average number of hierarchy levels involved in region detection
- $m_{lR}$: average number of nodes over all levels where (parts of) anomaly regions are handled
- $m_{aR}$: average number of anomalies handled over all levels
- $m_{fR}$: average number of anomaly regions detected and finalized over all levels – information about these regions is only forwarded following the multi-hop protocol
- $size_R$: average size of anomaly regions in bytes
- $size_{\{\mathcal{O},\mathcal{P},\mathcal{I}\}}$: average size of information needed to propagate regions upwards in the network hierarchy

This way, the number of regions is modeled by $m_{aR}, m_{lR}$ and $m_{fR}$. The size of regions is modeled by $m_{aR}, m_{lR}$ and $L$. Further, we assume that the anomaly regions are distributed uniformly over all cells. Based on these assumptions and the algorithm described above, we obtain:

$$
\begin{aligned}
C_{region} \quad = \quad & \underbrace{r_m \cdot c_{cpu}(anomalies) \cdot m}_{\text{anomalies at sources}} + \underbrace{r_a \cdot 2 \cdot (c_{msg} + 2 \cdot c_{byte}) \cdot (m - m_l)}_{\text{send and receive anomalies}} \\
& + \underbrace{r_a \cdot L \cdot (c_{cpu}(detect) \cdot m_{aR} + 2 \cdot (c_{msg} + size_{\{\mathcal{O},\mathcal{P},\mathcal{I}\}} \cdot c_{byte}) \cdot m_{lR})}_{\text{update and propagate regions}} \\
& + \underbrace{r_a \cdot (h \cdot (c_{msg} + size_R \cdot c_{byte}) + (h-1) \cdot (c_{msg} + size_R \cdot c_{byte})) \cdot m_{fR}}_{\text{forward finalized regions}}
\end{aligned}
$$

In this formula, we assume anomaly detection is done at the sources. If this is replaced by leader node-based detection, the $r_a$ in the first line must simply be replaced by $r_m$ (each measurement is sent, not only anomalies). $c_{cpu}(anomalies)$ corresponds to the CPU costs of the chosen method. Note that all listed cost formulas are worst case approximations, as

| measurement | time | energy |
|---|---|---|
| compute average of 10 values | $52.3\mu s$ | $0.272\mu J$ |
| compute average of 100 values | $245\mu s$ | $1.274\mu J$ |
| single addition | $2\mu s$ | $0.010\mu J$ |
| single division | $27\mu s$ | $0.140\mu J$ |
| single multiplication | $16.2\mu s$ | $0.08\mu J$ |
| sending 1 byte | $4.85ms(2.33 - 6.95ms)$ | $240.19\mu J(121 - 361\mu J)$ |
| sending 10 bytes | $4.9ms(2.8 - 7.4ms)$ | $252.93\mu J(146 - 385\mu J)$ |

Table 1: Average energy consumption measured on real sensors

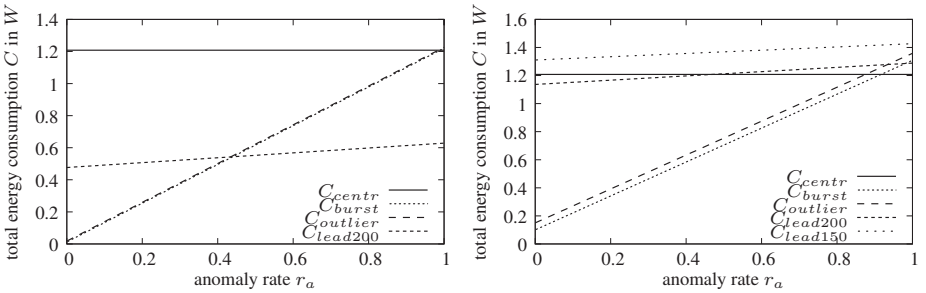
we use average values etc. Nevertheless, they are suited for analytically evaluating in-network processing by comparing the costs of each option. This is done in Section 6.

# 6 Evaluation

Due to space limitations, we focus the evaluation in this section on the distributed version of our approach. The centralized region detection approach has been evaluated in [FG08] and we showed the feasibility of the threshold propagation in Section 4. In the following, we present an analytical evaluation of the in-network processing options introduced in Section 5. For this, we instantiate the proposed cost formulas with values measured on real sensors and vary several cost factors. The purpose of this evaluation is *(i)* to identify the sensitive factors that have most influence on the actual choice, and *(ii)* to determine the benefits we get from in-network processing and in which situations. We expect the in-network methods being less energy consuming than the central approach up to a certain rate of anomalies $r_a$. The detection of anomalies on sources should perform best from this point of view, followed by the methods using leader nodes and hierarchy-based region detection.

We measured some typical Tmote Sky sensor nodes running TinyOS-1.x (16 bit microcontroller unit (MCU) MSP430F1611, 4 MHz clock rate, IEEE 802.15.4 compatible CC2420 transceiver with 250kBit/s). The MCU works on 16 bit integers, divisions are processed in software. For the sending operations of the transceiver we used maximal output power (+0 dBm). We assumed a battery voltage of 2.6 V and neglected any fluctuations that may occur in reality. All processing was done using the standard packet format of TinyOS-1.x, which means that for transmitting 1 byte raw data there are 12 bytes sent, due to headers, checksums etc. This corresponds to a raw sending time of about 0.384ms. Consequently, with 10 bytes raw data there are 21 bytes sent (about 0.672ms). Table 1 summarizes the most important results of the tests.

Roughly speaking, local processing is about 1000 times cheaper than communication. But, energy consumption of MCU operations is much more deterministic than communication in wireless networks. More complex routing protocols influence processing times and energy consumption. In our experiments, we did not apply such sophisticated protocols. Moreover, they would result in an overhead for both, processing and transmitting. The fluctuations observed for sending messages are due to the used CSMA protocol for radio transmissions, which uses random backup times among other things. Using TDMA this




(a) $w = 600$ (b) $w = 5000$

Figure 6: Varying anomaly rate $r_a$ ($m = 1000, m_l = 200(150), h = 3, r_m = 1$)

would not apply – but in turn there would be more effort on synchronization etc. We also neglected situations of high load in the network, which could result in transmission delays as well. Further, we did not consider switching between active and idle modes and techniques for optimizing energy consumption in this case (e.g., by abstaining from the switch process in certain situations). Summarizing, we measured in a general but practically meaningful environment, which allows us to identify meaningful differences between the in-network options.

Based on these observations, we can instantiate the formulas from Section 5 with concrete values. For this, we derived the average values from Table 1. Interestingly, we observed that the number of sensors $m$ and the rate of measurements $r_m$ have no influence on the decision of in-network processing. Of course, they influence the total energy consumption, but all methods scale equally with them. The most influential factor clearly is $r_a$. This is illustrated in Figure 6. We show the costs in a $m = 1000$ sensor network, with a hierarchy depth of $h = 3$ and one measurement per second ($r_m = 1$). The costs of the central approach are $C_{centr}$ and those of the anomaly detection on source level are $C_{burst}$ and $C_{outlier}$ respectively. The anomaly detection method using leader nodes is referred to by $C_{lead}$. Figure 6(a) shows that message costs outweigh processing costs significantly. Only with highest anomaly rates the in-network costs are above the central costs. Neighborhood-based anomaly detection on leader nodes is cheaper for all rates. This is due to the implicit aggregation of sensor messages on the lowest level of the hierarchy.

The differences in Figure 6(b) show that all methods scale with the window size $w$, which is the size of the sliding window on which anomaly detection is done. Only the costs of the method on leader nodes are significantly affected by $w$. With $m_l = 200$ leader nodes the costs of the in-network method are lower than those of the central approach up to an anomaly rate $r_a = 0.5$ (which is still a very high rate). For larger neighborhoods ($m_l = 150$) and high values of $w$, the costs are higher even if there is no anomaly detected at all.

This indicates that another sensitive factor is the ratio of sensors to leader nodes, i.e., the size of the neighborhoods. To illustrate this, we vary this ratio in Figure 7. We used an anomaly rate $r_a = 0.1$. Further, we again show the effect of the window size $w$. The figure reveals that only for large window sizes (short terms up to an hour are common in streaming systems) the central approach should be preferred if neighborhoods are rather

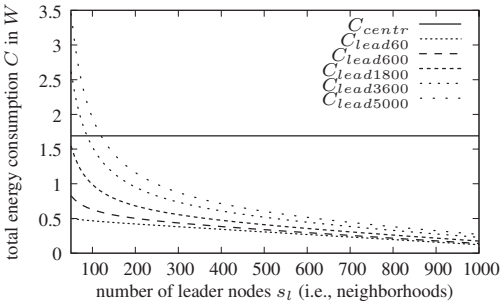
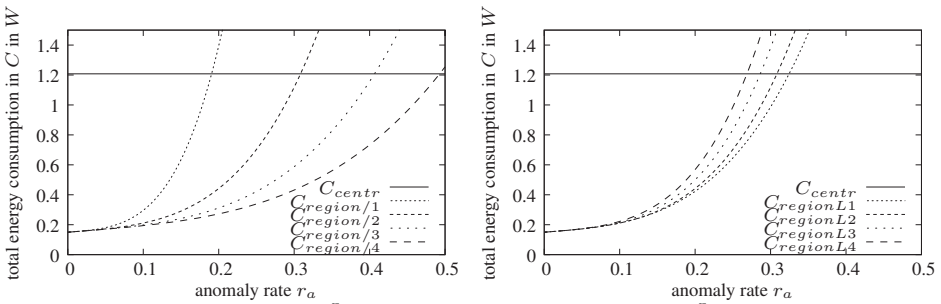



Figure 7: Varying number of leader nodes $m_l$ ($m = 1000, h = 4, r_m = 1, r_a = 0.1$)




(a) $L = 2, m_{aR} = m \cdot \frac{\frac{r_a}{r_m}}{\{1,2,3,4\}}$ (b) $m_{aR} = m \cdot \frac{\frac{r_a}{r_m}}{2}, L = \{1, 2, 3, 4\}$

Figure 8: In-network region detection ($m = 1000, m_l = 200, h = 3, r_m = 1, w = 5000$)

large. The plots for different $r_a$ with varying $m_l$ look similar, but in contrast they are close for small $m_l$ and differ more for large $m_l$ – but not as significantly as for different $w$.

Figure 7 also shows the rather weak effect of increasing the depth of the hierarchy, i.e., average hop count. In contrast to Figure 6 we used a hop count $h = 4$. Clearly, the energy consumption of the central approach rises, caused by the multi-hop protocol. The in-network methods rise as well, but significantly slower. With 200 leader nodes, the energy consumption of the method on leader nodes is at about $1.14$ for $r_a = 0.1$ in Figure 6(b). In Figure 7 the energy consumption for $w = 5000$ is at $1.17$, whereas the central approach increased about $0.5$ Watts.

Finally, we evaluated the in-network processing of anomaly region detection. As this is based on anomaly detection, we determined costs for both steps in conjunction (as we already did in the formula in Section 5). This only concerns the CPU costs for each method and influences performance of region detection negligibly. In our experiments we used the method for outlier detection on the sources exemplarily. It is rather difficult to identify suitable values for the used parameters $m_{aR}, m_{lR}$ and $m_{fR}$ without running tests on real data. However, the purpose of the cost comparison is to identify the sensitive parameters and to deduce the influence of region count and size. Thus, the effect of parameters is more important than their concrete values. Intuitively, all three depend on each other, and

all three depend on $r_a$ as well. We tested a wide range of concrete relations and concluded that the most influencing parameters are $m_{aR}$ and $L$. In Figure 8 we illustrate the effect of both. According to other tests we ran, a common average size of regions is about 6 sensors. Thus, we set $m_{lR} = m_{aR}/6$. The higher $L$, the larger are the regions and the smaller is $m_{fR}$ for constant $m_{lR}$. We chose to use $m_{fR} = \frac{m_{lR}}{L}$.

Figure 8 shows that, as expected, energy consumption for in-network region detection is much higher than for pure anomaly detection. Furthermore, it does not scale linearly and the point of "break even" concerning the central approach is earlier. However, in-network processing is still worthwhile for rather small (and thus, usual) anomaly rates. The more regions occur (larger $m_{aR}$), the smaller these anomaly rates and the larger the increase of energy consumption (see Figure 8(a)). The size of the regions (larger regions result in higher values for $L$) has a significant influence as well, but not as much as the number of regions (see Figure 8(b)).

Summarizing, in-network processing provides an excellent opportunity to reduce energy consumption, and thus to increase life time of sensors. Anomaly detection on sources should be delegated in principle. If leader nodes are used to identify neighborhood-based anomalies, the choice should depend on the crucial parameters like the window size $w$. As expected, region detection can often be better performed at the central instance. But, for low anomaly rates it is still a good option for saving energy. This effect decreases with increasing number of regions and their size – due to the hierarchy-based approach.

## 7 Conclusion

Detecting regions of anomalous phenomena in sensor networks is an interesting and challenging task. In this paper we presented an anomaly region detection approach that is aware of the obstacles in the sensor field. The presented algorithm allows us to derive anomaly regions with meaningful boundaries instead of regions described only by grouping the measurement points. We use the notion of a damping factor between pairs of sensors to represent spatial obstacles like buildings or mountains. With the help of the damping factor we are able to describe the spread of a phenomenon though the sensor field taking the damping effect of obstacles into account.

Transmitting data within a sensor network is one of the most energy consuming sensor operations. In order to minimize communication costs and consequently improve the network life time, we also presented an in-network processing strategy for our detection approach. We developed a formal cost model for both the intuitive centralized approach and the complete in-network computing. Finally, we also showed analytical and experimental results to evaluate our approaches.

## References

[BM07] Sabyasachi Basu and Martin Meckesheimer. Automatic outlier detection for time series: an application to sensor data. *Knowledge and Information Systems*, 11(2):137–

154, Feb 2007.

[cim] California Irrigation Management Information System (CIMIS). http://wwwcimis.water.ca.gov.

[DC03] Santpal Singh Dhillon and Krishnendu Chakrabarty. Sensor placement for effective coverage and surveillance in distributed sensor networks. In *Proc. of IEEE Wireless Communications and Networking Conference*, pages 1609–1614, 2003.

[DCXC05] Min Ding, Dechang Chen, Kai Xing, and Xiuzhen Cheng. Localized fault-tolerant event boundary detection in sensor networks. In *INFOCOM 2005: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 902–913, March 2005.

[FG08] Conny Franke and Michael Gertz. Detection and Exploration of Outlier Regions in Sensor Data Streams. In *SSTDM '08: International Workshop on Spatial and Spatiotemporal Data Mining in conjunction with ICDM 2008*, Dec 2008.

[GM04] Johannes Gehrke and Samuel Madden. Query Processing in Sensor Networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.

[HKT01] Jiawei Han, Micheline Kamber, and Anthony K. H. Tung. *Spatial Clustering Methods in Data Mining: A Survey*. Taylor and Francis, 2001.

[Int] MIT Computer Science and Artificial Intelligence Lab: Intel lab sensor data. http://db.csail.mit.edu/labdata/labdata.html.

[KEW02] Bhaskar Krishnamachari, Deborah Estrin, and Stephen B. Wicker. The Impact of Data Aggregation in Wireless Sensor Networks. In *ICDCSW '02*, pages 575–578, Washington, DC, USA, 2002. IEEE Computer Society.

[KGGK06] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 2–10, New York, NY, USA, 2006. ACM.

[KKPS08] Daniel Klan, Marcel Karnstedt, Christian Pölitz, and Kai-Uwe Sattler. Towards Burst Detection for Non-Stationary Stream Data. In *KDML 2008: Knowledge Discovery, Data Mining, and Machine Learning*, Oct 2008.

[KZ06] Ivana Kolingerová and Borut Zalik. Reconstructing domain boundaries within a given set of points, using Delaunay triangulation. *Comput. Geosci.*, 32(9):1310–1319, 2006.

[SPP+06] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB '06*, pages 187–198. VLDB Endowment, 2006.

[THH01] Anthony K. H. Tung, Jean Hou, and Jiawei Han. Spatial Clustering in the Presence of Obstacles. In *Proceedings of the 17th International Conference on Data Engineering*, pages 359–367, Washington, DC, USA, 2001. IEEE Computer Society.

[WCD+07] Weili Wu, Xiuzhen Cheng, Min Ding, Kai Xing, Fang Liu, and Ping Deng. Localized Outlying and Boundary Data Detection in Sensor Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1145–1157, 2007.

[YG02] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[ZPMZ04] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Spatial Queries in the Presence of Obstacles. In *EDBT '04: 9th International Conference on Extending Database Technology*, pages 366–384, Heraklion, Crete, Greece, March 2004.

[ZS03] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *KDD '03*, pages 336–345, New York, NY, USA, 2003.

[ZS06] Xin Zhang and Dennis Shasha. Better Burst Detection. In *ICDE '06*, pages 146–149, 2006.

# Priorisierte Verarbeitung von Datenstromelementen

Jonas Jacobi[1], André Bolles[1], Marco Grawunder[1], Daniela Nicklas[2] und H.-Jürgen Appelrath[1]

[1]Informationssysteme
Department für Informatik
Universität Oldenburg
D-26121 Oldenburg

[2]Datenbank- und Internettechnologien
Department für Informatik
Universität Oldenburg
D-26121 Oldenburg

[jonas.jacobi|andre.bolles|marco.grawunder|daniela.nicklas|appelrath]
@uni-oldenburg.de

**Abstract:** Datenstrommanagementsysteme, die effizient und skalierbar viele Anfragen gleichzeitig auf kontinuierlichen Daten auswerten können, sind eine vielversprechende Technologie für zukünftige Überwachungs- und Steuerungsanwendungen, z. B. bei dezentralen Energieanlagen. Dafür ist es jedoch notwendig, dass bestimmte Systemzustände (z. B. Warnungen oder Alarme) gegenüber dem Normalbetrieb bevorzugt behandelt werden. Im Gegensatz zu anderen Ansätzen, die solche Zustände durch eigene, vom Laufzeitsystem priorisierte Anfragen beschreiben, betrachten wir in diesem Beitrag priorisierte Datenstromelemente. Wir schaffen auf Basis einer logischen und physischen Operatoralgebra die formale Grundlage für die Priorisierung einzelner Datenstromelemente. Priorisierte Elemente profitieren darin doppelt. Zum einen können sie andere Elemente bspw. in Puffern „überholen“ und zum anderen können priorisierte Ergebnisse zustandsbehafteter Operatoren früher erzeugt werden, als bisherige Ansätze dies erlauben. Die Anfragesemantik bleibt dabei unverändert. Wir haben unseren Ansatz im Datenstrom-Framework ODYSSEUS implementiert und zeigen durch formale Abschätzungen und umfangreiche Messungen, wie durch die bevorzugte Behandlung priorisierter Elemente sowohl innerhalb von Anfrageoperatoren als auch beim Transport zwischen Anfrageoperatoren eine deutlich geringere Latenz erreicht werden kann. Davon können dann alle Anfragen profitieren, die solche priorisierten Elemente verarbeiten.

## 1 Einleitung

Wenn in Anwendungssystemen kontinuierlich große Datenmengen auftreten, ist es häufig aus Performanz- und Speicherplatzgründen nicht möglich, diese vor der Verarbeitung z. B. in einem Datenbankmanagementsystem (DBMS) zu speichern. Stattdessen müssen die Daten strombasiert verarbeitet werden, um unter Umständen erst später gegebenenfalls aggregierte oder ausgewählte Elemente zu archivieren. Auf eine solche Verarbeitung angewiesene Anwendungen sind zahlreich und nehmen in betriebswirtschaftlichen, technisch-ingenieurs-wissenschaftlichen und naturwissenschaftlichen Kontexten kontinuierlich zu. Charakteristisch ist, dass dynamische Systeme kontrolliert oder gesteuert werden sollen,

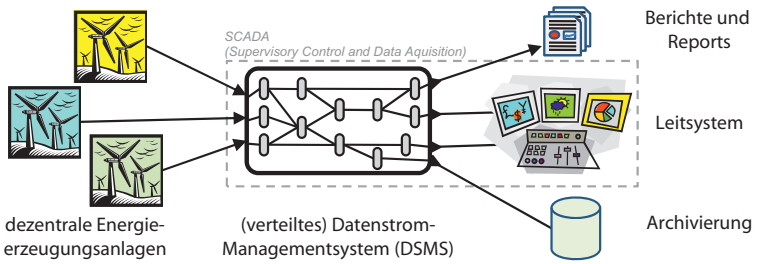



Abbildung 1: Szenario: Einsatz von Datenstrommanagementsystemen in Leitsystemen

die entweder durch Sensoren (wie z. B. technische Anlagen) oder Transaktionen (wie z. B. im Finanzsektor) beobachtet werden. Bisher werden solche Probleme in der Praxis meist mit Hard- oder Softwaresystemen gelöst, in welche die Verarbeitungslogik fest „einprogrammiert“ ist. Mit zunehmender Komplexität der Systeme und höherer Änderungsrate in den Anforderungen an die Systeme (welche eine bessere Wartbarkeit erfordert) werden diese Lösungen jedoch zunehmend unbrauchbar.

Seit einigen Jahren wird deswegen auf dem Gebiet der Datenstrommanagementsysteme (DSMS) geforscht, so dass mittlerweile erste kommerzielle Systeme verfügbar sind und eingesetzt werden. Analog zu einem DBMS ist die Kernidee dabei, das Informationsbedürfnis nicht durch ein Programm zu kodieren, sondern in einer Anfrage zu deklarieren, die dann von dem DSMS übersetzt, optimiert und ausgeführt wird. Im Unterschied zu einem DBMS sind die Anfragen jedoch meist kontinuierlich, das heißt sie werden registriert und über einen längeren Zeitraum hinweg auf den Eingangsdatenströmen verarbeitet. Das Ergebnis einer Anfrage auf Datenströmen ist in der Regel auch wieder ein Datenstrom.

Trotz der steigenden Bedeutung dieser Technologie sind viele Aspekte bisher ungelöst, was einem direkten kommerziellen Einsatz in einigen vielversprechenden Anwendungsdomänen, wie der Energiebranche, entgegen steht. In dieser Branche müssen vor allem durch die zunehmende Verbreitung regenerativer Energieerzeugung geographisch und organisatorisch immer mehr und weiter verteilte Anlagen und Stromnetze überwacht, gesteuert und reguliert werden. Der Einsatz von DSMS kann hier die Skalierbarkeit, Wartbarkeit und Flexibilität deutlich verbessern. Dazu muss zum einen ein performanter und hoch-ausfallsicherer Normalbetrieb sichergestellt werden, zum anderen wird für kritische Ausnahmesituationen eine zeitnahe (priorisierte) Verarbeitung benötigt.

Grundsätzlich gibt es zwei Ansätze für eine priorisierte Verarbeitung in Datenströmen: Zum einen können einzelne Anfragen im Datenstromsystem priorisiert werden, die dann vom System (bspw. durch einen Scheduler) bevorzugt bearbeitet werden, zum anderen können einzelne Elemente im Datenstrom mit bestimmten Eigenschaften (z. B. der Überschreitung eines Grenzwerts) priorisiert werden, wovon dann alle Anfragen profitieren, die diese Elemente verarbeiten. Unserem Wissen nach gibt es für letzteres bisher keine Ansätze im Datenstromkontext.

Ein typischer Anwendungsfall für eine elementbasierte Priorisierung findet sich z. B. bei

der Überwachung dezentraler Energieerzeugungsanlagen in einem Leitsystem (SCADA, Supervisory Control and Data Aquisition). Dabei müssen Statusmeldungen von Anlagen für die Darstellung in der Leitzentrale beispielsweise mit stromnetztopologischen Informationen angereichert werden. Außerdem werden auf den Meldungen – je nach Anlagentyp – weitere Analysen durchgeführt. Trifft nun eine Alarmmeldung ein, muss diese zeitnah zum Auftreten verarbeitet werden, und eine schnelle Reaktion auf Fehlerzustände ermöglichen. Da der Alarm prinzipiell in den gleichen Anfragen wie die restlichen Statusmeldungen verarbeitet wird, lässt sich keine Anfragepriorisierung durchführen, die die benötigten Ergebnisse schneller erzeugt. Insbesondere kann sich eine kritische Situation auch anhand von Grenzwertüberschreitungen eines Sensors ohne Alarmmeldungen identifizieren lassen.

Daher stellen wir in diesem Beitrag einen Ansatz zur Priorisierung einzelner Datenstromelemente vor, der es zum Ziel hat, die Latenz von Datenelementen zu verringern, die aufgrund ihres Inhaltes (Alarm, Warnung, Schwellenwert ...) besonders relevant sind. Wir beschreiben das Problem und formalisieren unseren Ansatz zur Priorisierung von Datenstromelementen in Kapitel 2. Kapitel 3 beschreibt eine physische Operatoralgebra, welche die priorisierte Verarbeitung von Datenstromelementen erlaubt. Kapitel 4 detailliert die von uns entwickelten operatorunabhängigen Techniken, mit denen priorisierte Elemente andere Elemente überholen können und die so eine optimierte Verarbeitung realisieren. Wir evaluieren den Ansatz in Kapitel 5, geben in Kapitel 6 eine Übersicht über verwandte Arbeiten und schließen in Kapitel 7 mit einem Fazit.

## 2 Priorisierte Datenstromelemente

Das Ziel dieser Arbeit ist es, besonders relevante Anfrageergebnisse mit möglichst geringer Latenz zu berechnen. Im Gegensatz zu bisherigen Arbeiten wird die Relevanz eines Ergebnisses nicht durch die Anfrage definiert (Anfragepriorisierung), sondern durch die Datenelemente, die zur Berechnung des Ergebnisses benötigt werden (Datenpriorisierung). Es gibt eine kleine Menge an Datenstromelementen innerhalb eines Datenstroms, die priorisiert verarbeitet werden muss. Ergebnisse, für die diese Elemente eine Rolle spielen, müssen also mit möglichst geringer Latenz berechnet werden. Jedes Element eines Datenstroms wird zu einem Zeitpunkt $t_{in}$ im DSMS registriert. Die Latenz eines Ergebnisses einer Operation ist die Dauer vom Zeitpunkt $t_{in}$ des ältesten Elements bzw. des Elements mit der höchsten Priorität, das zum Ergebnis beigetragen hat, bis zur Erzeugung des Ergebnisses. Bei einem Verbund zweier Alarmmeldungen gleicher Priorität wäre dies z. B. das ältere der beiden Eingabeelemente. Bei einem Verbund einer Alarmmeldung mit niedrig priorisierten Informationen ist für die Latenz nur das Eintreffen der Alarmmeldung relevant, da diese schnell verarbeitet werden soll. Natürlich sind auch andere Definitionen der Latenz eines Elementes (wie z. B. in [SCLP08]) denkbar und in anderen Anwendungskontexten sinnvoll.

Die Priorisierung ist nur für die Geschwindigkeit der Verarbeitung relevant und darf die Semantik von Anfragen nicht verändern. Zur Definition der Anfragesemantik stützen wir uns auf die logische Operatoralgebra aus [Krä07], und erweitern die ebenfalls darin definier-

te physische Algebra derart, dass ohne Veränderung der Anfragesemantik Priorisierung ausgedrückt werden kann und priorisierte Elemente andere Elemente bei der Verarbeitung überholen können.

Die logische Operatoralgebra aus [Krä07] definiert die Semantik der verschiedenen Operatoren über logischen Datenströmen. Wir definieren in Anlehnung einen logischen Datenstrom $S_T^l \in \mathbb{S}^l$ als Menge von Elementen $(e, t, n)$, wobei $e \in \Omega_T$ ein Tupel vom Typ $T$ ist und $n \in \mathbb{N}, n > 0$ die Anzahl des Tupels zum Zeitpunkt $t \in \mathbb{T}$ im Datenstrom definiert. Ein logischer Datenstrom $\{(a, 1, 2), (b, 2, 1)\}$ enthält also zum Zeitpunkt 1 zwei Kopien des Elements $a$ und zum Zeitpunkt 2 einmal das Element $b$. Die Definition ist unabhängig von einer Ordnung der Elemente im Datenstrom. Die Semantik eines logischen Operators wird über die vom Operator erzeugte Ausgabemenge definiert, je nach Operator in Abhängigkeit von einem bzw. zwei logischen Eingabeströmen. Als Beispiel sei die Definition des kartesischen Produkts $\times : \mathbb{S}_{T_1}^l \times \mathbb{S}_{T_2}^l \rightarrow S_{T_3}^l$ aus [Krä07] aufgeführt, wobei die Funktion $\circ : \Omega_{T_1} \times \Omega_{T_2} \rightarrow \Omega_{T_3}$ ein Ausgabetupel durch Konkatenation zweier Eingabetupel erzeugt und deren Vorkommen multipliziert:

$$\times(S_1, S_2) \coloneqq \{(\circ(e_1, e_2), t, n_1 \cdot n_2) | (e_1, t, n_1) \in S_1 \wedge (e_2, t, n_2) \in S_2\}$$

Physische Datenströme ($\mathbb{S}^p$) erlauben eine kompaktere Repräsentation der logischen Datenströme durch die Zusammenfassung zusammenhängender Zeitpunkte zu Intervallen und definieren eine Multimenge von Elementen $(e, [t_s, t_e))$, wobei $[t_s, t_e)$ die Gültigkeit des Elements im Datenstrom vom Startzeitpunkt $t_s$ bis zum Endzeitpunk $t_e$ repräsentiert. Damit unterscheidet sich diese Datenstromrepräsentation von der sonst üblichen Verwendung positiver und negativer Tupel zum Markieren der Gültigkeit (z. B. in [ABW06]). Die Elemente sind monoton steigend nach dem Startzeitstempel geordnet. Physische Datenströme lassen sich eindeutig auf logische Datenströme abbilden.

Zum Ausdrücken von Prioritäten führen wir eine spezielle Variante physischer Datenströme ein: Priorisierte physische Datenströme ($\mathbb{S}_{pr}^p$), oder kurz: priorisierte Datenströme.

Wir definieren schwach und stark priorisierte Datenströme um die Striktheit, mit welcher eine Ordnung zwischen Elementen innerhalb eines Datenstroms definiert wird, unterscheiden zu können. Ein stark priorisierter physischer Datenstrom $(S_{pr}^p, \leq)$ mit $S_{pr}^p \in \mathbb{S}_{pr}^p$ des Typs $T$ – im Folgenden kurz $S_{pr}^p$ – ist eine potentiell unendliche, geordnete Multimenge von Elementen $(e, [t_s, t_e), p)$, wobei wie gehabt $e \in \Omega_T$ ein Tupel vom Typ $T$, $[t_s, t_e)$ ein rechtsoffenes Zeitintervall in der diskreten Zeitdomäne $\mathbb{T}$ und $p \in \mathbb{N}_0$ zusätzlich die Priorität des Elements repräsentiert. Je wichtiger das Element, desto höher ist seine Priorität. Ein nicht priorisiertes Element wird durch $p = 0$ gekennzeichnet. Die Elemente eines solchen Datenstroms sind durch die Ordnungsrelation $\leq$ geordnet, die kompatibel zu der folgenden partiellen Ordnung sein muss:

$$<_{S_{pr}^p}^{strong} \coloneqq \{(a, b) \in S_{pr}^p \times S_{pr}^p | a.t_s < b.t_s \wedge a.p \geq b.p\} \quad (1)$$

Das bedeutet, dass Elemente der gleichen Priorität monoton steigend nach ihren Startzeitstempeln geordnet auftreten, jedoch Elemente höherer Priorität trotz eines jüngeren Zeitstempels vor Elementen mit geringerer Priorität eintreffen *können*. Wir weisen darauf hin, dass unter Elementen mit gleichen Startzeitstempeln keine Ordnung definiert wird, auch

nicht für Elemente mit unterschiedlichen Prioritäten. Dies liegt daran, dass Elemente trotz gleichen Zeitstempels sequentiell verarbeitet werden. Ansonsten müsste für alle Elemente eines Startzeitpunktes auf das Eintreffen eines jüngeren Elementes gewartet werden, um sicherzustellen, dass kein weiteres Element mit gleichem Startzeitstempel eintrifft, das in der Ordnung vor den bisherigen Elementen liegt. Das stünde im Widerspruch zu unserem Ziel, die Latenz zu optimieren.

Dass Elemente gleicher Priorität zeitlich geordnet sind, scheint einem intuitiven Verständnis von priorisierten Datenströmen zu entsprechen. Das Ziel dieser Arbeit ist aber nicht, die Ordnung der Elemente vorteilhaft für priorisierte Elemente zu gestalten, sondern ihre Latenz zu verbessern. Wir zeigen in Kapitel 3.1, dass beides nicht zusammenhängen muss, beziehungsweise sich unter bestimmten Umständen sogar widerspricht, und führen daher eine weitere Definition schwach priorisierter Datenströme ein.

Schwach priorisierte physische Datenströme garantieren keine Ordnung innerhalb priorisierter Elemente. Sie unterscheiden sich von einem stark priorisierten einzig dadurch, dass ihre Ordnung zum Folgenden kompatibel sein muss:

$$<_{S_{pr}^p}^{weak} := \{(a,b) \in S_{pr}^p \times S_{pr}^p | a.t_s < b.t_s \wedge b.p = 0\} \tag{2}$$

Ein physischer Strom kann mittels der Funktion $\varphi^{nonp \to p} : \mathbb{S}^p \to \mathbb{S}_{pr}^p$ in einen priorisierten physischen Strom gleichen Typs transformiert werden. Die zeitliche Ordnung wird beibehalten.

$$\varphi^{nonpr \to pr}(S_T^p) := \{(e,[t_s,t_e), priority(e)) \in \Omega_T \times (\mathbb{T} \times \mathbb{T}) \times \mathbb{N}_0 | (e,[t_s,t_e)) \in S_T^p\}$$

Die Funktion $priority : S_T^p \to N_0$ bestimmt dabei die Priorität eines Elements anhand seines Dateninhalts.

Die umgekehrte Transformation eines priorisierten in einen „normalen" physischen Datenstrom ($S^p$) erfolgt durch Abbilden der priorisierten Elemente auf ihr unpriorisiertes Gegenstück

$$(e,[t_s,t_e),p) \mapsto (e,[t_s,t_e))$$

und Herstellen der zeitlichen Ordnung auf dem Datenstrom nach

$$<_{S^p} := \{(a,b) \in S^p \times S^p | a.t_s < b.t_s\}.$$

Über diesen Zwischenschritt lässt sich auch ein priorisierter physischer Datenstrom in einen logischen Strom transformieren, über den die Anfragesemantiken eindeutig definiert sind.

# 3 Physische Algebra

In diesem Kapitel stellen wir einen Auszug aus der von uns entwickelten physischen Operatoralgebra vor, die eine bevorzugte Verarbeitung priorisierter Datenstromelemente

erlaubt. Dazu führen wir einen neuen physischen Operator ein, der die Elemente eines „normalen" physischen Datenstroms priorisiert und dazu – wie im vorigen Kapitel bei der Umwandlung in priorisierte Datenströme beschrieben – jedem Element eine Priorität zuordnet, die sich anhand einer zu übergebenen Funktion aus den Daten des Elements berechnet.

In der Definition priorisierter physischer Datenströme können Elemente „out-of-order" – d.h. in nicht zeitlicher Ordnung – eintreffen. Die physischen Operatoren müssen teilweise an dieses Verhalten angepasst werden, damit korrekte Anfrageergebnisse garantiert werden können.

Die zustandslosen, satzorientierten Operatoren (Selektion, Projektion, ...) funktionieren ohne Änderung auch auf priorisierten Datenströmen, da die Prioritäten einfach ignoriert werden können.

Anders hingegen verhält es sich mit den zustandsbehafteten, mengenorientierten Operatoren. Diese lassen sich in zwei Klassen unterscheiden: In solche, die sinnvoll mit priorisierten Elementen umgehen können und solchen, die für eine Verarbeitung die ursprüngliche Stromreihenfolge benötigen, für die eine Priorisierung einzelner Elemente also irrelevant ist. Zu letzteren gehören beispielsweise der Aggregations- sowie der Sortieroperator.

Die notwendigen Anpassungen zur Sicherstellung der korrekten Semantik der anderen Operatoren beziehen sich in erster Linie auf das Aufräumen des internen Zustandsspeichers, da dies bei den mengenorientierten Operatoren anhand der Zeitstempel auf Grundlage der zeitlichen Ordnung des Datenstroms geschieht, die bei einem priorisierten Strom nicht mehr garantiert ist. Ein Element im Zustandsspeicher eines mengenorientierten Operators kann genau dann verworfen werden, wenn garantiert ist, dass kein zukünftiges Element mehr ein überlappendes Zeitintervall besitzt. Dies wird aufgrund der Ordnung des Eingabestromes bisher daran erkannt, dass der Startzeitstempel eines eingehenden Elements größer als der Endzeitstempel des betreffenden Elements im Zustandsspeicher ist.

Neben den notwendigen Änderungen der Operatoren zur Sicherstellung der semantischen Korrektheit gibt es weitere, die zur Latenzverbesserung bei der Verarbeitung priorisierter Elemente beitragen.

Unsere mengenorientierten Operatoren benutzen sogenannte SweepAreas [DSTW02] als Zustandsspeicher. Der abstrakte Datentyp SweepArea wird mit drei Eingabeparametern parametrisiert: eine Ordnungsrelation $\leq$ sowie zwei binäre Prädikate $p_{query}$ und $p_{remove}$. Er stellt unter anderen folgende Methoden zur Verfügung, deren Verhalten von den Eingabeparametern beeinflusst wird:

**query(**element $s, j \in \{0, 1\}$**)** Liefert einen Iterator mit der Ordnung $\leq$ über alle Elemente $\hat{s}$ der SweepArea, für die das Prädikat $p_{query}$ mit $s$ und $\hat{s}$ als Parametern erfüllt ist. Der Parameter $j$ bestimmt dabei, ob an $s$ als erster und $\hat{s}$ als zweiter Parameter an $p_{query}$ übergeben wird, oder ob $\hat{s}$ der erste und $s$ der zweite Parameter sein soll.

**purgeElements(**element $s, j \in \{0, 1\}$**)** Entfernt alle Elemente $\hat{s}$ aus der SweepArea für die $p_{remove}$ mit $s$ und $\hat{s}$ als Parametern erfüllt ist. Der Parameter $j$ bestimmt wie bei **query** die Reihenfolge, in der $s$ und $\hat{s}$ übergeben werden.

Die folgende Definition eines Theta-Join-Operators benutzt diese Datenstruktur als internen Zustandsspeicher.

### 3.1 Kartesisches Produkt/Theta-Join-Operator

Beispielhaft wird im Folgenden ein Kartesisches Produkt/Theta-Join-Operator definiert, der korrekt mit priorisierten Datenströmen umgeht und bei der Erzeugung von Verbundelementen solche mit höherer Priorität unabhängig von den Startzeitstempeln anderer Elemente als erstes herausschreibt, so dass diese andere Elemente überholen.

Die möglichen Verbundpartner eines Elements werden mittels der `query`-Methode aus einer SweepArea ermittelt. Dafür wird folgendes Query-Prädikat verwandt, welches neben der Gültigkeit des Verbundprädikates $theta$ noch den Schnitt der Gültigkeitsintervalle einbezieht:

$$p^{\theta}_{query}(s, \hat{s}) \coloneqq \begin{cases} wahr & \text{wenn } \theta(e, \hat{e}) \wedge [t_s, t_e) \cap [\hat{t_s}, \hat{t_e}) \neq \varnothing, \\ falsch & \text{sonst.} \end{cases}$$

Elemente werden aus der SweepArea entfernt, wenn aufgrund der Ordnung der Datenströme in Zukunft keine Verbundpartner mehr eintreffen können. Um dies zu erkennen, können nur unpriorisierte Elemente ($p = 0$) dafür verwendet werden, da bei höher priorisierten Elementen nicht sichergestellt ist, dass diese nicht andere Elemente überholt haben und damit später weitere Verbundpartner eintreffen können.

$$p_{remove}(s, \hat{s}) \coloneqq \begin{cases} wahr & \text{wenn } t_s > \hat{t_e} \wedge p = 0 \\ falsch & \text{sonst.} \end{cases}$$

Die Ergebniselemente des Joins werden in einer nach der Ordnung $<^{p,t_s}$ aufsteigend sortierte Warteschlange eingefügt, wobei $<^{p,t_s}$ wie folgt definiert ist:

$$<^{p,t_s} \coloneqq \{(a, b) | a.p > b.p \vee (a.p = b.p \wedge a.t_s < b.t_s)\}$$

Damit werden die Ergebnisse nach absteigender Priorität ausgegeben und höher priorisierte Elemente können die weniger wichtigen in der Warteschlange überholen.

Die Priorität eines Ergebnisses der Verbundoperation berechnet sich aus den Eingabeelementen anhand einer zu übergebenen Funktion. Davon ausgehend, dass wichtige Informationen auch nach dem Verbund mit anderen Informationen wichtig sind, verwenden wir in unserem Anwendungskontext dafür das Maximum der Prioritäten der beiden Eingabeelemente.

Wir haben den in [Krä07] vorgestellten Algorithmus, der die Ripple-Join-Technik [HH99] auf push-basierte Verarbeitung überträgt, für die Verarbeitung priorisierter Datenströme angepasst.

Der in Algorithmus 1 definierte Join-Algorithmus verwendet eine Warteschlange zur Zwischenspeicherung für Verbundergebnisse und eine SweepArea für jeden Eingabedaten-

**Algorithmus 1** : Kartesisches Produkt ($\times$)/Theta-Join ($\bowtie$)

**Eingabe** : Priorisierte physische Datenströme $S_{in_0}, S_{in_1}$;
Funktion zur Berechnung von Prioritäten von Verbundelementen `newPriority`;

**Ausgabe** : Priorisierter physischer Datenstrom $S_{out} \leftarrow \varnothing$;

**Daten** : Seien $SA_0, SA_1$ leere SweepAreas($\leq^{t_e}, p^{\theta}_{query}, p_{remove}$);
$ts[0], ts[1], min_{ts} \in T \cup \perp; ts[0] \leftarrow \perp, ts[1] \leftarrow \perp, min_{ts} \leftarrow \perp$;
Sei Q eine nach $<_Q^{p,t_s}$ aufsteigend sortierte leere Prioritätswarteschlange;
Datenelement $e'$;
$j, k \in \{0, 1\}$;

`/* j indiziert in jeder Iteration den Strom, aus dem in Zeile 1 das Element s eingetroffen ist. k referenziert den jeweils anderen Eingabestrom. */`

1 **foreach** $s \coloneqq (e, [t_s, t_e), p) \hookleftarrow S_{in_j}$ **do**
2 $SA_k$.purgeElements(*s,j*);
3 $SA_j$.insert(*s*);
4 Iterator $qualifies \leftarrow SA_k$.query(*s,j*);
5 **while** $qualifies$.hasNext() **do**
6 Element$(\hat{e}, [\hat{t_s}, \hat{t_e}), \hat{p}) \leftarrow qualifies$.next();
7 **if** $j = 0$ **then**
8 $e' = e \circ \hat{e}$
9 **else**
10 $e' = \hat{e} \circ e$
11 **end**
12 $Q$.insert($e', [t_s, t_e) \cap [\hat{t_s}, \hat{t_e})$, newPriority($(e, [t_s, t_e), p), (\hat{e}, [\hat{t_s}, \hat{t_e}), \hat{p})$));
13 **end**
14 $ts[j] \leftarrow t_s$;
15 **if** *p = 0* **then**
16 $min_{ts} \leftarrow \min(ts[0], ts[1])$;
17 **end**
18 TRANSFER($Q, min_{ts}, S_{out}$);
19 **end**
20 **while** $\neg Q$.isEmpty() **do**
21 $Q$.extractMin() $\hookrightarrow S_{out}$;
22 **end**

strom. Jedes eingehende Element eines Datenstroms wird zunächst in die eigene SweepArea geschrieben und, falls es die Priorität 0 hat, dafür verwendet den Zustand der SweepArea des anderen Datenstroms aufzuräumen (Zeilen 3 und 4). Danach werden mittels der `query`-Methode alle passenden Verbundpartner aus der entsprechenden SweepArea ermittelt und für jeden Partner ein Verbundresultat berechnet, das zunächst in die Ausgabewarteschlange eingefügt wird (Zeile 12). Insbesondere wird die Priorität des neuen Elementes aus den beiden Verbundpartnern berechnet.

Die `TRANSFER`-Funktion (siehe Algorithmus 2) sorgt dafür, dass die Elemente der Warteschlange, für die es möglich ist, in den Ausgabestrom geschrieben werden. Dazu wird über den Zeitstempel $min_{t_s}$ bestimmt, ob noch Elemente auftauchen können, die aufgrund der zeitlichen Ordnung des Datenstroms vor vorhandenen Elementen herausgeschrieben werden müssen. Die von uns definierte schwache Ordnung priorisierter Datenströme (siehe Gleichung 2) verlangt, dass zur Berechnung von $min_{t_s}$ nur nicht priorisierte Elemente verwendet werden, aber erlaubt es priorisierte Ergebnisse schon früher als andere Elemente herauszuschreiben.

Wenn die Eingabeströme erschöpft sind, müssen noch alle übriggebliebenen Elemente aus der Warteschlange herausgeschrieben werden (Algorithmus 1, Zeilen $20 - 22$).

**Algorithmus 2** : TRANSFER-Funktion für schwach priorisierte Datenströme

```
   Eingabe :  Min-Prioritäts-Warteschlange Q;
              Zeitstempel min_ts; Ausgabestrom S_out;
 1 Iterator it ← Q.iterator();
 2 while it.hasNext() do
 3 |   Element (e, [t_s, t_e), p) ← it.next();
 4 |   if p > 0 ∨ t_e ≤ min_ts then
 5 |   |   it.remove();
 6 |   |   S_out ↩ (e, [t_s, t_e), p);
 7 |   else
 8 |   |   break;
 9 |   end
10 end
```

Wir haben uns bei der Realisierung für die Verwendung der schwachen Ordnung entschieden, da die starke Ordnung einige eklatante Nachteile hätte.

Die `TRANSFER`-Funktion für stark priorisierte Datenströme müsste sicherstellen, dass alle priorisierten Elemente gegenüber gleich oder höher priorisierten Elementen zeitlich geordnet sind. Das bedeutet, priorisierte Elemente würden erst dann herausgeschrieben, wenn ein nicht priorisiertes Element mit einem höheren Startzeitstempel in die Warteschlange eingefügt wird. Dies geschieht zu einem Zeitpunkt, zu dem das Element auch ohne jegliche Priorisierung herausgeschrieben worden wäre. Es bliebe einzig der Gewinn durch das Überholen in der Warteschlange selbst. Überholen von Elementen vor dem Operator hätte keinen Einfluss mehr auf die Latenz, da wieder auf das Eintreffen der überholten Elemente gewartet werden müsste. Damit hat ein priorisiertes Element dann zwar andere Elemente

überholt (evtl. mehr als in einem schwach priorisierten Datenstrom), die Latenz, mit der das Ergebnis bereitsteht, steigt aber, weil es erst viel später in den Ausgabestrom herausgeschrieben wird. Das heißt, in einem Plan würde nur noch der oberste Verbund- bzw. binäre, zustandsbehaftete Operator zu einem Gewinn bei der Latenz beitragen können, der zudem (je nach Verarbeitungskosten der darauf folgenden Operatoren) eher marginal ist. Aus diesem Grund haben wir uns zur Verwendung der schwachen Ordnung $<^{weak}_{S^p_{pr}}$ für priorisierte Datenströme entschieden, die diesen Nachteil nicht besitzt.

Im Folgenden soll nun skizzenhaft gezeigt werden, dass der von uns vorgestellte Join-Algorithmus auch die gewünschten Ergebnisse erzeugt und insbesondere semantisch äquivalent zum Algebraoperator in [Krä07] ist.

### 3.1.1 Semantische Korrektheit des Verbundoperators

Wenn ein Element eingelesen wird, gibt es zwei Möglichkeiten. Wenn Verbundpartner bereits eingelesen wurden, finden sie sich in der komplementären SweepArea und werden über die `query`-Methode ermittelt. Wenn Verbundpartner erst später im anderen Datenstrom eintreffen, ist das Element seinerseits aber in der SweepArea und wird selbst über die `query`-Methode zurückgeliefert. Dass die `query`-Methode nur korrekte Verbundpartner zurückgibt, lässt sich leicht über das Prädikat $p_{query}$ nachvollziehen.

Damit der Algorithmus also korrekte Ergebnisse liefert, muss nur noch sichergestellt sein, dass kein Element aus einer SweepArea entfernt wird, für das später noch weitere Verbundpartner eintreffen können. Zu diesem Zweck soll das Remove-Prädikat genauer untersucht werden. Seien dazu $in_j$ das zuletzt gelesene Element eines Eingabestroms $S_{in_j}$ und $SA_i$ mit $i \neq j$ die Menge der Elemente der SweepArea des anderen Eingabestroms. Sollte die Priorität von $in_j$ größer als 0 sein, gilt automatisch $\forall e \in SA_i : \neg p_{remove}(in_j, e)$.

Es bleibt also zu zeigen:

$$in.p = 0 \Rightarrow \nexists e \in SA_i : p_{remove}(in_j, e) \wedge \exists s \in S_{in_j} : \neg(s <^{weak}_{S_{in_j}} in_j) \wedge p_{query}(s, e).$$

Angenommen es gäbe solch ein Element $e$, dann ist $e.t_e < in_j.t_s$ (wegen $p_{remove}(in_j, e)$). Außerdem muss es ein Element $s \in S_{in_j}$ mit $\neg(s <^{weak}_{S_{in_j}} in_j) \wedge p_{query}(s, e)$ geben. Das bedeutet

$$\begin{aligned} & \exists s \in S_{in_j} : s.t_s \geq in_j.t_s \wedge p_{query}(s, e) \\ \Rightarrow \quad & \exists s \in S_{in_j} : s.t_s \geq in_j.t_s \wedge [s.t_s, s.t_e) \cap [e.t_s, e.t_e) \neq \varnothing \\ \Rightarrow \quad & \exists s \in S_{in_j} : s.t_s \geq in_j.t_s \wedge s.t_s < e.t_e. \end{aligned}$$

Dies steht im Widerspruch zu $e.t_e < in_j.t_s$, womit gezeigt wurde, dass kein Element aus einer SweepArea entfernt wird, wenn später noch weitere Verbundpartner eintreffen können.

Es bleibt noch zu zeigen, dass der Ausgabedatenstrom auch der korrekten Ordnung unterliegt.

### 3.1.2 Korrektheit der Ordnung des Ausgabedatenstroms

Der von einem Operator erzeugte Ausgabedatenstrom $S_{out}$ mit der Ordnung $<$ muss bzgl. $<^{weak}_{S_{out}}$ sortiert sein. Im Folgenden zeigen wir, dass der von Algorithmus 1 erzeugte Datenstrom diese Bedingung erfüllt.

Es sei $t_{s,min,k} = min(t_{s,1}, t_{s,2})$ mit $t_{s,j} = max(\{t_s | (e, [t_s, t_e), 0) \in S_{in_j,k}\})$ für $j \in \{0, 1\}$ der minimale Startzeitstempel der bis zum $k$-ten Durchlauf eingelesenen nicht-priorisierten Elemente der Eingabedatenströme des Joins. Sei weiterhin $S_{out,k}$ die Menge der nach dem $k$-ten Durchlauf durch die Funktion `TRANSFER` in den Ausgabedatenstrom geschriebenen Elemente.

Es gilt $<^{weak}_{S_{out,k}} \subseteq <$, denn:

$$\begin{array}{rll} & \forall (a,b) \in <^{weak}_{S_{out,k}}: & a.t_s < b.t_s \wedge b.p = 0 \\ \overset{\text{wgn. } b.p=0}{\Longrightarrow} & \forall (a,b) \in <^{weak}_{S_{out,k}}: & a.p > b.p \vee (a.p = b.p \wedge a.t_s < b.t_s) \\ \Longrightarrow & \forall (a,b) \in <^{weak}_{S_{out,k}}: & (a,b) \in <^{p,t_s}_{S_{out,k}} \end{array} \tag{3}$$

Es bleibt zu zeigen, dass $<^{weak}_{S_{out}}$ für $S_{out} = \bigcup_{k=1}^{\infty} S_{out,k}$ gilt, also nicht nur die im $k$-ten Durchlauf erzeugte Ausgabe der Ordnung unterliegt, sondern der gesamte Ausgabestrom. Auch diese Bedingung ist erfüllt, da die Funktion `TRANSFER` im $k$-ten Durchlauf nur jeweils alle Elemente $s_Q$ bzgl. $<^{p,t_s}_{Q}$ sortiert aus der Warteschlange $Q$ in den Ausgabedatenstrom $S_{out}$ schreibt, für die $s_Q.t_s < t_{s,min} \vee s_Q.p > 0$. Damit ist $max(\{t_s | (e, [t_s, t_e), 0) \in S_{out,k}\}) \leq t_{s,min,k}$. Das bedeutet, dass danach keine weiteren Elemente mehr in die Queue eingefügt werden können, die nach $<^{weak}_{S_{out}}$ vor einem der herausgeschriebenen Elemente liegen sollten.

Damit wurde gezeigt, dass der Ausgabedatenstrom, welcher von Algorithmus 1 erzeugt wird, bzgl. $<^{weak}_{S_{out}}$ sortiert ist.

### 3.1.3 Abschätzung zur Verbesserung der Latenzen

Um eine Abschätzung darüber vornehmen zu können, wie sich der Algorithmus 1 auf die Verbesserung der Latenzen auswirken kann, müssen zunächst einige Randbedingungen festgelegt werden. In einem allgemeinen Datenstromanfrageplan spielen sehr viele Parameter, wie Scheduling-Strategien, Zusammensetzung des Anfrageplans oder auch die zeitliche Reihenfolge der Elemente im Datenstrom eine Rolle. Wir betrachten bei dieser Abschätzung daher einen Queryplan mit nur einem „priorisierten Join“, der sich vorteilhaft auf die Latenzen der Elemente auswirkt und die Selektivität $\sigma$ hat. Weiterhin seien $w_j$ die Fensterbreiten auf Datenströmen $S_j$ sowie $f_j$ die Frequenzen, mit denen Elemente in den Datenströmen $S_j$ auftauchen, wobei $j \in \{0, 1\}$ gilt. Wir gehen davon aus, dass die Fensterbreiten vor dem Eintreffen der Elemente im hier betrachteten Join nicht verändert werden, bspw. durch einen vorhergehenden Join. Weiterhin sei $t_{plan}$ die nach dem Join verbleibende Verarbeitungszeit eines einzelnen Elementes im Anfrageplan. Der Einfachheit halber gehen wir hier von einer Single-Thread-Scheduling-Strategie aus, so dass die Verarbeitungszeit des verbleibenden Anfrageplans für alle Elemente proportional zu $t_{plan}$ ist. Es

ist leicht einzusehen, dass sich für die am höchsten priorisierten Elemente im schlechtesten Fall kein Vorteil durch das Überholen nicht priorisierter Elemente ergibt, da in diesem Fall ein hoch priorisiertes Element zwar eher im Eingabedatenstrom eines Joins auftaucht, jedoch noch keine Verbundpartner vorhanden sind und alle bereits überholten Elemente vorher ihre Verbundpartner vorfinden. Für den besten Fall, dass bereits alle Verbundpartner des überholenden Elementes vorhanden sind, lässt sich jedoch eine Abschätzung vornehmen, wie groß der Gewinn ist, der durch das Überholen im Join erreicht werden kann.

Hierzu sei o. B. d. A. angenommen, dass ein priorisiertes Element $e_p$, welches bereits $x$ niedriger priorisierte Elemente überholt hat, im linken Eingabedatenstrom des Joins auftaucht. Wenn bereits alle Verbundpartner für dieses Element im rechten Eingabedatenstrom vorhanden sind, dann ist die Ausgabewarteschlange des Join fast leer. In ihm befinden sich noch genau die Verbundelemente, die durch das letzte Element des linken Eingabedatenstrom erzielt wurden. Dies sind im Durchschnitt $\sigma \cdot w_0 \cdot f_1$ Verbundelemente. Diese Ergebnisse können nun durch das neue priorisierte Element überholt werden.

Weiterhin können die Verbundelemente von $e_p$ sofort herausgeschrieben werden, auch wenn ihr Startzeitstempel größer als $min_{t_s}$ ist. Sie müssen nicht darauf warten, dass Elemente eintreffen, die $min_{t_s}$ über ihren Startzeitstempel hinaus erhöhen. Damit überholt jedes Verbundelemente von $e_p$ im Schnitt zusätzliche $\sigma \cdot \sum_{n=1}^{w_0 \cdot f_0} (w_1 + \frac{n}{f_0}) \cdot f_1 \cdot \frac{\cdot w_0}{w_0 + w_1}$ Elemente. Außerdem muss nicht auf das Eintreffen der entsprechenden anderen Elemente nach $e_p$ gewartet werden, so dass im Schnitt noch $\frac{w_0^2}{2 \cdot (w_0 + w_1)}$ gewonnen wird (hierbei wird der Verständlichkeit halber davon ausgegangen, dass die Fenster in Systemzeit und nicht in einer davon unabhängigen Applikationszeit definiert sind).

Geht man jetzt davon aus, dass jedes dieser Element noch eine Verarbeitungszeit $t_{plan}$ im verbleibenden Anfrageplan verursacht und die Verarbeitungszeit eines Elementes im Join durchschnittlich $t_{join}$ ist, dann ergibt sich ein durchschnittlicher Gewinn für die Latenzen von Verbundelementen von $e_p$ zu

$$t_{win} = x \cdot t_{join} + \frac{w_0^2}{2 \cdot (w_0 + w_1)} + \sigma \cdot (\sum_{n=1}^{w_0 \cdot f_0} (w_1 + \frac{n}{f_0}) \cdot \frac{w_0}{w_0 + w_1} + w_0) \cdot f_1 \cdot t_{plan}$$

## 4 Überholen von Elementen im Datenstrom

Mit Hilfe der Operatoren lässt sich bereits eine Verbesserung der Latenz eines priorisierten Datenstromelements erreichen. Neben diesem intraoperatorbasierten Ansatz bietet es sich ebenfalls an, ein Überholen auch zwischen den Operatoren (Inter-Operator) zu ermöglichen, was zum einen weiteres Optimierungspotential bietet und zum anderen auch Pläne ohne spezielle (binäre) Operatoren vom Überholen profitieren lässt.

Wir gehen in unserer Arbeit von einem push-basierten Ansatz aus, d.h. Operatoren (Quellen) geben ihre produzierten Daten an Nachfolger (allgemein Senken) weiter. Für die Quelle ist es prinzipiell irrelevant, was die Senke mit den Daten macht. Insbesondere spielt es keine Rolle, ob der Nachfolger tatsächlich ein Operator oder ein zwischenspeichernder

Puffer ist. In der Abbildung 2 ist beispielhaft der Aufbau eines Operatorplans dargestellt.

Am unteren Ende befinden sich die Quellen, die Daten für die Senken produzieren. Zwischen Quelle und Senke kann dabei ein Puffer eingefügt werden. Ein Scheduler sorgt dafür, dass die untersten Quellen eines Planes und die Puffer ihre Daten weiterleiten. Wenn zwischen zwei Operatoren kein Puffer eingefügt ist, blockiert der unterste Operator so lange, bis der oberste Operator seine Daten geschrieben hat. Erst durch die Einführung von Puffern kann ein effektives Scheduling stattfinden. Im markierten Pfad in Abbildung 2 würde ein Tupel immer den kompletten Pfad nach dem Puffer durchlaufen, da es nicht zwischengespeichert werden kann, so dass keine Operatoren im Pfad parallel ausgeführt werden können. Für das Überholen im Datenstrom spielen also nur die Stellen eine Rolle, an denen sich Puffer zwischen zwei Operatoren befinden.

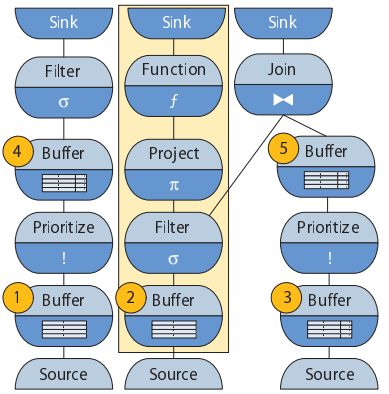



Abbildung 2: Aufbau eines Operatorplans mit Quellen und Senken

Grundsätzlich sind die in der Abbildung 3 dargestellten Ansätze für das Überholen im Datenstrom zwischen Operatoren möglich. Zu sehen sind jeweils zwei Operatoren und ein dazwischen geschalteter Puffer. In den Puffern befinden sich Datenstromelemente, die jeweils mit einer Priorität versehen sind. Ein Element mit Priorität 0 ist nicht priorisiert.

Im ersten, *direct mode* genannten Ansatz werden in den Puffern nur Elemente ohne Priorität gespeichert und Elemente mit einer Priorität direkt an die nachfolgende Senke weitergeleitet. Priorisierte Elemente werden also direkt im Kontext des selben Threads weiterverarbeitet, womit in diesem Fall der untere Operator blockiert. Aus diesem Grund sind solche Puffer nie direkt über den untersten Operatoren eines Planes einsetzbar. Ansonsten könnte es passieren, dass der Thread für den Quellenoperator so lange mit der weiteren Verarbeitung priorisierter Elemente beschäftigt ist, dass dieser eingehende Elemente „verpasst“. Außerdem besteht ein erhöhter Synchronisationsbedarf in den Operatoren, da die Verarbeitungsmethode nun gleichzeitig mit zwei Elementen aus dem selben Eingabestrom aufgerufen werden kann. Wird bspw. für die Verarbeitung ein Plan vertikal geteilt und ein Thread führt die Operatoren des unteren Teils aus und ein anderer die des oberen, kann es passieren, dass der untere Thread – durch die direct-Puffer – Elemente in obere Operatoren pusht, während diese gerade vom oberen Thread ausgeführt werden.

Der zweite Ansatz (*weak order mode*) fügt alle nicht priorisierten Elemente wie gehabt an das Ende des Puffers ein. Priorisierte Elemente werden ohne Berücksichtigung der Priorität an den Anfang des Puffer geschrieben. Dies hat den Vorteil, dass beim Einfügen eine Sortierung priorisierter Elemente vermieden werden kann, hat aber den Nachteil, dass unterschiedliche Prioritäten nicht unterschiedlich gewichtet werden.

Der letzte Ansatz (*strong order mode*) sorgt schließlich dafür, dass wenn mehrere priorisierte Elemente im Puffer vorhanden sind, diejenigen mit der höchsten Priorität auch am weitesten nach oben geschrieben werden, allerdings auf Kosten teurerer Einfügeoperatio-

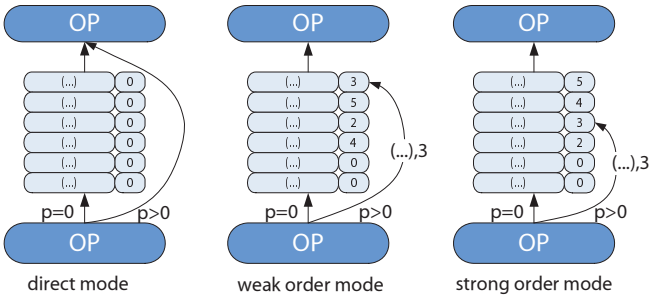



Abbildung 3: Puffermodi zur Bevorzugung priorisierter Datenstromelemente

nen.

## 5 Evaluation

Wir haben den Ansatz zur prioritätsbasierten Verarbeitung von Datenstromelementen im Rahmen unseres Java-basierten ODYSSEUS-Frameworks [JG08] für Datenstrommanagementsysteme implementiert und evaluiert. Unsere Messungen haben sich zunächst darauf beschränkt, zu untersuchen, in wie weit sich die Latenzen für einzelne sich im Datenstrom befindende Elemente mit und ohne Prioritäten verändern. Dabei haben wir es uns zu Nutze gemacht, dass wir in ODYSSEUS an das eigentliche Datenobjekt nahezu beliebige Metadaten anhängen können. In unserem Fall haben wir die Latenz durch die Vergabe von Zeitstempeln messen können: Beim Eintritt in den Plan wurde ein Element mit einem Startzeitstempel versehen und beim Erreichen des obersten Operators mit dem passenden Endzeitstempel, die Differenz ist die Latenz des Elements. Um nicht von Datenverteilungen in Eingabedatenströmen abhängig zu sein, haben wir statt echter Operatoren nur simulierte Benchmark-Operatoren im Anfrageplan verwendet, die jeweils mit Ausführungszeiten und Selektivitäten konfiguriert werden konnten. Die Benchmark-Operatoren verändern die Zeitstempel dabei nicht. So können wir sicher sein, dass wir nur die Effekte messen, die sich durch das Überholen in den Puffern ergeben und die nicht durch spezielle Datenverteilungen bedingt sind. Es bedeutet aber auch, dass nicht das volle Optimierungspotential unseres Ansatzes genutzt wurde, da so keine Vorteile aus den neuen Operatorimplementierungen gezogen werden konnte. Der einzige reguläre Operator ist der Operator zum Setzen der Prioritäten eines Elements. Genauso wie die Latenzinformation wird die Priorität eines Elements auch mit Hilfe von angehängten Metadaten realisiert.

Wir haben für unsere Tests einen komplexen Anfrageplan mit einer maximalen Tiefe von neun bestehend aus ca. 60 Operatoren (plus Puffer) verwendet. Die Tests fanden auf einem Intel Core 2 Duo mit 2,4 Ghz und 2 CPUs, mit 3,2 GByte Hauptspeicher und einem CentOS 5.2 Linux statt.

In den Tests wurden die Anfragen mit verschiedenen, teilweise aus der Literatur bekannten, Scheduling-Strategien und mit den unterschiedlichen Pufferkonzepten ausgeführt. Im

Einzelnen haben wir die folgenden Strategien für das Scheduling verwendet:

**Round Robin:** Die Operatoren werden einfach in der Reihenfolge eingeplant, wie sie im Plan vorkommen. Wenn ein Operator keine Daten zum Verarbeiten vorliegen hat, wird der nächste Operator der Liste betrachtet. Zwei spezielle Ansätze des Round Robin sind die beiden folgenden, die eine spezielle Reihenfolge der Operatoren festlegen.

**Aurora Min-Cost:** Die aus dem Aurora-System stammende Strategie [CcR+03] führt eine topologische Sortierung des Anfrageplans durch, wodurch sicher gestellt wird, dass kein Operator zur Verarbeitung angestoßen wird, dessen Eingabeoperatoren zu einem früheren Zeitpunkt hätten Daten produzieren müssen. Die Verarbeitung wird also insgesamt von unten nach oben und mit möglichst geringem Scheduling-Overhead durchgeführt. Eine mögliche Scheduling-Reihenfolge für den Anfrageplan in Abbildung 2 wäre also $1, 2, 3, 4, 5$.

**Aurora Min-Latency:** Bei dieser ebenfalls aus dem Aurora-System stammenden Strategie wird versucht, die Latenz für ein einzelnes Element im Strom zu reduzieren, in dem immer komplette Pfade von den Operatoren zu den Senken eingeplant werden. Die Reihenfolge der Pfade ist dabei aufsteigend nach der Latenz, die sie für ein Element verursachen, festgelegt. Eine mögliche Scheduling-Reihenfolge für den Anfrageplan in Abbildung 2 wäre also $4, 1, 4, 5, 3, 5, 2$. Auf Grund fehlender Kostenmodellinformationen konnten wir nicht die Originalversion der Strategie verwenden, sondern verzichten auf eine Sortierung der Pfade nach der größten Produktivität. Dies sollte sich eigentlich auch nur zu Beginn der Verarbeitung für die ersten Tupel auswirken und ist aus diesem Grund zu verkraften.

**Biggest-Queue:** Dies ist eine dynamische Strategie, die bei jedem Scheduling-Aufruf immer die Operatoren ausführt, deren Eingabepuffer die meisten Daten enthält.

**Highest-Priority-Queue:** Diese Strategie ist schließlich die einzige, die die Priorität der Daten berücksichtigt. Es werden immer die Operatoren angestoßen, deren vorgelagerter Puffer das Element mit der höchsten Priorität enthält.

In der Standardimplementierung verarbeitet jede Strategie in jedem Durchlauf genau ein Element und wählt dann anschließend einen neuen Operator. Zusätzlich können alle Strategien noch in einem speziellen Modus bei jedem Durchlauf eine Menge oder gar alle Elemente des Puffers verarbeiten (in [ACc+03] Train-Scheduling genannt). Zur Unterscheidung haben wir diesen Modus durch ein angehängtes '+'-Zeichen gekennzeichnet.

Insgesamt haben wir die Scheduling-Ansätze mit fünf verschiedenen Pufferrealisierungen getestet.

**normal:** Dieser Ansatz stellt die Vergleichsbasis dar und unterscheidet nicht zwischen unterschiedlichen Prioritäten. Die Elemente werden einfach in einer verketten Liste gespeichert und nach dem FIFO-Prinzip verarbeitet. Um vergleichbare Laufzeiten zwischen verschieden priorisierten Elementen zu bekommen, wird aber auch in diesem Plan der Operator zum Setzen der Priorität verwendet. Dieser trägt aber mit

Verarbeitungszeiten im Mikrosekundenbereich nicht zu einer relevanten Verschlechterung der Latenz bei.

**direct:** Wir bereits oben beschrieben, werden hier priorisierte Elemente immer sofort an den Nachfolge-Operator geschoben. Die anderen Operatoren werden in eine Liste eingefügt und dann nach dem FIFO-Prinzip verarbeitet.

**weak order:** Dies ist der Ansatz, bei dem alle Elemente in den Puffer geschrieben werden, priorisierte Element aber nach dem LIFO-Prinzip verarbeitet werden. Sie werden also an den Kopf der Liste gesetzt.

**strong order:** In diesem Ansatz werden die priorisierten Elemente an den Anfang der Liste geschrieben, allerdings bezüglich ihrer Priorität einsortiert. In der Liste befinden sich also gleichzeitig priorisierte und nicht priorisierte Elemente.

**strong order2:** In Analogie zum *strong order*-Puffer werden auch hier die priorisierten Elemente nach ihrer Priorität sortiert, jedoch gibt es für priorisierte und für nicht priorisierte Elemente unterschiedliche Listen.

Die *strong order*-Umsetzung hat sich in ersten Tests als nicht verwendbar herausgestellt, da sie – aufgrund des erhöhten Verarbeitungsaufwands beim Einfügen – in allen Tests zu einer Vervielfachung der Latenz geführt hat. Aus diesem Grund haben wir diese Realisierung nicht weiter betrachtet. Das andere Extrem, für jede Priorität eine eigene Liste zu nutzen, ist bisher auch nicht weiter betrachtet worden und könnte noch einmal genauer untersucht werden.

Die Art, wie in dem Anfrageplan die Puffer platziert werden, ist noch statisch: Nach jeder Quelle, vor jedem zustandsbehafteten binären Operator und nach dem Prioritätsoperator wird entsprechend der Pufferrealisierung ein Puffer eingefügt. Eine Besonderheit ergibt sich, wenn der *direct*-Ansatz verwendet wird: In diesem Fall wird ein normaler Puffer und keine der genannten Spezialimplementierungen hinter die Quelle gesetzt um ein Blockieren des Quellen-Threads zu vermeiden. Auch bei der Platzierung von Puffern im Anfrageplan sind noch Untersuchungen von anderen Strategien denkbar und sinnvoll.

Getestet wurde das Verhalten mit einer konstanten Datenrate von $10000$ Elementen pro Sekunde und einer Gesamtanzahl von $100000$ Elementen, die jede der Quellen produziert. Insgesamt werden durch die Anfragen $1,4$ Millionen Elemente verarbeitet. Für jede Prioritätsstufe größer $0$ werden dabei Elemente mit einer Wahrscheinlichkeit von $10\%$ erzeugt, d.h. bspw. wenn es drei unterschiedliche Prioritäten $(0, 9, 10)$ gibt, dann sind etwa $20\%$ aller Elemente priorisiert.

Ein Ergebnis welches relativ schnell offensichtlich wurde, ist dass sich der *strong order2*-Ansatz umso schlechter verhielt, je mehr unterschiedliche Prioritäten im Plan vorkommen, insbesondere in Kombination mit einer '+' Strategie. Dies ist auch nicht weiter verwunderlich, da das Einsortieren in die Liste mehr Zeit kostet, je mehr priorisierte Elemente bereits in der Liste enthalten sind.

Im Folgenden haben wir nur noch Anfragen mit vier unterschiedlichen Prioritäten, d.h. $0$ für unpriorisierte Elemente und $8, 9$ und $10$ als priorisierte Elemente, betrachtet. Hierbei wurde deutlich, dass trotz des hohen Anteils an priorisierten Elementen, sich keine

erkennbaren Unterschiede für die einzelnen Prioritätsstufen ausmachen ließen – *strong order2* also keinen Mehrwert brachte. Hier muss noch genauer untersucht werden, ob andere Prioritätsverteilungen im Strom Auswirkungen haben könnten. In der Abbildung 4 sind aus diesem Grund nur noch die beiden *direct*- und *weak order*-Pufferverfahren jeweils im Verhältnis zum *normal*-Pufferverfahren gegenüber gestellt. Ein Wert von $0,5$ bedeutet dabei, dass sich die durchschnittliche Latenz im Verhältnis zum Verfahren ohne Berücksichtigung der Prioritäten halbiert hat, ein Wert von $2$ bedeutet, dass sie sich verdoppelt hat.

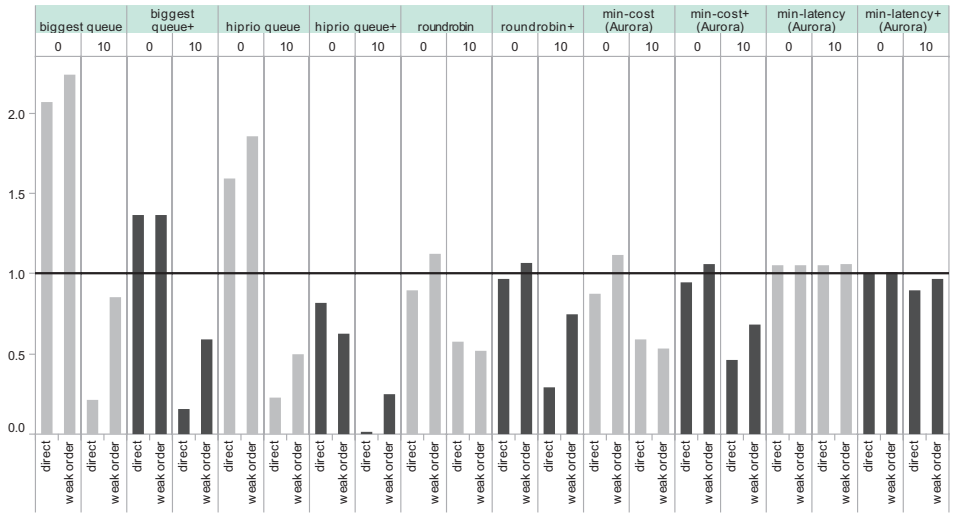



Abbildung 4: Änderung der Latenz gegenüber Normal-Puffermodus

Es fällt auf, dass in fast allen Verfahren eine Verbesserung der Latenz priorisierter Elemente zu verzeichnen ist. Die Verbesserung fällt dabei fast durchgehend sehr deutlich aus. Ebenso fällt auf, dass der *direct*-Ansatz fast immer zu einer größeren Verbesserung führt, als die *weak order*-Puffer. Dies war zu erwarten, da sich der Nachteil dieses Ansatzes (Blockierung des unteren Threads) erst bei sehr teuren Anfrageplänen (also welchen mit hohen Verarbeitungszeiten pro Element) bemerkbar machen sollte. Hier wäre also noch zu untersuchen, ab wann die Nachteile des *direct*-Ansatzes überwiegen. Bei der *Highest-Priority-Queue* lässt sich die größte Latenzverbesserung für priorisierte Elemente feststellen. Dies liegt daran, dass es die einzige Strategie ist, die das Scheduling an den priorisierten Elementen orientiert. Dadurch, dass zunächst Puffer ausgewählt werden, die priorisierte Elemente enthalten, lassen sich in Verbindung mit den *direct*-Puffern Verbesserungen von mehr als Faktor fünfzig beobachten. Grundsätzlich sind die Strategien, die mehrere Elemente auf einmal pro Puffer verarbeiten, durch den verringerten Scheduling-Overhead besser.

Die Laufzeiten der Anfragen und damit der Durchsatz scheinen nicht von den Puffermodi beeinflusst zu werden. Ausnahmen bilden die *Highest-Priority-Queue*- und die *Biggest-Queue*-Strategie, wenn diese nach jedem Element einen neuen Puffer für das Scheduling

auswählen (in Abbildung 4 ohne angehängtes '+' dargestellt). In den beiden Fällen steigen die Laufzeiten und wie in Abbildung 4 zu sehen auch die Latenzen für normale Elemente deutlich. Hierbei scheint im getesteten Szenario der Scheduling-Overhead so groß zu werden, dass es zu einer Art Überlast des Systems kommt und sich die Puffer nach den Quellenoperatoren zunächst stark füllen, bevor sie dann abgearbeitet werden.

Die beste durchschnittliche Latenz über alle Elemente liefert die *Aurora Min-Latency*-Strategie. Bei ihr ließen sicher aber leider so gut wie keine Gewinne durch die Priorisierung erzielen. Für die Zukunft scheint daher eine Verbindung der *Highest-Priority-Queue*-Strategie mit der *Min-Latency*-Strategie vielversprechend zu sein, wenn es auf die Optimierung der Gesamtlatenz unter zusätzlicher Berücksichtigung der Prioritäten ankommt.

## 6 Verwandte Arbeiten

Eine effiziente Verarbeitung von Datenströmen ist entscheidend auch von der Strategie abhängig, mit der die Operatoren eines Anfrageplans zur Ausführung gebracht werden [SCLP08]. Eine Reihe von Arbeiten hat sich bereits mit diesem Thema befasst. Stellvertretend sollen hier die Arbeiten zum Chain-Scheduling [BBDM03] und in Aurora [CcR+03], in NiagaraCQ [HFAE03] und PIPES [CHK+07] genannt werden. Alle uns bekannten Ansätze betrachten zum Priorisieren die Operatoren und optimieren ihre Strategien i.d.R. auf einen hohen Durchsatz, lediglich Aurora erlaubt die Zuweisung von Quality-of-Service Ansprüchen an Anfragen, wodurch eine gewisse Priorisierung auf Anfrageebene erfolgen kann. Ebenfalls nur auf Operatorebene betrachtet [UF01] die Priorisierung von Operatoren innerhalb einer Anfrage. Im Gegensatz dazu ist es mit unserem Ansatz möglich, unterschiedliche Prioritäten innerhalb des selben Pfades eines Anfrageplans zu betrachten.

Eine wesentliche Vorrausetzung für unseren Ansatz stellt die "out-of-order"-Verarbeitung von Datenstromelementen dar. In der Arbeit von [LTS+08] wurde gezeigt, dass diese Art der Verarbeitung im Kontext von Verzögerungen zu geringerem Speicherverbrauch, einer geringeren durchschnittlichen Latenz sowie zu einer kürzeren Laufzeit der Anfragen führen kann. Genau wie in [LTS+08] werden auch in [DR04] Punctuations genutzt, um ein Nicht-Blockieren von zustandsbehafteten Operatoren zu realisieren. In letzterer Arbeit werden sie nebem dem Einsatz von Zeitfenstern dazu genutzt, die Erzeugung von Ergebnissen in einem Join-Operator zu beschleunigen und damit ebenfalls eine geringere durchschnittliche Latenz zu erzielen. Beim Einsatz von Punctuations besteht jedoch die Gefahr, dass evtl. Tupel verworfen werden müssen, falls eine Punctuation an einem Eingabedatenstrom auf Grund fehlender oder fehlerhafter Fortschrittsinformationen über den Datenstrom zu früh erzeugt wird (vgl. [SW04]). Dieses Problem ergibt sich auch beim Schätzen der maximalen Verspätung eines Tupels (Slack), wie es beispielsweise in [ACc+03, LLD+07] angewandt wird. Wir stellen mit dieser Arbeit eine „out-of-order"-Verarbeitung einzelner Datenstromelemente vor, die ohne Punctuations oder Slack-Parameter auskommt und zusätzliche Optimierungsmöglichkeiten bei der Verarbeitung dieser Elemente bietet.

## 7 Fazit

Es gibt Anwendungsfälle für Datenstrommanagementsysteme (wie die Überwachung von Energieerzeugungsanlagen), in denen die Latenz bei der Verarbeitung besonderer Datenstromelementen (wie Alarme, Warnungen) wichtig ist. Bisherige Ansätze zur Anfragepriorisierung helfen bei diesem Problem nicht weiter, da ein entsprechendes Element im Extremfall in allen Anfragen gleichzeitig verarbeitet werden muss. Unsers Wissens nach sind wir die ersten, die ein Verfahren vorstellen, das es ermöglicht, wichtige Datenstromelemente bei der Verarbeitung zu bevorzugen. Wir haben in dieser Arbeit eine formale Grundlage für die Priorisierung einzelner Elemente in Datenströmen geschaffen. Der vorgestellte Ansatz bietet gleich zwei Vorteile bei der Verarbeitung priorisierter Elemente. Zum einen ermöglicht er es priorisierten Elementen andere Elemente im Datenstrom zu überholen und zum anderen können priorisierte Ergebnisse zustandsbehafteter Operatoren schneller erzeugt werden, als die normale Verarbeitung dies erlauben würde. Dabei wird die Anfragesemantik nicht geändert. Damit dies gewährleistet ist, haben wir die Algorithmen der zustandsbehafteten Operatoren angepasst. In dieser Arbeit wurde exemplarisch ein prioritätskompatibler Joinalgorithmus vorgestellt und nicht nur gezeigt, dass dieser korrekte Ergebnisse erzeugt, sondern auch eine Abschätzung des Gewinns bei der Latenz für priorisierte Elemente geliefert.

Wir haben verschiedene Strategien für Puffer in Anfrageplänen entwickelt, die priorisierte Elemente bevorzugt behandeln. In einer Evaluation mit diversen Schedulingstrategien und unseren neuen Puffermodi konnte ein meist sehr deutlicher Gewinn bei der durchschnittlichen Latenz festgestellt werden. Eine in dieser Arbeit neu entwickelte Scheduling-Strategie, die die Priorisierung von Elementen berücksichtigt, konnte besonders starke Reduktionen der Latenz erzielen.

Zukünftig wollen wir – neben ausführlicheren Tests weiterer Szenarien – vor allem die Verbindung von Schedulingstrategien mit elementbasierter Priorisierung untersuchen. Die Latenz bei Verbünden mit hoch priorisierten Elementen hängt auch vom rechtzeitigen Eintreffen der Verbundpartner ab. Daher wollen wir Techniken untersuchen, mit deren Hilfe diese schneller bereitgestellt werden können.

**Danksagung**: Wir danken Martin Hecker für seine wertvollen Hinweise.

## Literatur

[ABW06] Arvind Arasu, Shivnath Babu und Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2), 2006.

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul und Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2), 2003.

[BBDM03] Brian Babcock, Shivnath Babu, Mayur Datar und Rajeev Motwani. Chain : Operator Scheduling for Memory Minimization in Data Stream Systems. In Alon Y. Halevy,

Zachary G. Ives und AnHai Doan, Hrsg., *SIGMOD 2003: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM, 2003.

[CcR+03] Donald Carney, Ugur Çetintemel, Alex Rasin, Stanley B. Zdonik, Mitch Cherniack und Michael Stonebraker. Operator Scheduling in a Data Stream Manager. In *VLDB 2003: Proc. of 29th International Conference on Very Large Data Bases*, 2003.

[CHK+07] Michael Cammert, Christoph Heinz, Jürgen Krämer, Bernhard Seeger, Sonny Vaupel und Udo Wolske. Flexible Multi-Threaded Scheduling for Continuous Queries over Data Streams. In *International Workshop on Scalable Stream Processing Systems*, 2007.

[DR04] Luping Ding und Elke A. Rundensteiner. Evaluating window joins over punctuated streams. In *CIKM '04: Proc. of the thirteenth ACM international conference on Information and knowledge management*, New York, NY, USA, 2004. ACM.

[DSTW02] Jens-Peter Dittrich, Bernhard Seeger, David Scot Taylor und Peter Widmayer. Progressive Merge Join: A Generic and Non-blocking Sort-based Join Algorithm. In *VLDB 2002, Proc. of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, 2002.

[HFAE03] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref und Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *VLDB 2003: Proc. of the 29th international conference on Very Large Data Bases*. VLDB Endowment, 2003.

[HH99] Peter J. Haas und Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In Alex Delis, Christos Faloutsos und Shahram Ghandeharizadeh, Hrsg., *SIGMOD 1999: Proc. ACM SIGMOD International Conference on Management of Data*. ACM Press, 1999.

[JG08] Jonas Jacobi und Marco Grawunder. ODYSSEUS: Ein flexibles Framework zum Erstellen anwendungsspezifischer Datenstrommanagementsysteme. In Hagen Höpfner und Friederike Klan, Hrsg., *Grundlagen von Datenbanken*, Jgg. 01/2008 of *Technical Report*. School of Information Technology, International University in Germany, 2008.

[Krä07] Jürgen Krämer. *Continuous Queries over Data Streams – Semantics and Implementation*. Dissertation, Philipps-Universität Marburg, Marburg an der Lahn, 2007.

[LLD+07] Ming Li, Mo Liu, Luping Ding, Elke A. Rundensteiner und Murali Mani. Event Stream Processing with Out-of-Order Data Arrival. In *ICDCSW '07: Proc. of the 27th International Conference on Distributed Computing Systems Workshops*, Washington, DC, USA, 2007. IEEE Computer Society.

[LTS+08] Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson und David Maier. Out-of-Order Processing: A New Architecture for High-Performance Stream Systems. In *VLDB 2008: Proc. of 34th International Conference on Very Large Data Bases*, 2008.

[SCLP08] Mohamed A. Sharaf, Panos K. Chrysanthis, Alexandros Labrinidis und Kirk Pruhs. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Trans. Database Syst.*, 33(1), 2008.

[SW04] Utkarsh Srivastava und Jennifer Widom. Flexible time management in data stream systems. In *PODS 2004: Proc. of the twenty-third ACM SIGMOD-SIGACT-SIGART symposiumon Principles of database systems*, New York, NY, USA, 2004. ACM Press.

[UF01] Tolga Urhan und Michael J. Franklin. Dynamic Pipeline Scheduling for Improving Interactive Query Performance. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao und Richard T. Snodgrass, Hrsg., *VLDB 2001: Proc. of 27th International Conference on Very Large Data Bases*. Morgan Kaufmann, 2001.

# Stream-Based Web Service Invocation

Steffen Preißler, Hannes Voigt, Dirk Habich, Wolfgang Lehner
Technische Universität Dresden
Lehrstuhl für Datenbanken
dbgroup@mail.inf.tu-dresden.de

**Abstract:**
Service-oriented architectures (SOA) based on Web service technology play an increasingly important role in many different application areas. The current service invocation methodology suffers from performance problems and heavy resource consumption when services are used to process large amounts of data. A number of solutions to this problem have been proposed. Unfortunately, these modified invocation methodologies are applicable only to a limited number of business scenarios, because they do not provide all features of the traditional methodology. In this paper, we introduce a new service invocation methodology that allows large volume data processing and does not limit applicability. Our approach macerates the existing request–response paradigm and incorporates stream semantics into the Web service invocation methodology without reducing its features. We describe our stream-based service invocation and present evaluation results that show the advantages of our approach.

## 1 Introduction

The paradigm of service-oriented architectures (SOA) becomes more and more attractive to a variety of application areas. The base concept of SOA is a service. A service represents a mechanism to enable access to one or more capabilities using a prescribed interface [OAS06]. Web services [W3C02] are the de-facto standard for implementing such a mechanism and thus service-oriented infrastructures. To use a Web service, a client sends a request to the service interface. The service processes this request and afterwards, according to the offered functionality, sends a response back to client. This so called *request–response paradigm* is a foundation of service-orientation and guarantees loosely-coupled and autonomous services.

The communication between a client and the services it invokes is message-based. The size of these messages is performance critical. The well investigated memory consumption and performance problem in the XML-based message communication arises when serializing or de-serializing XML messages to or from other programming language paradigms [CGB02, MF05, CYZ+06, HPL+07]. This inefficiency is owed to the impedance mismatch between the hierarchical XML model and the (mostly object-oriented) programming language model that processes the XML content [vE03]. Thus a conversion needs to be done, which devours up to 90% of the processing time [DA03] and up to ten times the in-memory size of its correspondent text-based XML representa-

tion [vE03] when creating the whole Document Object Model (DOM) tree. If large data sets have to be exchanged between client and service, these issues become more evident.

Streaming is a much more efficient way to process large sets of equally structured data items – well studied in the field of data stream system. However, the stream processing paradigm is hard to realize with the current Web service concept, mainly because the request–response paradigm of Web service communication does not support service invocations with continuous data. The message serialization and de-serialization is processed in a single step before the whole message is send and after the whole message arrived, respectively. The straight forward approach of mapping every data items to a single service invocations proved to be very inefficient [GYSD08]. Additionally it strongly restricts the business logic of the service, since it does not allow to process data items in a common context.

In this paper we introduce a concept to add stream semantics to the Web service invocation and its execution. Our innovative Web service invocation approach macerates the traditional *request–response paradigm* to incorporate stream-based semantics and establishes, thereby, an efficient way to transmit and process continuous and large-scale data sets by a specific service. Our concept is (i) more efficient in memory consumption and performance, (ii) provides a common context for all data item in a large volume data set and (iii) allow for more functionality like intermediate results.

The reminder of the paper is organized as follows. First, we present our approach for stream-based Web service invocation and execution in detail (Section 2). Second, we evaluate our approach against current techniques for data transmission on Web service level, as they can be found in current infrastructures, in terms of response time (Section 3). Finally, we conclude and give a short outlook on future work (Section 4).

## 2 Stream-based Web Services

In this section we introduce our approach to incorporate stream semantics into the Web Service invocation and its execution. It enables clients to transfer large scale and continuous data to a service as a stream. Furthermore, it enables the service to process this data in a stream-based fashion. This leads to lesser memory consumption within the service instance, a higher throughput and instant response items of already finished data parts. These statements are evaluated against other approaches in Section 3.

To provide a formal foundation for the upcoming sections, let $D$ be a data set with $n$ data items $d_i$ so that $D = (d_1, d_2, \ldots, d_n)$. Each data item $d_i$ represents some kind of equally structured XML data. A given data set $D$ also can be seen as an amount of single-item messages with an equally structured content. So the content of a single-item message corresponds to one data item $d_i$ and data set $D$ corresponds to the sum of all single-item messages sent to the service. This mapping provides a unified view of all data that emerges between a client and a service and allows for the usage of our stream-based approach in a consistent and a more general way.

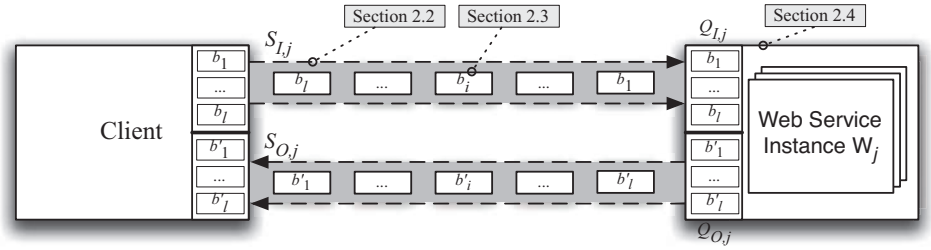



Figure 1: Invocation model with stream semantics.

## 2.1 Overview

The foundation of SOA is the *request–response paradigm*. Therefore, our stream-based Web Service invocation and its execution in this architecture should retain this fundamental building block. So to incorporate stream semantics into the Web Services invocation model and to preserve the fundamental paradigm, three adjustments have to be made. An overview of the adjustments described below is depicted in Figure 1 and includes the reference to the specific sections.

First we define a request message $R$ containing data set $D$ of size $n$ as an input stream $S_I$ for the service. Furthermore a response message $R'$ returned by the service containing data set $D'$ is defined as an output stream $S_O$. Since the input stream is limited to the size of $D$ a stream is only established for the time $D$ is transmitted. Hence it represents a request in a traditional point of view. Every request $R_j$ with one separate $D_j$ creates a new input stream $S_{I,j}$ and, depending on the service's functionality, an output stream $S_{O,j}$ (see Figure 1). This implies that every $\{S_{I,j}, S_{O,j}\}$ pair belongs to exactly one client request $R_j$ and one service instance $W_j$ processing $R_j$. Now we have two streams which preserve a common context for all data items in $D$.

Second, data set $D$ has to be modified somehow to be processed by the service in a stream-based fashion. So data set $D$ is divided into $l$ processing buckets $b_i$ with $B = (b_1, b_2, \ldots, b_l)$ that can be executed by the service in one step. The size and structure of all processing buckets $B$ are predetermined by the service and have to be described in the service description (Section 2.3).

Third, in the traditional service invocation model the way a service processes the XML data has to be adjusted. Traditionally, one service instance $W_j$, which is instantiated by one service request $R_j$, can randomly access the whole data set $D_j$ of its request. $W_j$ builds $D'_j$ as the content for $R'_j$ until $D_j$ is processed completely. To incorporate a stream-based data processing into the service execution, the service instance must be aware of an input queue where items $d_{i,j}$ arrive and an output queue where response items $d'_{i,j}$ have to be delivered to. Both queues work concurrently that means the instance can already deliver response items while still receiving request items. In the following subsections we will describe the three modifications in more detail.

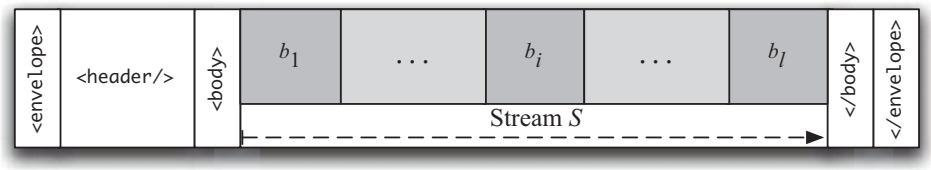



Figure 2: Stream-based SOAP message.

### 2.2 Stream Definition

The streams are realized by taking the SOAP specification. Remember a SOAP message consists of a header containing meta data and a body containing the user data that means the data set $D$. We structure the stream in the same way. The header comprises meta data about the stream itself and is used for negotiation. The body contains the data set distributed over $l$ processing buckets $b_i$. Figure 2 depicts the structure of this stream. Actually, the whole stream of a request message $R$ can be seen as stretched SOAP message.

To transfer $R$ with stream semantics, the SOAP header is sent with all information that would normally be specified within the traditional SOAP header including all WS-* extensions. This opens the lower level HTTP connection. Since the whole message is already streamed using TCP/IP and HTTP, these lower level protocols are used as a channel to establish the stream and to stream structured XML content.

After establishing the input stream $S_I$, the starting SOAP `body` tag is transmitted. Now the client can send data set $D$ with the help of processing buckets $b_i$, which actually are XML snippets of the data set $D$ with a predefined structure and size (see Section 2.3). Note that the buckets are transmitted in a deeper XML hierarchy level than the already transfered `body` tag. When the client has finished sending $D$, it transmits the closing SOAP `body` tag, which resides on the same hierarchical level like the starting `body` tag, and the closing SOAP `envelope` tag. This triggers the disconnection of $S_I$.

The output stream $S_O$ from the service to the client is established when the first response bucket $b'_1$ has to be sent. This behavior conforms to the traditional approach of sending the response message. The establishment of output stream $S_O$ works in the same methodology as the input stream $S_I$. Since one processing bucket $b_i$ can be processed by the service in one step, the response bucket $b'_i$ can be streamed back immediately after successfully processing $b_i$ and while still receiving further buckets. $S_O$ is closed when the last bucket $b'_l$, the closing `body` tag and the closing `envelope` tag is sent. The advantage of sending the corresponding `body` and `envelope` tags as stream delimiter is that the whole stream itself forms one large SOAP message. Therefore it can be read by a traditional service, which buffers the whole XML stream and process the full DOM tree at once.

### 2.3 Bucket Definition and Description

In this section, we define the structure of processing buckets in detail and how these processing buckets are described within the service description. As already mentioned in 2.1 we divide $D$ into $l$ processing buckets. This enables $D$ to be processed in a stream-based fashion since one bucket defines the granularity of $D$, that is transmitted over the stream and to the service in one step. Furthermore the concept of processing bucket acts as an intermediate layer between $D$ and its data items $d_i$ and adds conceptional flexibility for further extensions shortly described in Section 4.

Remember, data set $D$ represents an array of equally structured data items $d_i$. We define one bucket $b_i$ containing $m$ data items with $b_i = (d_1, \ldots, d_m)$. Each bucket is simply a collection of single data items in pure XML notation without further information. So one bucket has the simple *textual* form of $b_i = (e_1, \ldots, e_m)$ with $e_i$ denotes the root element of data item $d_i$ enclosing all its children elements.

The number of data items in each processing bucket can differ between a request bucket $b_i$ and its response bucket $b'_i$. The most common assumption is, that every data item $d_i$ generates exactly one response item $d'_i$. This describes a $N : N$ relationship where $N$ data items generate $N$ response items. If one data item generates a number of response items, a $1 : N$ relationship is reflected. An example is a service that returns all invoices for a given customer id as single response items. With both relationships, the response stream allows to stream back response items although not all request items have been processed completely.

A different case is the $N : 1$ relationship as it can be found in services implementing several aggregate functions that, e.g., computes the average value of all data items. Since the exact average value is computed with the arrival of the last data item $d_n$, the client has to wait for $D$ to be processed completely. Thus no response items can be streamed back in between. But in certain scenarios, preliminary values may be already used for further processing or for a definition of stop criterions. To avoid this drawback, the application logic on service side can be extended to put preliminary values to the output stream. These preliminary values can be flagged with the help of an additional attribute, which can be easily added to the response item's XML structure within the service description.

The definition of processing buckets is predetermined by the service. Since the service provides the implementation to process the buckets, its definition must reside within the WSDL document of the service. The client uses this bucket definition to transmit the given data set accordingly. In addition, the WSDL must indicate the capability of the stream-based service. To describe these two parts, we augment the WSDL document in two sections. First, we add an attribute `streaming` in its own namespace to the `operation` element in the porttype section to indicate the stream-based processing capability of this operation. Second, we augment the XML Schema definition for the request message and the response message of this operation. Since data set $D$ is an array of equally structured data items $d_i$, the XML Schema definition describes an unbounded set of complex elements containing the structure of every data item. To define processing buckets, we add an attribute `processingBuckets` to the element that forms the envelope for one data

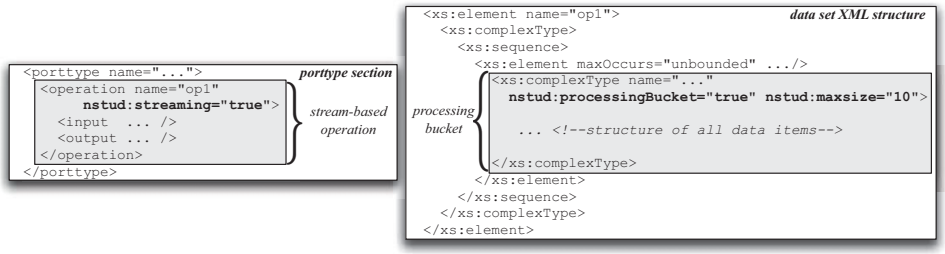



Figure 3: WSDL extensions within porttype section and the data set XML structure.

item. Additionally, an attribute `maxsize` prescribes the maximum number of data items that can be bundled as one processing bucket. This attribute allows optimizing the data transfer for small data items. Figure 3 depicts the WSDL extensions for the `op1` operation to flag it for its stream-based capability and to define its processing buckets. The processing bucket definition for response items is not shown but is labeled in the same fashion as the request items.

### 2.4 XML Processing Model

For a stream-based XML processing of $D$ on service side, the traditional XML processing model and hence the programming model have to be adapted. To access the XML document transmitted to the service, current approaches and implementations stores the whole XML message in a buffer and the Document Object Model (DOM) tree is built. A reference of this tree, that is, the reference of its root element is passed to the functional implementation which can randomly access all nodes of the document. The document $D$ is processed and a response document $D'$ is built. Only after processing $D$ successfully, $D'$ is returned to the framework. This execution model serves well in the context of the strict request–response and hence stepwise execution model. In the context of stream-based document processing three modifications have to be done to the application logic and to the framework hosting it.

As a first modification we introduce an input queue $Q_I$. Since we are not able to guarantee that all data items $d_i$ can be processed in the speed the processing buckets arrive, the input queue buffers all processing buckets $b_i$ and provides an interface for the application logic to access the application data within these buckets. This input queue $Q_I$ is instantiated for every service instance $W_j$ and is already depicted in Figure 1.

Second, we incorporate a cursor-like, forward-only XML processing model within the application logic. We assume that not all data items $d_i$ fit into the service instances main memory at once, thus the incoming data items have to be accessible in a cursor-like, forward-only fashion and the application logic has to apply some window semantics to operate on all $d_i$. The application logic of $W_j$ pulls every processing bucket out of the

input queue $Q_{I,j}$ and processes it accordingly. Since a processing bucket consists of a list of data items in pure XML text, these data items can be parsed and a partial DOM-tree can be built representing all data items in one processing bucket.

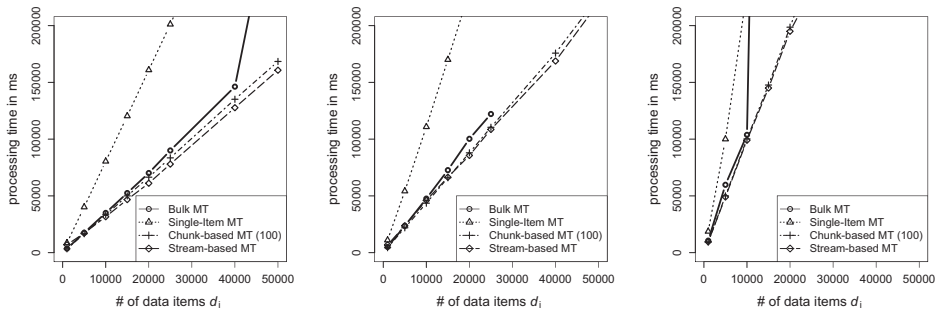
The third modification for our approach is the supply of an interface to place already processed data items $d'_i \in D'$ in output queue $Q_O$ which is used by the hosting framework to send intermediate results over the output stream. This allows the framework for sending response buckets $b'$ while still receiving the request. The framework bundles the items in $Q_O$ into processing buckets according the service description. This output queue $Q_O$ is instantiated for every service instance $W_j$ and is already depicted in Figure 1.

## 3 Evaluation

In this section, we evaluate our stream-based Web service invocation and execution approach. We investigate how efficiently a large data set can be processed by a Web service. Assume a large data set $D = (d_1, d_2, \ldots, d_n)$ with thousands of equally structured business items $d_i$. Furthermore, assume a client, which wants this data set $D$ to be processed by a Web service $W$. The client requires a certain number $k$ of service invocations (requests) $r_i$ to process the whole data set. We denote the set of all required invocations as $R$ with $R = (r_1, r_2, \ldots, r_k)$.

We investigate four invocation models, that means four ways to process $D$ with the Web service $W$. One of those four is our *stream-based message transfer* approach. We compare our approach with the three traditional invocation models *bulk message transfer*, *single-item message transfer* and *chunk-based message transfer*. *Bulk message transfer* is the straightforward way to process data set $D$ with a Web service $W$. Utilizing *bulk message transfer*, we pack the whole data set $D$ with size $s_D = \sum_1^n s_{d_i}$ into one request and send it to the service with one service call $r$ so that $|R| = 1$. After receiving the whole request message $r$ and thereby the whole data set $D$, the service $W$ processes the complete data set. As result $W$ creates a response message $r'$ and sends it back to the caller. *Single-item message transfer* is the opposite way to process a data set $D$ with a Web service $W$. In *single-item message transfer* we put every item $d_i \in D$ in one separate service request $r_i$ so that $|R| = |D|$. After one item $d_i$ has been processed by the service $W$ and we have received the response, we send a new request containing the next item $d_{i+1}$. *Chunk-based message transfer* is the generalization of the two aforementioned models and was first formally considered in [SMWM06]. The main idea is to distribute the items $d_i$ of the data set $D$ to $k$ different service requests $r_j$ so that $|R| = k$ with $1 \leq k \leq |D|$. If the items are uniformly distributed, $r_j$ contains $\frac{|D|}{k}$ data items of $D$. Obviously, *bulk message transfer* corresponds to *chunk-based message transfer* with $k = 1$ and *single-item message transfer* corresponds to *chunk-based message transfer* with $k = |D|$. The chunk size significantly influences the response time and can be tuned accordingly. For example, [GYSD08] proposes an approach to automatically optimize the chunk size.

To evaluate the performance behavior of the four invocation models, we conducted a series of experiments. In detail, we measured the response time of a Web service invoked with a




(a) One Client. (b) Two Clients. (c) Four Clients.

Figure 4: Total response time of different invocation models.

data set $D$ for different numbers of concurrent invocations and different sizes of $D$. The service and the clients were hosted on a single machine with 2GB RAM. 512MB RAM have been assigned to the server as heap size for hosting this single service. Response times were measured on client side. All results are an average of 30 runs. For *chunk-based message transfer* we chose a chunk size of 100 data items. Other chunk sizes showed worse results.

Figure 4 illustrates the server's total response time, that means how long the client had to wait for all data items being processed. Figure 4(a), Figure 4(b) and Figure 4(c) show the results for one client, two concurrent clients and four concurrent clients, respectively. As we see, the number of concurrent clients influences the response time of the server by a constant factor independently of the used invocation model. Accordingly, we see similar results for each number of concurrent clients. *Single-item message transfer* has the worst total response time. Putting every single item in a single message leads to a significant overhead in message size and message parsing, which results in longer response times. The overhead arises because the message header is sent and parsed in total $n$ times. *Bulk message transfer* and *stream-based message transfer* do not involve such overhead and, consequently, show better response times. Although *chunk-based message transfer* also leads to overhead, its overhead is significantly smaller and is compensated by advantages in message processing. *Bulk message transfer* shows the best performance of all invocation models for small data sets. For larger data sets, it is outperformed by *chunk-based message transfer* and our *stream-based message transfer* approach. *Stream-based message transfer* shows for large data sets the lowest total runtime of all invocation models.

Figure 4 also illustrates the scalability of the four invocation models. Of course, for all invocation models the response time increases linearly with the size of the data set $D$. But the increment differs from model to model. *Single-item message transfer* scales worst, since its overhead increase with the size of the data set $D$. *Bulk message transfer* scale better as long as size of $D$ remains within the memory limit (40.000 data items in our experiment); close to the memory limit, the performance breaks down; beyond the memory

limit, *bulk message transfer* is not applicable anymore. *Chunk-based message transfer* and *stream-based message transfer* scale similarly good as *bulk message transfer* does below the memory limit. But since both have a constant memory usage they scale also for data sets larger than the memory.

Aside from their performance behavior, the four invocation models differ in their functional properties. For many business scenarios, the application logic essentially requires to process data items within a common context. *Single-item message transfer* does not provide a common context since every item is processed separately. *Chunk-based message transfer* processes data items chunk-wise; therefore a common context exists for all items of one chunk. Nevertheless, many scenarios need a common context for all data items. *Bulk message transfer* and *stream-based message transfer* provide such a common context inherently and are in this respect not restricted to certain business scenarios.

To conclude, the evaluation shows that large data sets cannot be processed efficiently with common approaches. *Bulk message transfer* as the traditional invocation model maps directly to the request–response paradigm. It preserves one common context for all items of the data set it and enables the service to be implemented stateless. But it suffers from long latency, i.e., the time until a client receives the first response message, and bad scalability. In contrast, the invocation model *single-item message transfer* packs every data item into a single request message. This results in low latency, but it shows a worse performance and does not provide one common context. *Chunk-based message transfer*, as the generalized approach, shows the best performance if the optimal chunk size is chosen. But it has mediocre latency and lacks a common context for all data items. Our new *stream-based message transfer* approach proves to be superior to the traditional invocation models. With shortest total response time and the best scalability it shows the best performance [PVHL08]. It offers low latency, so the client receives first responses after a short time. Additionally, it is applicable to a wide range of use cases, because it inherently provides a common processing context for all data items.

## 4 Conclusion and Outlook

In this paper, we have extended the Web service environment to enable stream-based invocations and XML processing. Thereby, we utilize the existing protocols and specifications to lift the already existent streaming in low-level protocols up to the application layer. We also implemented our approach within an existing SOAP engine and evaluated the performance benefits in comparison to other common techniques. In general, our stream-based approach consists of (i) an extended service interface, (ii) an enhanced SOAP message interpretation with a stream bucket concept, and (iii) an appropriate XML processing model on application layer. From the evaluation, we can conclude that our stream-based approach decreases memory consumption and CPU usage for large scale data, while preserving full applicability.

However, the proposed data bucket concept is considered as a first step, because one bucket consists only of pure application data and no further information is annotated. This results

in a compatible but not very robust stream management. A more sophisticated bucket design incorporating metadata about a bucket allow for more flexibility in bucket processing, more robust data transmission and fine-grained exception management. To annotate a bucket with metadata, the application data in one bucket can be enclosed with a bucket envelope in a separate namespace. This envelope comprises metadata like a unique bucket identifier for a request item and response item correlation or other information like performance hints or service status that the client framework can use to optimize its transfer. A fine-grained exception management can be achieved by replacing certain response items with warnings or exceptions that apply only to these items. All of this was not scope of this paper and is considered as future work.

## 5 Acknowledgements

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference "'01MQ07012"'. The authors take the responsibility for the contents.

## References

[CGB02] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. Investigating the Limits of SOAP Performance for Scientific Computing. In *HPDC*, 2002.

[CYZ+06] Shiping Chen, Bo Yan, John Zic, Ren Liu, and Alex Ng. Evaluation and Modeling of Web Services Performance. In *ICWS*, pages 437–444, 2006.

[DA03] K. Devaram and D. Andresen. SOAP Optimization via Parameterized Client-side Caching. In *Parallel and Distributed Computing and Systems*. ACTA Press, 2003.

[GYSD08] Anastasios Gounaris, Christos Yfoulis, Rizos Sakellariou, and Marios Dikaiakos. Robust runtime optimization of data transfer in queries over WS. In *ICDE*, 2008.

[HPL+07] Dirk Habich, Steffen Preissler, Wolfgang Lehner, Sebastian Richly, Uwe Aßmann, Mike Grasselt, and Albert Maier. Data-Grey-Box Web Services in Data-Centric Environments. In *ICWS*, pages 976–983, 2007.

[MF05] Ana C. C. Machado and Carlos A. G. Ferraz. Guidelines for performance evaluation of web services. In *WebMedia*, pages 1–10. ACM Press, 2005.

[OAS06] OASIS. OASIS Reference Model for Service Oriented Architecture 1.0, 2006. `http://www.oasis-open.org/specs/index.php#soa-rmv1.0`.

[PVHL08] Steffen Preissler, Hannes Voigt, Dirk Habich, and Wolfgang Lehner. Data Stream Semantics in Service-Oriented Environments. Technical report, Technische Universität Dresden, 2008. `http://wwwdb.inf.tu-dresden.de/files/publications/PVHL08.pdf`.

[SMWM06] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query Optimization over Web Services. In *VLDB*, pages 355–366, 2006.

[vE03] Robert van Engelen. Pushing the SOAP Envelope with Web Services for Scientific Computing. In *ICWS*, pages 346–352, 2003.

[W3C02] World Wide Web Consortium W3C. Web Service specifications, 2002. `http://www.w3.org/2002/ws/`.

# Dissertationspreise

# Top-K Aggregation Queries in Large-Scale Distributed Systems*

[PhD Thesis Summary]

Sebastian Michel
Max-Planck-Institut für Informatik, Saarbrücken, Germany
(Current: Ecole Polytechnique Fédérale de Lausanne, Switzerland)

**Abstract:** Distributed top-$k$ query processing has become an essential functionality in a large number of emerging application classes like Internet traffic monitoring and Peer-to-Peer Web search. This work addresses efficient algorithms for distributed top-$k$ queries in wide-area networks where the index lists for the attribute values (or text terms) of a query are distributed across a number of data peers.

## 1 Introduction

The success and growth of the Web and the Internet is spurring the development of an ever increasing number of interesting application classes, from Internet-scale monitoring, to aggregation queries in sensor networks, and to peer-to-peer Web searching. Internet traffic monitoring is crucial for understanding the nature of modern applications' load characteristics such as P2P file sharing, news feeds, or Blogs, and uses network instrumentation at different levels and time scales. As the underlying routers and other components of the observatory infrastructure are highly distributed, analyzing the logged data often requires distributed aggregation and iceberg queries (i.e., top-k computations over aggregated traffic measures ). Similar requirements arise for detecting traffic anomalies such as network intrusions or denial-of-service attacks. Sensor networks are gaining great importance for monitoring environmental data such as water quality measures in rivers or other measurements of the physical world. Here, too, evaluating the data naturally leads to distributed aggregation queries where one is often interested only in the top-k query results, e.g., the top water streams with the highest nitrate concentration. P2P Web search is an emerging alternative to centralized search engines that bear various intriguing potentials: lower susceptibility to search engine spam and manipulation, exploitation of behavior and recommendations of users and entire user communities implicit in bookmarks, query logs, and click streams, and collaborative search for advanced expert queries. In such a setting, queries would combine page scoring information from several peers that maintain different index lists. As in standard Web search, users often look only at the top-10 results. From our point of view, the common key feature of all such applications is that the data are distributed over a number of nodes at large scale and in an ad-hoc manner, and that this

*This paper is a summary of the PhD thesis "Top-K Aggregation Queries in Large-Scale Distributed Systems", by Sebastian Michel (Max-Planck-Institut für Informatik), Universität des Saarlandes, Saarbrücken, Germany, 2007.

data must be collected, and some aggregation function be applied, with the desired goal being the identification of the $k$ most relevant/interesting data items. We focus on efficient top-$k$ query algorithms in distributed environments.

### 1.1 Problem Statement and Computational Model

We consider a distributed system with $N$ peers, $P_j$, $j = 1, ..., N$, that are connected, e.g., by a distributed hash table or some overlay network. Data items are either documents such as Web pages or structured data items such as movie descriptions. Each data item has associated with it a set of descriptors, text terms or attribute values, and there is a precomputed score for each pair of data item and descriptor. The inverted index list for one descriptor is the list of data items in which the descriptor appears sorted in descending order of scores. These index lists are the distribution granularity of the distributed system. Each index list is assigned to one peer (or, if we wish to replicate it, to multiple peers).

The overall goal is to efficiently find the top-$k$ items (documents) or, in case of approximate algorithms, come as close as possible to the true top-$k$ results. We measure the quality of the approximate result by the fraction of documents in the approximate top-$k$ result that are also in the true top-$k$ result, i.e. the relative recall.

### 1.2 Contribution

- We present the KLEE algorithmic framework [MTW05a] as a fundamental building block towards efficient top-k query processing in distributed systems.
- We present techniques to model value distributions and show how these models can be used to reason about parameter values that play an important role in the overall performance of KLEE.
- We present the GRASS algorithmic framework. The GRASS algorithms come with three different kinds of optimization techniques [NBM+08]: we introduce a technique to efficiently leverage the knowledge of the input data characteristics to tune thresholds that are of fundamental importance. We show how hierarchical query plans can be generated using the aforementioned cost model to build optimal query execution plans that drastically increase the overall performance. We introduce a method to select a sample of input data sources that still provides reasonably accurate results but can be, at the same time, more efficiently handled. All these techniques result in a significantly increased overall performance.
- We present probabilistic guarantees for the aforementioned algorithms.
- Moreover, we address the issue of building a highly scalable search engine. We have developed Minerva$\infty$ [MTW05b], a scalable and efficient Peer-to-Peer Web search engine. The distinguishing feature of Minerva$\infty$ is the high distribution both in the data and computational dimensions. The key idea is to give up the nodes' autonomy and distribute each index list over multiple peers, as a key step towards a system with unlimited scalability. We expect that nodes will autonomously crawl the web, discovering *documents* and computing *scores* of documents, with each score reflecting a document's importance with respect to *terms* of interest.

## 2 Related Work

Top-k query processing has received much attention in a variety of settings such as similarity search on multimedia data [CGM04, CGM04, Fa99, FLN03, GBK00, BGRS99, NCS+01, dVMNK02], ranked retrieval on text and semi-structured documents in digital libraries and on the Web [AdKM01, TWS04, BJRS03], spatial data analysis [BBK01, CP02], network and stream monitoring [BO03, KOT04, CW04] collaborative recommendation and preference queries on e-commerce product catalogs [YPM03, MBG04, BGM02, GBK01], and ranking of SQL-style query results on structured data sources in general [ACDG03, CDHW04, BCG02]. [BCG02] addresses the mapping of top-k queries into range queries that can be handles by the query optimizer in a conventional RDBMS.

The first distributed TA-style algorithm has been presented in [BGM02, MBG04]. The emphasis of that work was on top-$k$ queries over Internet data sources for recommendation services (e.g., restaurant ratings, street finders). Because of functional limitations and specific costs of data sources, the approach used a hybrid algorithm that allowed both sorted and random access but tried to avoid random accesses. Scheduling strategies for random accesses to resolve expensive predicates were addressed also in [CwH02]. In our widely distributed setting, none of these scheduling methods are relevant as they still incur an unbounded number of message rounds. The method in [SMwW+03] addresses P2P-style distributed top-$k$ queries but considers only the case of two index lists distributed over two peers. Its key idea is to allow the two cohort peers to directly exchange score and candidate information rather than communicating only via the query initiator. Unfortunately, it is unclear and left as an open issue how to generalize to more than two peers.

In contrast, state-of-the-art algorithms for distributed top-$k$ aggregation use a fixed number of communication rounds to bound latency and aim to minimize the total network bandwidth consumption. The first algorithm in this family was the *TPUT* (Three-Phase Uniform Threshold) algorithm [CW04], in which a query coordinator, typically the network node which initiates the query, executes a three-phase distributed threshold algorithm. We will give a more detailed description of the algorithm in the following section. *TPAT* [YLW+05] is a modification of TPUT where the threshold, that is the same for all index lists, is adapted to the specifics of the value distributions; however, the authors state that their solution may incur infeasible computational cost. We will also consider the issue of adaptive thresholds and introduce an efficient way to calculate them.

### 2.1 The Three Phase Uniform Threshold Algorithm (TPUT)

The main idea of TPUT [CW04] is to transform the top-$k$ query into a range query where the range is determined via an estimation of the $min$-$k$ value.

1. $min$-$k$ **estimation phase:** The query initiator $P_{init}$ retrieves the top $k$ items from each of the input index-lists. Subsequently, $P_{init}$ calculates the worstscore for all observed items and ranks them accordingly. The worstscore is the aggregation of the actually observed values. Similarly, the bestscore denotes the best possible score an item can achieve in the future. The worstscore of the item currently at rank $k$ is denoted as $min$-$k$.

2. **Candidate retrieval phase:** Based on the $min\text{-}k$ estimation, TPUT sends the $min\text{-}k/m$ threshold to all involved peers that send back all $(itemId, score)$-pairs with $score \geq min\text{-}k/m$. This ensures that all candidates (potential members of the final top-$k$ result) have been found, in at least one of the lists. After the $min\text{-}k$ value has been recalculated, TPUT throws away all items with $bestscore < min\text{-}k$.
3. **Missing scores lookup phase:** For all the remaining candidates, TPUT looks up the missing scores by sending to each peer $P_i$ a list of the candidates that have not been seen in list $L_i$ so far. $P_{init}$ can now calculate the exact score for all candidates, i.e. the true top-$k$ results have been identified.

| **Phase 1** | | | **Phase 2** | | | **Phase 3** | | |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{(a, 12)}$ | $\mathbf{(b, 8)}$ | $\mathbf{(a, 17)}$ | $\mathbf{(a, 12)}$ | $\mathbf{(b, 8)}$ | $\mathbf{(a, 17)}$ | $\mathbf{(a, 12)}$ | $\mathbf{(b, 8)}$ | $\mathbf{(a, 17)}$ |
| $\mathbf{(b, 10)}$ | $\mathbf{(c, 7)}$ | $\mathbf{(z, 13)}$ | $\mathbf{(b, 10)}$ | $\mathbf{(c, 7)}$ | $\mathbf{(z, 13)}$ | $\mathbf{(b, 10)}$ | $\mathbf{(c, 7)}$ | $\mathbf{(z, 13)}$ |
| $(c, 8)$ | $(e, 6)$ | $(e, 11)$ | $\mathbf{(c, 8)}$ | $\mathbf{(e, 6)}$ | $\mathbf{(e, 11)}$ | $\mathbf{(c, 8)}$ | $\mathbf{(e, 6)}$ | $\mathbf{(e, 11)}$ |
| $(d, 6)$ | $(z, 4)$ | $(f, 10)$ | $\mathbf{(d, 6)}$ | $(z, 4)$ | $\mathbf{(f, 10)}$ | $\mathbf{(d, 6)}$ | $\mathbf{(z, 4)}$ | $\mathbf{(f, 10)}$ |
| $(e, 3)$ | $(m, 2)$ | $(c, 6)$ | $(e, 3)$ | $(m, 2)$ | $\mathbf{(c, 6)}$ | $\mathbf{(e, 3)}$ | $(m, 2)$ | $\mathbf{(c, 6)}$ |
| $(h, 3)$ | $(g, 2)$ | $(r, 5)$ | $(h, 3)$ | $(g, 2)$ | $(r, 5)$ | $(h, 3)$ | $(g, 2)$ | $(r, 5)$ |
| $(f, 2)$ | $(o, 1)$ | $(b, 5)$ | $(f, 2)$ | $(o, 1)$ | $(b, 5)$ | $(f, 2)$ | $(o, 1)$ | $\mathbf{(b, 5)}$ |

Table 1: Sample TPUT execution for a top-2 query: Phase 1 (left): Retrieve the top-2 items from each list. Phase 2 (middle): Retrieve all items with score above $min\text{-}k/3$ = 6. Phase 3 (right): Retrieve missing score via random lookups.

## 3 The KLEE Algorithmic Framework

TPUT ensures a small query response times using a small, fixed number of (only three) communication phases. We also adopt the requirement for a small number of communication phases. However, KLEE goes far beyond. The salient features and novel contributions of KLEE are the following:

- KLEE comes with two flavors, one involving only two and one involving three communication phases. It recognizes that the number of communication phases is only one aspect of guaranteeing short response times, which, in turn, is only one aspect of overall efficiency. In particular, as limited network and IO bandwidth appear to be key contributors to response times, KLEE ensures that significantly smaller messages are exchanged and that random IOs at participating peers are avoided, resulting in strong gains in response time and network bandwidth and lighter peer loads compared to TPUT.
- KLEE is the first to make a strong case for approximate top-$k$ algorithms for wide-area networks, showing how significant performance benefits can be enjoyed, at only small penalties in result quality.
- KLEE provides a flexible framework for top-$k$ algorithms, allowing for trading-off efficiency versus result quality and bandwidth savings versus the number of communication phases.
- We have implemented KLEE and a number of competing algorithms and conducted comprehensive experimental performance evaluation using real-world and synthetic data, which shows the consistent superiority of KLEE over its competitors.

- KLEE is equipped with various fine-tuning parameters and we provide a discussion of how these can be automatically adjusted to underlying data and system characteristics.

The proposed approach is based on having a per-query coordinator peer and a set of cohort peers. In our setting, the coordinating peer is the peer where the query was initiated, $P_{init}$. The cohort peers, are the peers storing the index lists, based on which the document scores will be computed. The algorithm is structured to proceed in a number of phases, with each phase consisting of a round-trip communication between the coordinator and the cohorts. In general, in each phase, the coordinator requests and receives from each peer a portion of the peer's local index information, which permits the coordinator to run a top-$k$ algorithm (such as the TA algorithm or variants) based on the collected information about the peers' index lists.

### 3.1 The HistogramBlooms Structure

In KLEE, each peer maintains a set of statistical metadata describing its index list. In particular, histogram-based information is maintained to describe the distribution of scores in the index list. The range of possible score values cover the range $(0, 1]$. For simplicity, we assume that peer histograms are equi-width, consisting of n cells, each cell being responsible for $(1/n)th$ of the score range. It would be straightforward to employ other forms of histograms. Associated with each cell $i$, each peer maintains the following information

- The lower and upper values, $lb[i]$, $ub[i]$, respectively, defining the range of scores being covered by this cell,
- The value of $freq[i]$, defining the number of document IDs whose scores in the peer's index list fall within $lb[i]$ and $ub[i]$,
- The average score, $avg[i]$, computed over all scores in the cell, and
- A synopsis of the document IDs whose scores fall in this cell, $filter[i]$. In particular, this compact representation is constructed using Bloom filters [Bl70].

In the first phase, at the coordinator's request, each cohort peer replies with its local top-$k$ list, and a fraction of its HistogramBlooms data structure. The coordinator then can address the missing-scores problem as follows: for every peer $P_i$ that has not reported a score for docID, using the Bloom-filter cell summaries of $P_i$ and the hash functions, it can find to which histogram cell of peer $P_i$ the docID belongs say $c$, (by simply testing for membership of docID in the filters of each cell, and stopping when a test is successful). Then, it can use the average score associated with that histogram cell, $avg[c]$, to replace the missing score of $P_i$ for docID.

With the additional knowledge of the documents' scores, the query initiator is able to compute a tighter $min\text{-}k/m$ score bound which leads to a decreased network consumption in the second communication phase as less documents qualify for the range query.

KLEE comes in addition use of an optional filtering phase for the data transmission in the second phase (cf., [MTW05a]).

## 4 Probabilistic Guarantees

TPUT is exact, i.e. it calculates the true top-$k$ result. KLEE [MTW05a] is an approximate version of TPUT that does not employ the third phase and, in addition, uses compression techniques based on Bloom filters and a technique to filter out unpromising candidates.

For now we assume that the compression techniques applied in KLEE are perfect, i.e. do not cause any false positives. It is straightforward to configure Bloom filters to provide this error-free behavior with high probability [Bl70]. So, we actually consider the approximate version of TPUT (without the third phase) and disregard KLEE's optimization techniques.

We consider the state of the algorithm after the second phase. All items with bestscore $< min\text{-}k$ have been pruned away. This is the standard pruning technique that does not introduce errors. The error in the relative recall at this stage of the algorithm is caused by the ordering of the items since this ordering is based on incomplete information (i.e., not fully evaluated candidate items). We assume that the item that is currently at rank $k+1$ has the highest chance to get into the top-$k$ result. In general, an item at rank $> k+1$ can have a higher chance to get into the top-$k$ than the item rank $k+1$ if it has been seen in fewer index lists than the item at rank $k+1$ and thus may achieve a higher final score. In our scenario it does not make sense to distinguish between the different index lists since for all of them the algorithm reads down to $min\text{-}k/m$ and we do not consider the score distributions in the tails of the index lists. Since we try to derive a general probabilistic guarantee, we cannot treat items individually. In this setting, assuming that the item at rank $k+1$ has the highest chance to get into the top-$k$ is a meaningful assumption.

With this assumption we have derived an upper bound for the probability that the item at rank-$k+1$ will get into the final top-$k$ result, hence can derive the expected result accuracy [Mi07].

## 5 Optimizing Top-k Queries

We consider the optimization of distributed top-$k$ queries in wide-area networks: We present GRASS, an algorithmic framework [NBM$^+$08] that consists of three optimization techniques.

- we introduce a technique to efficiently leverage the knowledge of the input data characteristics to tune score thresholds that are of fundamental importance. The basic KLEE method and related algorithms transform the top-$k$ retrieval problem into range-queries where the ranges are determined using uniform score thresholds. We propose the usage of non-uniform thresholds, and present an efficient optimization algorithm to adapt the threshold to the index-lists score distribution characteristics.
- we show how hierarchical query plans can be generated using the aforementioned cost model to build optimal query execution plans that drastically increase the overall performance. Consider, for example, a query with one very large and several small input lists residing on different peers. It would be better to perform the top-$k$ query at the peer with the large list, have the small peers ship their items to the large peer, and only send the final result to the query initiator. We use a dynamic programming approach that considers all possible query plans and chooses the cheapest plan w.r.t. our cost model.

- we introduce a sampling method to select a subset of the input data sources that still provides accurate results but can be, at the same time, more efficiently handled. We have performed experiments on real Web data that show the benefits of distributed top-$k$ query optimization both in network resource consumption and query response time.

## 6 Minerva Infinity

An architectural alternative to the computational model that underlies our algorithms is to distribute each index list over multiple peers, as a key step towards a system with unlimited scalability. We expect that nodes will autonomously crawl the web, discovering documents and computing scores of documents, with each score reflecting a document's importance with respect to terms of interest. This results in index lists, one for each term, containing relevant documents and their scores for a term. In a succeeding step, each peer distributes its set of *(docId, score, term)-triplets* across the participating peers. One way of doing so is to use a standard order-preserving hash function that assigns each triplet to a node based on the score's hash-value plus a term-specific offset. While this obviously distributes the triplets over the peers it will create a load imbalance because of the skewed (Zipf-like) score distribution typically observed in real-world index lists. To overcome this problem, we have developed a more sophisticated hash function that distributes index lists over the participating peers in a load-balancing and, at the same time, order-preserving way. But even with such a hash function it is infeasible to distribute a single index list over all peers, since this would cause a gigantic communication overhead as all peers would have to be contacted in order to retrieve the required information. To overcome this problem, we restrict the placement of the *(docid, score, term)-triplets* for a particular term to a subset of all peers. These small networks, called term index networks (TIN), help to limit the number of peers contacted during retrieval. In general, TINs can form separate overlay networks, but for simplicity we model a TIN simply as a (circular) doubly-linked list. The top-$k$ query processing proceeds in rounds, in which a coordinator peer retrieves batches of (docId, score, term)-triplets from the nodes that are part of the query-term specific TINs. We believe that our design choices are a big step towards a scalable P2P search engine. The Minerva$\infty$ [MTW05b] architecture has been implemented, and performance experiments have been conducted.

## 7 Conclusion

We have considered distributed top-$k$ algorithms, where the index lists for the query attributes are spread across multiple peers. We have presented the KLEE framework for distributed top-$k$ query processing. KLEE's salient features set it apart from related work in several ways. First, KLEE makes for the first time a strong case for approximate top-$k$ algorithms in widely distributed environments. Second, KLEE allows the trading-off of result quality vs performance. KLEE even allows for trading-off between bandwidth vs the number of communication phases. KLEE achieves great performance gains in network bandwidth, query response times, and local peer load, and high quality results. Moreover, we have considered means to model score distributions and how these score models can

be used to reason about parameter values that play an important role in the overall performance of KLEE. We have presented the family of GRASS algorithms that use these score models for optimizing the query execution. GRASS shows significant performance gains, in particular for queries that involve many data sources. We have derived probabilistic guarantees for the algorithms presented, to show both analytically and by a comprehensive experimental study their suitability for various application classes. Furthermore, we have presented Minerva∞, a novel architecture for a peer-to-peer web search engine. The key distinguishing feature of Minerva∞ is its high-levels of distribution for both data and processing.

## 8 Acknowledgment

I would like to express my gratitude to all those who gave me the possibility to complete my PhD thesis. I am deeply indebted to my supervisor Prof. Dr.-Ing. Gerhard Weikum, Max-Planck-Institut für Informatik, Saarbrücken. I have been very lucky being able to work with him and a number of other very nice and highly talented people during my PhD. In particular notable for their manifold contributions to this thesis, Prof. Dr. Peter Triantafillou, University of Patras, Greece, and Dr. Thomas Neumann, Max-Planck-Institut für Informatik, Saarbrücken.

## References

[ACDG03] Agrawal, S., Chaudhuri, S., Das, G., and Gionis, A.: Automated ranking of database query results. In: *1st Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, January 5-8, 2003*. 2003.

[AdKM01] Anh, V. N., de Kretser, O., and Moffat, A.: Vector-space ranking with effective early termination. In: Croft, W. B., Harper, D. J., Kraft, D. H., and Zobel, J. (Hrsg.), *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*. S. 35–42. ACM. 2001.

[BBK01] Böhm, C., Berchtold, S., and Keim, D. A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.* 33(3):322–373. 2001.

[BCG02] Bruno, N., Chaudhuri, S., and Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Database Syst.* 27(2):153–187. 2002.

[BGM02] Bruno, N., Gravano, L., and Marian, A.: Evaluating top-k queries over web-accessible databases. In: *Proceedings of the 18th International Conference on Data Engineering, ICDE, 26 February - 1 March 2002, San Jose, CA*. S. 369–. IEEE Computer Society. 2002.

[BGRS99] Beyer, K. S., Goldstein, J., Ramakrishnan, R., and Shaft, U.: When is "nearest neighbor" meaningful? In: Beeri, C. and Buneman, P. (Hrsg.), *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.* volume 1540 of *Lecture Notes in Computer Science*. S. 217–235. Springer. 1999.

[BJRS03] Bawa, M., Jr., R. J. B., Rajagopalan, S., and Shekita, E. J.: Make it fresh, make it quick: searching a network of personal webservers. In: Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003. S. 577–586. 2003.

[Bl70] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*. 13(7):422–426. 1970.

[BO03] Babcock, B. and Olston, C.: Distributed top-k monitoring. In: Halevy, A. Y., Ives, Z. G., and Doan, A. (Hrsg.), *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*. S. 28–39. ACM. 2003.

[CDHW04] Chaudhuri, S., Das, G., Hristidis, V., and Weikum, G.: Probabilistic ranking of database query results. In: Nascimento, M. A., Özsu, M. T., Kossmann, D., Miller, R. J., Blakeley, J. A., and Schiefer, K. B. (Hrsg.), *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. S. 888–899. Morgan Kaufmann. 2004.

[CGM04] Chaudhuri, S., Gravano, L., and Marian, A.: Optimizing top-k selection queries over multimedia repositories. *IEEE Trans. Knowl. Data Eng.* 16(8):992–1009. 2004.

[CP02] Ciaccia, P. and Patella, M.: Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.* 27(4):398–437. 2002.

[CW04] Cao, P. and Wang, Z.: Efficient top-k query calculation in distributed networks. In: Chaudhuri, S. and Kutten, S. (Hrsg.), *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*. S. 206–215. ACM. 2004.

[CwH02] Chang, K. C.-C. and won Hwang, S.: Minimal probing: supporting expensive predicates for top-k queries. In: Franklin, M. J., Moon, B., and Ailamaki, A. (Hrsg.), *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*. S. 346–357. ACM. 2002.

[dVMNK02] de Vries, A. P., Mamoulis, N., Nes, N., and Kersten, M. L.: Efficient k-nn search on vertically decomposed data. In: Franklin, M. J., Moon, B., and Ailamaki, A. (Hrsg.), *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*. S. 322–333. ACM. 2002.

[Fa99] Fagin, R.: Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.* 58(1):83–99. 1999.

[FLN03] Fagin, R., Lotem, A., and Naor, M.: Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.* 66(4):614–656. 2003.

[GBK00] Güntzer, U., Balke, W.-T., and Kießling, W.: Optimizing multi-feature queries for image databases. In: Abbadi, A. E., Brodie, M. L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., and Whang, K.-Y. (Hrsg.), *Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*. S. 419–428. Morgan Kaufmann. 2000.

[GBK01] Güntzer, U., Balke, W.-T., and Kießling, W.: Towards efficient multi-feature queries in heterogeneous environments. In: International Symposium on Information Technology (ITCC 2001), 2-4 April 2001, Las Vegas, NV, USA. S. 622–628. IEEE Computer Society. 2001.

[KOT04] Koudas, N., Ooi, B. C., Tan, K.-L., and 0003, R. Z.: Approximate nn queries on streams with guaranteed error/performance bounds. In: Nascimento, M. A., Özsu, M. T., Kossmann, D., Miller, R. J., Blakeley, J. A., and Schiefer, K. B. (Hrsg.), *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. S. 804–815. Morgan Kaufmann. 2004.

[MBG04] Marian, A., Bruno, N., and Gravano, L.: Evaluating top-*k* queries over web-accessible databases. *ACM Trans. Database Syst.* 29(2):319–362. 2004.

[Mi07] Michel, S.: *Top-K Aggregation Queries in Large-Scale Distributed System*. PhD thesis. Universität des Saarlandes. Saarbrücken, Germany. 2007.

[MTW05a] Michel, S., Triantafillou, P., and Weikum, G.: Klee: A framework for distributed top-k query algorithms. In: Böhm, K., Jensen, C. S., Haas, L. M., Kersten, M. L., Larson, P.-Å., and Ooi, B. C. (Hrsg.), *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. S. 637–648. ACM. 2005.

[MTW05b] Michel, S., Triantafillou, P., and Weikum, G.: $\text{Minerva}_{\text{infinity}}$: A scalable efficient peer-to-peer search engine. In: Alonso, G. (Hrsg.), *Middleware*. volume 3790 of *Lecture Notes in Computer Science*. S. 60–81. Springer. 2005.

[NBM+08] Neumann, T., Bender, M., Michel, S., Schenkel, R., Triantafillou, P., and Weikum, G.: Optimizing distributed top-k queries. In: Web Information Systems Engineering - WISE, 9th International Conference, Auckland, New Zealand, September 1-3. S. 337–349. 2008.

[NCS+01] Natsev, A., Chang, Y.-C., Smith, J. R., Li, C.-S., and Vitter, J. S.: Supporting incremental join queries on ranked inputs. In: Apers, P. M. G., Atzeni, P., Ceri, S., Paraboschi, S., Ramamohanarao, K., and Snodgrass, R. T. (Hrsg.), *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. S. 281–290. Morgan Kaufmann. 2001.

[SMwW+03] Suel, T., Mathur, C., wen Wu, J., Zhang, J., Delis, A., Kharrazi, M., Long, X., and Shanmugasundaram, K.: Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In: Christophides, V. and Freire, J. (Hrsg.), *WebDB*. S. 67–72. 2003.

[TWS04] Theobald, M., Weikum, G., and Schenkel, R.: Top-k query evaluation with probabilistic guarantees. In: Nascimento, M. A., Özsu, M. T., Kossmann, D., Miller, R. J., Blakeley, J. A., and Schiefer, K. B. (Hrsg.), *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. S. 648–659. Morgan Kaufmann. 2004.

[YLW+05] Yu, H., Li, H.-G., Wu, P., Agrawal, D., and Abbadi, A. E.: Efficient processing of distributed top- queries. In: Andersen, K. V., Debenham, J. K., and Wagner, R. (Hrsg.), *Database and Expert Systems Applications, 16th International Conference, DEXA 2005, Copenhagen, Denmark, August 22-26, 2005, Proceedings*. volume 3588 of *Lecture Notes in Computer Science*. S. 65–74. Springer. 2005.

[YPM03] Yu, C. T., Philip, G., and Meng, W.: Distributed top-n query processing with possibly uncooperative local systems. In: Freytag, J. C., Lockemann, P. C., Abiteboul, S., Carey, M. J., Selinger, P. G., and Heuer, A. (Hrsg.), *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*. S. 117–128. Morgan Kaufmann. 2003.

# Efficient adaptive retrieval and mining in large multimedia databases

Ira Assent
Data Mining and Data Exploration Group
RWTH Aachen University, Germany

**Abstract:** Multimedia databases are increasingly common in science, business, entertainment and many other applications. Their size and high dimensionality of features are major challenges for efficient and effective retrieval and mining. Effective similarity models are usually computationally far too complex for straightforward usage in large high dimensional multimedia databases.

We propose efficient algorithms for these effective models that show substantially improved scalability. Our index-based methods for efficient query processing and mining restrict the search space to task relevant data. Multistep filter-and-refine approaches using novel filter functions with quality guarantees ensure that fast response times are achieved without any loss of result accuracy.

## 1 Multimedia Databases

There is tremendous growth in multimedia data in application domains such as medicine, engineering, biology and numerous others. New technologies such as computer tomography or digital photography produce increasingly large volumes of data. At the same time, decreasing storage prices allow archiving large amounts of multimedia data at relatively low cost. For database technology, large multimedia databases require efficient and effective strategies for accessing and processing of multimedia data. Examples of typical multimedia content are illustrated in Figure 1: stock data time series, music, medical magnetic resonance images, shape and color images, as well as gene expression data.

Traditional database management systems have established themselves as reliable and powerful techniques for archiving and retrieving textual or numeric data. Traditional meta data information, however, is typically not adequate for multimedia applications. For example, users typically search for images not with respect to their size on hard disk, but with respect to content related features like color or shape. Consequently, multimedia database systems should provide content-based access. Likewise, aggregating the content of multimedia databases is not straightforward. Traditional reporting tasks query databases for (aggregated) numbers. Aggregating multimedia objects, or, more precisely, their respective feature values, in such a way is not meaningful. Knowledge discovery techniques aggregate multimedia data with respect to type of media and application focus such that expedient summaries are formed.

Multimedia databases thus should provide (1) content-based access, (2) knowledge discov-

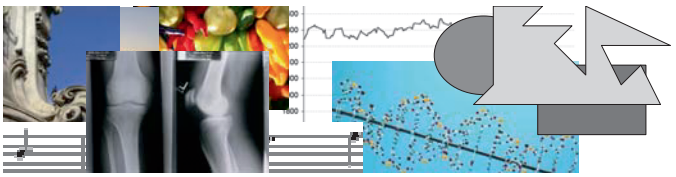


Figure 1: Examples of multimedia data

ery methods, (3) scalability to large data volumes, (4) scalability to high dimensionality of features, (5) good runtime performance.

In this work, we focus on efficient content-based retrieval and mining. Using multistep filter-and-refine architectures, our novel algorithms and lower bounding filters yield substantial runtime improvements without loss of accuracy. The general idea is illustrated in Figure 2. Efficient filters greatly reduce the number of objects relevant to a query to a small set of candidates. These candidates are then refined to the actual result set. Fulfillment of the ICES criteria ensures that the filter is indexable (works on multidimensional index structures), complete (no false dismissals), efficient (fast one-to-one filter comparison) and selective (small candidate sets). Moreover, we devise indexing techniques that greatly reduce database access for similarity search or for data mining.

## 2 Similarity search and retrieval

Content-based retrieval searches for similar multimedia objects to a query object. Figure 3 gives an example for a small image database. Content-based retrieval requires efficient algorithms on effective similarity models. Similarity models define the relevant characteristics of multimedia objects, for example color distribution in images, or pixel frequencies in shapes. Such histogram features can be compared via distance functions which assign a degree of dissimilarity to any pair of histograms.

The type of distance function is crucial for the effectivity of the similarity search process, as it models which multimedia objects are considered alike. For different types of features, different similarity models have been proposed. One of the simpler models that is commonly used for both histograms and time series is the Euclidean distance, a distance from the family of $L_p$ norms. Based on the differences of corresponding histogram bins or time series values, respectively, the overall dissimilarity between features is computed. Adaptable similarity models allow for incorporation of expert knowledge on application domain.

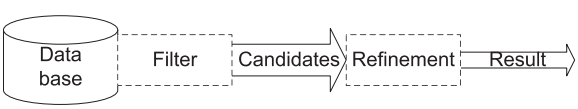



Figure 2: Multistep query processing

Figure 3: Similarity search

The Earth Mover's Distance, introduced in computer vision, computes an overall match on two (histogram) features based on a ground distance in feature space. Likewise, Dynamic Time Warping, originally from speech recognition, allows for stretching and squeezing of time series to match time series features. We propose new efficient retrieval methods for these models.

### 2.1 Histogram based search under the Earth Mover's Distance

Content-based similarity search requires effective similarity models as well as efficient query processing. Many multimedia features can be represented as histograms, i.e. vectors of attributes. For meaningful comparison of histogram features, distance functions provide dissimilarity values. Comparing image histograms using $L_p$-norms such as the Euclidean distance is very sensitive to minor shifts in feature value distribution.

A recent approach from computer vision towards human similarity perception, the Earth Mover's Distance (EMD) [RT01] models similarity as the amount of changes necessary to transform one image feature into another one. Formally, the Earth Mover's Distance is defined as a Linear Programming problem which can be solved using the simplex method as in [Dan51, HL90]. Its computation, however, is too complex for usage in interac-

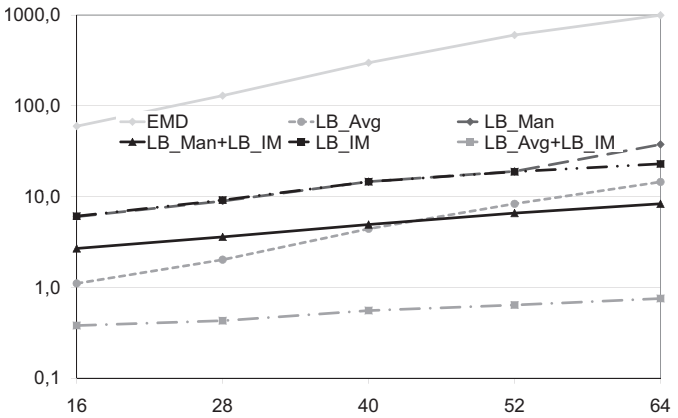



Figure 4: Dimensionality, response time in seconds

tive multimedia database scenarios. In order to enable efficient query processing in large databases, we propose an index-supported multistep algorithm. We therefore develop new lower bounding approximation techniques for the Earth Mover's Distance which satisfy high quality criteria including completeness (no false dismissals), index-suitability and fast computation. High dimensions are problematic for most index structures; we therefore use a dimensionality reduction implicit in the averaging lower bound or explicit in our Manhattan reduction. As this still leaves us with far too many candidates, two-phase filtering that combines the power of the individual filters, ensures an efficient retrieval procedure.

We demonstrate the efficiency of our approach in extensive experiments on large image databases. An important parameter for the performance of lower bounding approximations is the size of the histograms involved. On a database of 200,000 color images, we varied histograms from dimension 16 to 64. Figure 4 shows the corresponding response times. With increasing dimensionality, the computation of the existing $LB_{Avg}$ [RT01] increases in complexity. The response times of our $LB_{Man}$ are more closely related to its high selectivity ratios. The overhead of our novel, more complex, lower bound $LB_{IM}$ is greater than that of the other two lower bounds. Its combination with a low-dimensional $LB_{Avg}$-index yields the best performance improvements. We include the sequential scan Earth Mover's Distance computation as a baseline comparison. Note that the improvement for 64 dimensions comparing Earth Mover's Distance and the best multistep concept is from 1000 seconds to less than one second, i.e. more than three orders of magnitude. These efficiency gains do not come at the expense of accuracy. We prove completeness of our approach (no false dismissals), and refine using the exact Earth Mover's Distance (no false alarms) [AWS06]. For further approaches to efficient similarity search under the Earth Mover's Distance see also [AWMS08, AKS07, WAKS08].

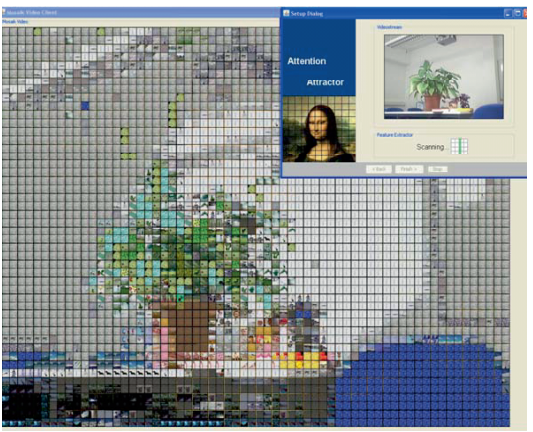



Figure 5: Similarity search video still

As a case study that demonstrates both the efficiency and effectiveness of this setup, we built a video streaming demonstration. A video camera records people passing by, and a monitor shows an alienated version, "mosaic" in real time. This system has been suc-

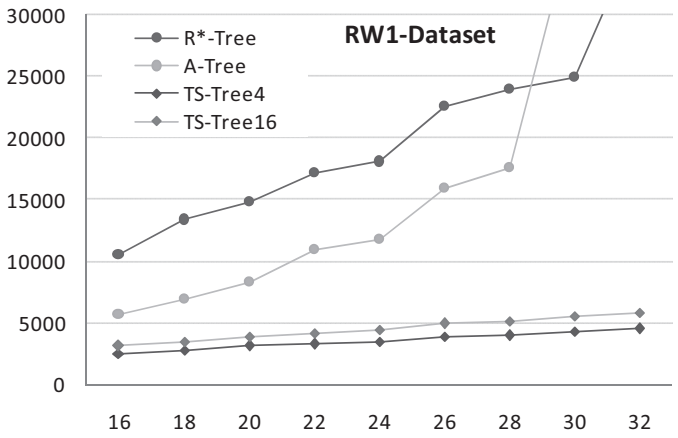



Figure 6: Dimensionality, number of page accesses for RW1 time series

cessfully used on several occasions to demonstrate state-of-the-art similarity search in a practical setting (see Figure 5) [AKS07].

### 2.2 Time Series Search and Retrieval

Continuous growth in sensor data and other temporal data increases the importance of retrieval and similarity search in time series data. Efficient time series query processing is crucial for interactive applications. Existing multidimensional indexes like the R-tree provide efficient querying for only relatively few dimensions [Keo02]. Time series are typically long which corresponds to extremely high dimensional data in multidimensional indexes. Due to massive overlap of index descriptors, multidimensional indexes degenerate for high dimensions and have to access the entire data by random I/O. Consequently, the efficiency benefits of indexing are lost.

We propose the TS-tree (time series tree), an index structure for efficient time series retrieval and similarity search. Exploiting inherent properties of time series quantization and dimensionality reduction, the TS-tree indexes high-dimensional data in an overlap-free manner. During query processing, powerful pruning via quantized separator and meta data information greatly reduces the number of pages which have to be accessed, resulting in substantial speed-up.

In thorough experiments on synthetic and real world time series data we demonstrate that our TS-tree outperforms existing approaches like the R*-tree or the quantized A-tree. As an example, we investigate the scalability and the query performance of the TS-tree on synthetic and real world data sets using the $L_2$ norm, i.e. Euclidean distance. Please not that our approach also works very well for Dynamic Time Warping (DTW) [BC94]. To analyze scalability, we report the number of pages read averaged over 25 nearest neighbor queries. The more the dimensionality of a time series data set is reduced the more time

series typically have to be refined. As they are usually very long, scalability in terms of dimensionality is very important. Figures 6 illustrates the results for RW1 data (synthetic random walk data of size 250,000), for time series length 256, reduced to dimensionality 16 to 32 using piecewise aggregate approximation. The TS-tree scales very well since its overlap-free separator split is not impaired by effects of high dimensional data spaces ("curse of dimensionality"). The R*-tree and A-tree, however, degrade with increasing dimensionality. Overall, the TS-tree outperforms both A-tree and R*-tree by nearly one order of magnitude [AKAS08].

# 3 Knowledge discovery in databases

Knowledge discovery in databases (KDD) extracts novel, potentially useful patterns from databases for generation of knowledge about the database content as a whole. The KDD process, depicted in Figure 7, consists of four steps from the original raw data to the actual knowledge generation [HK01]. In the first step, the data is integrated and cleaned of potential errors, then it is transformed and projected to the task relevant parts of the data. The central step, the data mining step, extracts patterns from this task relevant data, which is then visualized for human evaluation. In this work, we focus on automatic aggregation of the data into meaningful groups, the so-called clustering. Clustering groups objects such that similar ones are within the same group, whereas dissimilar ones are in different groups. We study efficient and effective clustering for high dimensional large databases.

## 3.1 Subspace clustering

In high-dimensional data, clusters are typically concealed by irrelevant attributes and do not show across the full space. Subspace clustering mines clusters hidden in subspaces of high-dimensional data sets. Density-based approaches have been shown to successfully mine clusters of arbitrary shape even in the presence of noise in full space clustering [EKSX96]. Direct extensions of density-based approaches to subspace clustering, how-

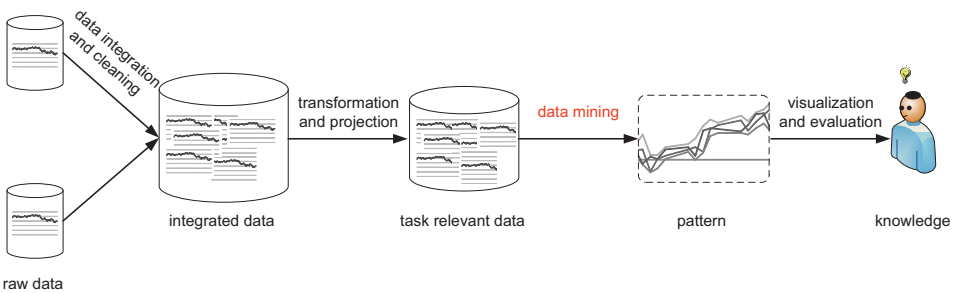



Figure 7: The knowledge discovery process

| | Objects | Dimensions | Classes | Source |
|---|---|---|---|---|
| Pendigits | 7494 | 16 | 10 | [AN07] |
| Vowel | 990 | 10 | 11 | [AN07] |
| Glass | 214 | 9 | 6 | [AN07] |
| Shapes | 160 | 17 | 9 | [KWX+06] |

Table 1: Real world data sets

ever, entail runtimes which are infeasible for large high-dimensional data sets, since the number of possible subspace projections is exponential in the dimensionality of the data. Consequently, existing subspace clustering algorithms trade-off efficiency for accuracy.

We propose lossless efficient detection of subspace clusters via a new density-conserving grid data structure which benefits from efficiency gains without losing any clusters. By detecting clusters in a depth-first manner, our EDSC (efficient density-based subspace clustering) algorithm avoids excessive candidate generation.

In thorough experiments on synthetic and real world data sets, we demonstrate that our lossless approach yields accurate clusters and is faster than existing subspace clustering algorithms by orders of magnitude. Real world data used in our experiments is characterized in Table 1. We evaluate the quality of EDSC by determining how accurate the hidden structure of the data given by the classes is identified. For each data set efficiency and accuracy of each algorithm is presented in Table 2. We can see that the EDSC algorithm shows the best quality for all real world data sets. Compared with SUBCLU [KKK04] the EDSC algorithm shows significantly better runtimes. The grid-based SCHISM [SZ04] shows not only worse quality in all data sets but in some also higher runtimes. In the shape data set SCHISM only finds 2-dimensional clusters and thus cannot be compared with the other subspace clustering algorithms which also detect the higher dimensional clusters. Our experiments on large and high-dimensional synthetic and real world data sets show that EDSC outperforms recent subspace clustering algorithms by orders of magnitude while maintaining superior quality [AKMS08].

| | EDSC | | SCHISM | | SUBCLU | |
|---|---|---|---|---|---|---|
| | F1 | time [s] | F1 | time [s] | F1 | time [s] |
| Pendigits | 48% | 5743 | 25% | 5278 | 24% | 43825 |
| Vowel | 35% | 47 | 21% | 88 | 17% | 310 |
| Glass | 57% | 3 | 47% | 8 | 52% | 10 |
| Shape | 51% | 34 | 2% | 0.2 | 40% | 1079 |

Table 2: Quality and runtimes on real world data

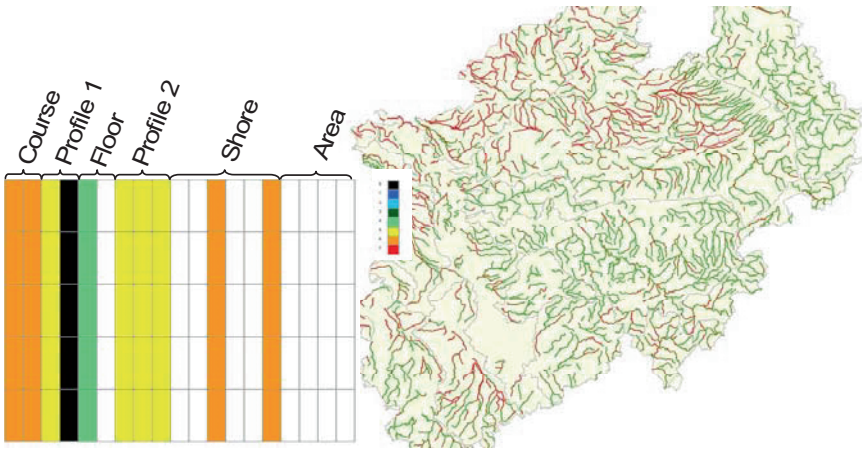



Figure 8: Cluster visualization for river data

### 3.2 MultiClustering

Environmental sensors produce data streams at successive time points which are often archived for further analysis. Applications like stock market analysis or weather stations gather ordered time series of different values in large temporal databases. Weather stations for example use multiple sensors to measure e.g. barometric pressure, temperature, humidity, rainfall. Many other scientific research fields like observatories and seismographic stations archive similar spatial or spatio-temporal time series.

In a current project of the European Union on renaturation of rivers, the structural quality of river segments is analyzed. For German rivers, about 120,000 one-hundred-meter segments were evaluated according to 19 different structural criteria, e.g. quality of the riverbed [LUA03]. They were mapped to quality categories. The sequence order of the segments is given by the flowing direction of the rivers. Hydrologists have devised packages of measures for different structural damages, formalized in rules specifying the attribute value constraints and the attributes influenced positively. An example constraint might be that a certain segment has good quality riverbed and riverbanks and poor river bending. This could be improved by measures like adding deadwood to positively influence river bending. Finding and analyzing these patterns helps hydrologists summarize the overall state of rivers, give compact representations of typical situations and review the extent to which these situations are covered by measures envisioned. They can identify those patterns which are problematic, i.e. have low quality ratings, but are not yet covered by measures.

From a computer science point of view, finding the intrinsic structure of these multidimensional time series is a two-fold task: (1) detect frequent patterns within time series for all possible pattern lengths, (2) then detect parallel occurrences of these patterns.

Patterns are ranges of values (which correspond to several categories of river quality struc-

ture) found in several (sub-)time series. Pattern analysis has to take into account that the data is subjective and fuzzy, because structural quality of rivers was mapped by different individuals. Our approach is based on weighting by kernel densities in a density-based clustering approach. This effectively detects fuzzy time series patterns. These patterns are clustered efficiently for arbitrary lengths using monotonicity properties. We transform these time series pattern clusters into a cluster position space such that mining parallel patterns can be reduced to efficient $FP$-tree frequent itemset mining.

Multiclusters are visualized in a geographical information system, as e.g. in Figure 8. Those river segments which are part of the cluster are marked by dark lines. The corresponding cluster summary is depicted on the left side. It indicates the cluster length of five river sections (top to bottom) as well as the cluster range of ten out of nineteen attributes (left to right). A more detailed discussion can be found in [AKGS06, AKGS08].

## 4 Conclusion

In this work, we propose efficient and effective retrieval and mining of multimedia data for both histogram and time series data. Key techniques of our work are database methods. Indexing organizes the data such that during query processing or mining, only relevant parts of the data have to be accessed to allow for efficient runtimes. Coupled with multistep filter-and-refine architectures, we are able to further enhance runtime performance by fast generation of candidate sets in a filter step that is refined to ensure neither false negatives nor false positives.

## References


[AKAS08] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The TS-Tree: Efficient Time Series Search and Retrieval. In *Proceedings of the 11th International Conference on Extending Data Base Technology (EDBT), Nantes, France*, 2008.

[AKGS06] Ira Assent, Ralph Krieger, Boris Glavic, and Thomas Seidl. Spatial Multidimensional Sequence Clustering. In *Proceedings of the 1st International Workshop on Spatial and Spatio-temporal Data Mining (SSTDM), in conjunction with the 2006 IEEE International Conference on Data Mining (ICDM)*, 2006.

[AKGS08] Ira Assent, Ralph Krieger, Boris Glavic, and Thomas Seidl. Clustering Multidimensional Sequences in Spatial and Temporal Databases. *International Journal on Knowledge and Information Systems (KAIS)*, 2008.

[AKMS08] Ira Assent, Ralph Krieger, Emmanuel Müller, and Thomas Seidl. EDSC: Efficient Density-Based Subspace Clustering. In *Proc. ACM 17th Conference on Information and Knowledge Management (CIKM), Napa Valley, USA*, pages 1093–1102, 2008.

[AKS07] Ira Assent, Ralph Krieger, and Thomas Seidl. AttentionAttractor: Efficient video stream similarity query processing in real time. In *Proceedings of the IEEE 2007 International Conference on Data Engineering (ICDE)*, pages 1509–1510, 2007.

[AN07] Arthur Asuncion and David Newman. University of California Machine Learning Repository, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2007.

[AWMS08] Ira Assent, Marc Wichterich, Tobias Meisen, and Thomas Seidl. Efficient similarity search using the Earth Mover's Distance for large multimedia databases. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE), Cancun, Mexico*, 2008.

[AWS06] Ira Assent, Andrea Wenning, and Thomas Seidl. Approximation Techniques for Indexing the Earth Mover's Distance in Multimedia Databases. In *Proceedings of the 2006 IEEE International Conference on Data Engineering (ICDE)*, 2006.

[BC94] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 229–248, 1994.

[Dan51] George Dantzig. *Application of the simplex method to a transportation problem*, pages 359–373. John Wiley and Sons, 1951.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregano, 1996. AAAI Press.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HL90] Frederick Hillier and Gerald Lieberman. *Introduction to Linear Programming*. McGraw-Hill, 1990.

[Keo02] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, pages 406–417. VLDB Endowment, 2002.

[KKK04] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-Connected Subspace Clustering for High-Dimensional Data. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 246–257, 2004.

[KWX+06] Eamon Keogh, Li Wei, Xiaopeng Xi, Sang-Hee Lee, and Michail Vlachos. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 882–893, 2006.

[LUA03] LUA NRW. River quality data. http://www.lua.nrw.de, 2003.

[RT01] Yossi Rubner and Carlo Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer Academic Publishers, 2001.

[SZ04] Karlton Sequeira and Mohammed Zaki. SCHISM: A New Approach for Interesting Subspace Mining. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 186–193, 2004.

[WAKS08] Marc Wichterich, Ira Assent, Philipp Kranen, and Thomas Seidl. Efficient EMD-based Similarity Search in Multimedia Databases via Flexible Dimensionality Reduction. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), Vancouver, BC, Canada.*, pages 199–212, 2008.

# Continuous Queries over Data Streams – Semantics and Implementation

Jürgen Krämer

Philipps-Universität Marburg
kraemerj@informatik.uni-marburg.de

**Abstract:** Recent technological advances have pushed the emergence of a new class of data-intensive applications that require continuous processing over sequences of transient data, called data streams, in near real-time. Examples of such applications range from business activity monitoring and online analysis of sensor data to trend detection in stock ticker data. This work presents a solid and powerful foundation for processing continuous queries over data streams. We (i) define a sound semantics for continuous sliding window queries, (ii) introduce a unique stream algebra implemented with efficient online algorithms, and (iii) sketch the adaptive runtime environment. Instead of adapting view maintenance techniques, our work carries over and enhances findings from temporal databases to meet the challenging requirements of the data stream computation model.

## 1 Introduction

Traditional database systems have proven to be well-suited to the organization, storage, and retrieval of finite datasets. In recent years, however, new data-intensive applications have emerged that need to process data which is continuously arriving at the system in the form of potentially unbounded, time-varying sequences of data items, termed *data streams*. Examples belonging to this new class of stream-oriented applications can be found in a diversity of application domains including sensor monitoring, finance, transaction log analysis, and network security [SCZ05, GÖ03b, BBD+02]:

- *Sensor networks* are used in a variety of applications for monitoring physical or environmental conditions, for instance, in traffic management, location tracking, supply chain management based on the upcoming RFID technology, medical monitoring, and manufacturing processes. Queries may involve the detection of unusual and often complex conditions, even across multiple streams, with the aim to activate alarms or trigger actions.
- *Network traffic management* involves online monitoring and analysis of network packets to find out information on traffic flow patterns for routing system analysis, bandwidth usage statistics, and network security. In particular, intrusion detection requires real-time responses, for instance, to prevent denial-of-service attacks.

- *Electronic trading* often relies on the online analysis of financial data obtained from stock tickers and news feeds. Specific goals include discovering correlations, identifying trends, and forecasting stock prices. More current results maximize arbitrage profits.
- *Transaction logging* is performed in many applications, generating huge volumes of data, for example, web server logs and click streams, telephone call records, user account logging, or event logs in online auction systems such as eBay. Queries over these streams of log entries could be used to initiate immediate actions on specific customer behavior, detect suspicious access patterns that could indicate fraud or attacks, or identify performance bottlenecks with the aim to improve service reliability.

In recent years a dramatic escalation in feed volumes has been observable due to steady technological progress. For example, every day WalMart records 20 million sales transactions, AT&T generates 275 million call records, and eBay logs 10 million bids. Market data feeds can even generate multiple thousands of messages per second. While storing these massive data sets is possible to a certain extent, extracting valuable information from the resultant histories is often extremely expensive because the overwhelming data volumes accumulated over months or even years cannot be searched and analyzed in acceptable time. As a consequence, the archived data is often discarded after the retention time has elapsed, without any prior analysis but to free space for new data. Even in applications with less restrictive time requirements it is thus advisable to pre-process data streams in order to extract relevant information and reduce data volumes.

**Data Stream Management Systems (DSMSs)** Traditional database systems are not designed to support high-volume, low-latency stream processing because queries are expected to run continuously and return new answers as new data arrives, without the need to store data persistently [BBD+02, GÖ03a]. For this reason, novel techniques and systems dedicated to the challenging requirements in stream processing have been developed.

In our work, we restrict the evaluation of continuous queries to *sliding windows* over the potentially unbounded data streams to keep resource requirements in bounds. This technique has the advantage that it emphasizes new data, which in the majority of real-world applications is considered more important than older data.

Rather than focusing on isolated research problems such as scheduling strategies or individual stream operators, our main objective has been to develop a general-purpose query engine for data streams. As part of our research, we have spent significant time and effort in the implementation of our Java library for data stream management called *PIPES* (*Public Infrastructure for Processing and Exploring Streams*) to demonstrate the feasibility and practical applicability of our methods. With PIPES, we pursue a library approach rather than building a monolithic system because we are convinced that it is almost infeasible to develop a general-purpose and still highly efficient DSMS for the plethora of streaming applications [KS04]. PIPES provides powerful and generic building blocks for the construction of a fully functional DSMS which can be tailored to the specific needs of an

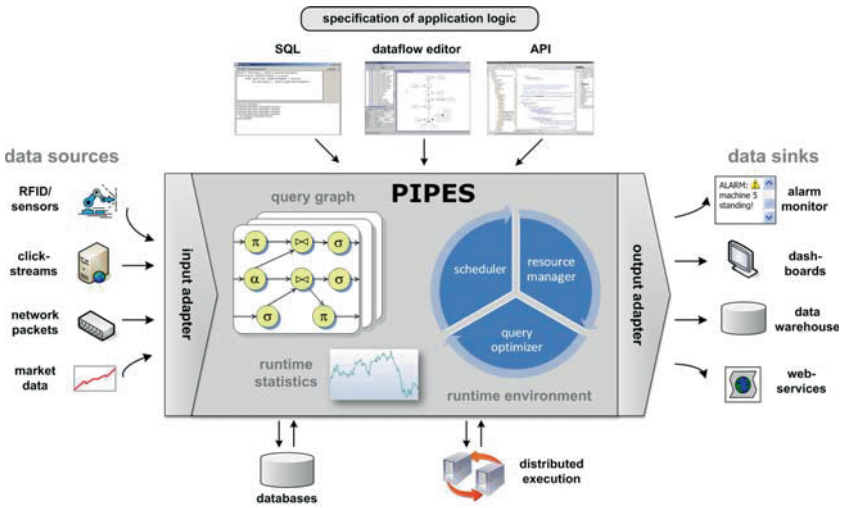



Figure 1: Architectural overview of our stream processing infrastructure PIPES

application domain. The following sections sketch its query language, semantic foundations, algorithms and implementation concepts, and its adaptive runtime environment (see Fig. 1).

## 2 Query Formulation

Today, many applications dealing with continuous queries over data streams rely on hand-coded queries to directly process elements at delivery. Custom coding typically entails high development and maintenance costs and it is almost impossible to optimize queries automatically. PIPES overcomes these drawbacks by offering two ways to express complex application logic in an easy and comprehensible manner: (i) via a *graphical user interface* that enables users to specify data flow by combining operators from a stream algebra in a *procedural* manner, or (ii) via a *declarative query language* inheriting the basic syntax from the prevalent SQL standard.

As SQL is meant for one-time queries over relations and not for continuous queries over data streams, we basically enhanced the `FROM` clause with a `WINDOW` specification to express sliding window queries. Our grammar reuses and extends a subset of the window constructs definable in SQL:2003 for OLAP functions. Our query language is at least as expressive as the *Continuous Query Language (CQL)* [ABW06], while staying closer to SQL:2003 standards (see [Krä07] for a formal proof). It even allows users to define and include derived streams as well as complex nested queries with aggregates or quantifiers.

**Example:** *Pair trading strategy* – Notify me if the average ratio of the prices of stock AG1 and AG2 in the last 10 minutes deviates from the common ratio of 5 to 3 by more than 10%.

```
SELECT *
FROM (SELECT AVG(AG1.price/AG2.price) AS ratio
      FROM   AG1 WINDOW(RANGE 10 MINUTES),
             AG2 WINDOW(RANGE 10 MINUTES)) AS SUBQUERY
WHERE ABS(SUBQUERY.ratio - 5/3) > 0.1*5/3;
```

## 3 Query Semantics

Query formulation only makes sense if the query semantics is precisely defined. To guarantee predictable and repeatable query results, continuous query semantics should be defined independent of how the system operates internally. Unlike traditional one-time queries, continuous queries produce output over time. Therefore, it is important to incorporate the *notion of time* into the semantics. Since queries expressed in our query language are translated into a query plan composed of operators from our *stream algebra*, the semantics of a query results from the semantics of the contributing operators.

**Logical Stream Algebra** In analogy to traditional database systems, we distinguish between a logical and physical operator algebra. Our *logical algebra* precisely defines the semantics of each operation in a direct way by modeling data streams as temporal multisets. In order to preserve the well-understood semantics of the relational algebra to the extent possible, our algebra contains a stream-counterpart for every operator in the extended relational algebra, except for sorting. Each of these, so-called *standard operators* is *snapshot-reducible*. We denote a stream operation $op_S$ with inputs $S_1, \ldots, S_n$ as snapshot-reducible to its relational counterpart $op_R$ if, for every time instant $t$, the snapshot at $t$ of the results of $op_S$ is equal to the results of applying $op_R$ to the snapshots of $S_1, \ldots, S_n$ at time instant $t$. A *snapshot* of a stream at time $t$ can be considered as a relation since it represents the multiset of all tuples valid at this instant. Fig. 2 illustrates the temporal concept of snapshot reducibility [SJS01].

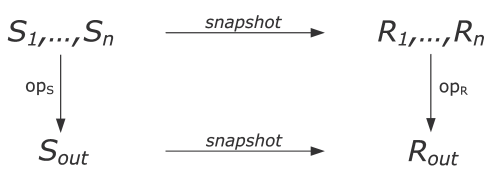



Figure 2: Snapshot-reducibility

In addition, our algebra provides novel *window operators* to define the scope of operations. Rather than integrating windows directly into operators, we separated the functionalities to avoid redundancy and facilitate the exchange of window types. The combination of window operators with our stream variants of the relational operators creates their windowed counterparts. The input of operators can be the output of an arbitrary operator plan. Overall, our logical algebra assigns an exact meaning to any continuous query at any point in time. As our logical algebra reveals the temporal properties of stream operations, it represents a valuable tool for exploring and validating equivalences.

**Plan Generation** Parsing and translating a query from its textual representation into a logical plan closely resembles query plan construction in conventional database systems. Whenever the parser identifies a window expression following a stream reference, the corresponding window operator is installed in the query plan downstream of the node standing for the stream reference.

Figure 3: Operator plan for pair trading query

**Example:** Fig. 3 demonstrates the query plan for the pair trading example query, using the following operators: time-based window ($\omega^{time}$), Cartesian product $\times$, map $\mu$, scalar aggregation $\alpha$, and filter $\sigma$.

**Query Optimization** We denote two query plans *equivalent* if they generate *snapshot-equivalent* results, i. e., if for any time instant $t$ their snapshots are equal. Due to defining our standard operations in semantic compliance with [SJS01], the well-known transformation rules such as join reordering and selection push-down carry over from conventional and temporal databases to stream processing. Here, semantic compliance means that our stream operators generate *snapshot-multiset-equivalent results* to the temporal operators presented in [SJS01]. The conventional transformation rules are applicable to any snapshot-reducible subplans of the query graph, i. e., all contiguous plans composed of standard operators. For a snapshot-reducible subplan, any optimizations with conventional transformation rules lead to an equivalent plan.

In [Krä07], we enrich this extensive set of *algebraic equivalences* with novel rules for the window constructs. Besides these logical optimizations, we investigate several *physical optimizations* dealing with expiration patterns, stream ordering, and time granularity. We identify vital properties of subplans that enable the system to apply these optimizations. Altogether, our approach paves the way for powerful query optimizations. We refer the interested reader to [KS09] for further information.

## 4 Stream Algorithms

The data stream model introduces new challenges for the implementation of queries. First, algorithms no longer have random access to their input, only *sequential* access. Second, algorithms that need to store some state information from stream elements that have been seen previously, e. g., the join and aggregation, must be computable within a *limited amount of space*, while the streams themselves are unbounded. Third, some implementations of relational operators are blocking. Since streams may be infinite, blocking operators cannot be applied in the stream computation model because they would produce no output. Fourth, algorithms should process incoming elements *on-the-fly* and generate

output continuously over time. This implies that amortized processing time per stream element should be kept small.

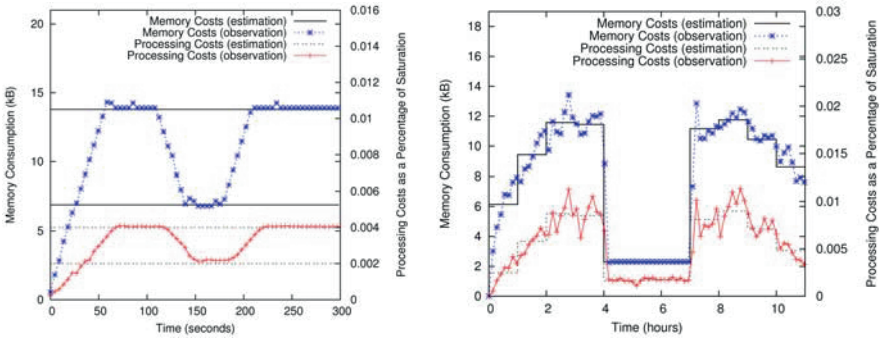
**Physical Stream Algebra** Our *physical stream algebra* reveals the implementation of continuous queries. It consists of push-based, nonblocking, stream-to-stream algorithms that produce results. Although results are semantically equivalent to the logical algebra, our physical algebra considers streams as sequences of tuples tagged with time intervals non-decreasingly ordered by start timestamps. Window operators set the lifespan of tuples according to the type and size of the window. Hence, the time interval of a stream element indicates the validity of the tuple according to the specified windows. Placing window operators upstream of stateful operators in a query plan avoids blocking and restricts resource requirements because tuples expire from the state.

While stateless operators such as selection or map do not consider the associated time intervals and thus can deal with potentially infinite windows well, time intervals affect stateful operators as follows. Elements are relevant to a stateful operator as long as their time interval may overlap with the time interval of any future stream element. This also means that a stateful operator can purge those elements from its state whose time interval cannot intersect with the time interval of any incoming stream element in the future. For operator state maintenance, our algorithms make use of generic sweepline data structures that allow for efficient probing and eviction of stream elements. See [Krä07, KS09] for details.

**Example:** The physical *join* operator has to satisfy the following two conditions to be snapshot-reducible: (a) for two contributing stream elements, the join predicate has to be satisfied for their tuples and (b) their time intervals have to intersect. The time interval assigned to the join result corresponds to the interval intersection. Our stream algorithm adapts the ripple join technique [HH99] to push-based query processing. The join state consists of two sweepline data structures, one for each input, and a min-priority queue at the output. For each incoming element, (i) expired elements are purged from the opposite data structure, (ii) the incoming element is inserted into the own data structure, (iii) join results are determined by probing the opposite data structure and inserted into the priority queue, and (iv) the minimum of the priority queue is extracted and appended to the output stream as long as the start timestamp ordering requirement of the output stream cannot be violated.

**Positive-Negative Approach** To the best of our knowledge, our time-interval approach (TIA) is unique in the stream community. Related work embracing similar semantics implements the positive-negative approach (PNA) which propagates different types of elements, namely positive and negative ones tagged with a timestamp, through an operator plan to control expiration [ABW06, GHM+07]. The window operators are stateful and generate the positive and negative elements according to their semantics. As a consequence, stream rates are doubled which means that twice as many elements as in the time-interval approach have to be processed by operators in general.

Our implementation comparison and experimental studies conclude that TIA has definite




(a) Changing the window sizes

(b) Changing the time granularity

Figure 4: Effects of adjustments to window sizes and time granularity on resource usage

advantages for *time-based* sliding window queries, which is the most common type of query in streaming applications. Because TIA does not have to handle negative elements, TIA roughly outperforms PNA by a factor of two. Due to the stateless time-based window operator, TIA needs to allocate less memory than PNA. For queries composed of stateless operators, join, and duplicate elimination, results for TIA are available at the same time as the positive result is propagated for PNA. For grouping, aggregation, and difference, however, TIA has a slightly delayed response time, because operators cannot emit results as long as a temporal overlap with future elements is possible. The latency advantage for PNA also applies to queries with *count-based* or *partitioned windows*. However, it is questionable to which extent these windows are used in practice because of their often ambiguous semantics. See [Krä07] for details.

The fundamental difference between PNA and TIA is the way windowing constructs are implemented. TIA uses time intervals to model validity like in temporal databases, whereas PNA originates from the maintenance of materialized views.

## 5 Adaptive Query Execution

**Adaptive Resource Management** Due to the long-running nature of queries, DSMSs need to control their resources adaptively since stream characteristics and query workload may vary over time. Our approach to *adaptive resource management* adjusts window sizes and time granularities to keep system resource usage within bounds [CKSV08]. Extensions to our query language allow the user to specify *quality of service (QoS) ranges* to restrict window sizes and time granularities. For time-based sliding window queries it is apparent that there is a direct correlation between window sizes and the memory usage of stateful operators. Because stateful operators have to keep all valid elements in their state, the state becomes larger the longer the validity is, and smaller the shorter the validity is (see Fig. 4(a)). The changes also affect CPU load, because the smaller the state the faster becomes element retrieval and eviction.

An advantage of this technique is that, even for small windows and high system load, it preserves those results computed from elements that are temporally close to each other, which is just the basic idea of sliding windows [GÖ03b]. Our second QoS constraint refers to the time granularity at which results, especially aggregates, are computed. For a lot of real-world queries, the user neither needs nor wants results at finest time granularity. Increasing the granule size associated with a stream in our physical algebra affects any downstream operators, particularly scalar aggregation and grouping with aggregation. The effects include savings in memory allocation and computational costs but also reductions to stream rates. Decreasing the granule size inverts the effects (see Fig. 4(b)).

Based on current runtime statistics, the resource manager has to adjust the window sizes and time granularities to keep the overall resource utilization within bounds. For this purpose, its objective function is to maximize overall QoS while keeping the individual settings for window and granule size within the pre-defined QoS ranges.

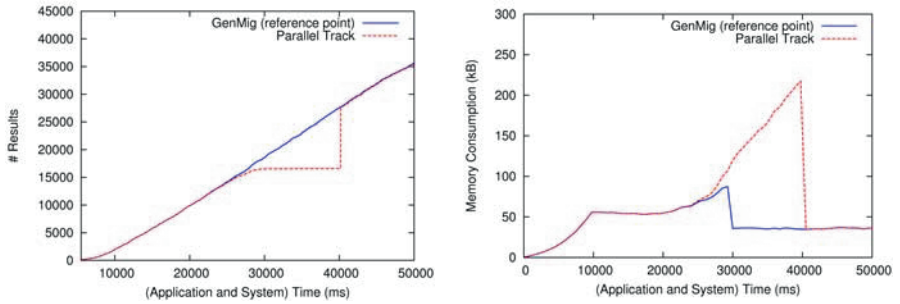
In contrast to random load shedding [TCZ+03], the main advantage of these methods is that the system can give *reliable guarantees* on query answers. Because query results are exact with regard to the selected QoS settings, adaptations do not collide with query optimization at runtime and consequently increase the flexibility and scalability of a DSMS.

**Cost Model** Our extensive cost model for the physical algebra allows for the *estimation of resource consumption* at operator-, query-, and system-level based on *stream characteristics* [CKSV08]. The stream characteristics involve the average inter-arrival time, the average validity of tuples, and the granule size. Based on our cost functions for the individual physical operators, the average memory costs, i. e., the amount of memory allocated for state maintenance, and the average processing costs per unit in application time can be estimated under acceptable system load at steady state. The resource utilization of a query plan is computed by summing up the individual costs of the involved operators.

Such a cost model is crucial to any adaptive runtime component in a DSMS because it can be used to quantify the effects of changes to query plans on the resource usage in advance. On the one hand, our resource manager employs the cost model to estimate the effects of adaptation techniques on query plans in order to determine to which extent window and granularity sizes need to be adjusted to keep resource usage in bounds. On the other hand, our query optimizer uses the cost model to select the best execution plan, namely the plan with the lowest estimated cost, from a set of semantically equivalent plans. Furthermore, the cost model enables the system to decide whether sufficient resources are available to accept a new query or not.

Our extensive experimental studies on real-world and synthetic data streams prove the accuracy of our cost model and the effectiveness of our adaptation techniques (see Fig. 4), even for experiments on larger settings where thousands of continuous queries were processed concurrently over several hours (see [CKSV08]).

**Plan Migration** PIPES is able to exploit *query optimization* with the aim to save system resources and improve scalability. Besides possibilities to optimize queries prior to execution and to share the execution of queries with common subexpressions, it provides




(a) Output stream characteristics (b) Memory usage

Figure 5: Comparison of plan migration strategies; 4-way nested-loops join switched from left-deep to right-deep join tree at runtime

methods for *dynamic query optimization* [YKPS07, KYC+06]. *GenMig*, our novel *plan migration strategy*, performs a gradual transition from the old plan, which has become inefficient over time due to changes in the data characteristics, to the new, more efficient plan at runtime (see Fig. 5(a)).

The core idea of our approach is to define an appropriate *split time* representing an explicit point in application time that separates the processing of the old and new plan. Stream elements delivered by the input streams are assigned to the plans as follows. For all tuples valid at time instants prior to the split time results are computed by the old plan, otherwise by the new plan. The migration is finished as soon as the time progression in any input stream reached the split time.

Because GenMig treats the old and new plans as snapshot-equivalent black boxes, the complete set of transformation rules holding in our stream algebra becomes applicable for optimization purposes. This is a major advance over the existing parallel-track technique [ZRH04], which is limited in its scope to plans with stateless operators and joins. Furthermore, our experiments confirm that GenMig exhibits a smoother output rate, lower memory and CPU overhead and finishes migration earlier (see Fig. 5(b)).

## 6 Conclusions

Our work tackles one of the most challenging issues in continuous query processing, namely, the development of a coherent and efficient stream processing engine whose powerful processing primitives derive from suitable extensions of the relational algebra. In addition to our theoretical results that establish a semantically sound and powerful foundation for data stream management, we have also pointed out the practicability of our work by successfully applying PIPES to various application domains including traffic management, online auctions, sleep medicine, and factory automation. As the data stream model finds widespread use in modern applications, stream processing engines like PIPES are expected to gain considerable popularity.

## References


[ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15(2):121–142, 2006.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *PODS*, pages 1–16, 2002.

[CKSV08] Michael Cammert, Jürgen Krämer, Bernhard Seeger, and Sonny Vaupel. A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems. *IEEE TKDE*, 20(2):230–245, 2008.

[GHM+07] Thanaa M. Ghanem, Moustafa A. Hammad, Mohamed F. Mokbel, Walid G. Aref, and Ahmed K. Elmagarmid. Incremental Evaluation of Sliding-Window Queries over Data Streams. *IEEE TKDE*, 19(1):57–72, 2007.

[GÖ03a] Lukasz Golab and M. Tamer Öszu. Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In *VLDB*, pages 500–511, 2003.

[GÖ03b] Lukasz Golab and M. Tamer Özsu. Issues in Data Stream Management. *SIGMOD Record*, 32(2):5–14, 2003.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In *SIGMOD*, pages 287–298, 1999.

[Krä07] Jürgen Krämer. *Continuous Queries over Data Streams – Semantics and Implementation*. PhD thesis, Philipps-Universität Marburg, 2007.

[KS04] Jürgen Krämer and Bernhard Seeger. PIPES - A Public Infrastructure for Processing and Exploring Streams. In *SIGMOD*, pages 925–926, 2004.

[KS09] Jürgen Krämer and Bernhard Seeger. Semantics and Implementation of Continuous Sliding Window Queries over Data Streams. *ACM TODS*, pages 1–61, 2009. (to appear).

[KYC+06] Jürgen Krämer, Yin Yang, Michael Cammert, Bernhard Seeger, and Dimitris Papadias. Dynamic Plan Migration for Snapshot-Equivalent Continuous Queries in Data Stream Systems. In *EDBT Workshops*, pages 497–516, 2006.

[SCZ05] Michael Stonebraker, Ugur Cetintemel, and Stan Zdonik. The 8 Requirements of Real-time Stream Processing. *SIGMOD Record*, 34(4):42–47, 2005.

[SJS01] Giedrius Slivinskas, Christian S. Jensen, and Richard T. Snodgrass. A Foundation for Conventional and Temporal Query Optimization Addressing Duplicates and Ordering. *IEEE TKDE*, 13(1):21–49, 2001.

[TCZ+03] Nesime Tatbul, Ugur Cetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load Shedding in a Data Stream Manager. In *VLDB*, pages 309–320, 2003.

[YKPS07] Yin Yang, Jürgen Krämer, Dimitris Papadias, and Bernhard Seeger. HybMig: A Hybrid Approach to Dynamic Plan Migration for Continuous Queries. *IEEE TKDE*, 19(3):398–411, 2007.

[ZRH04] Yali Zhu, Elke A. Rundensteiner, and George T. Heineman. Dynamic Plan Migration for Continuous Queries Over Data Streams. In *SIGMOD*, pages 431–442, 2004.

# Industrieprogramm

# Embedded Analytics in Front Office Applications

Erik Nijkamp, Martin Oberhofer

IBM Research & Development GmbH
Schönaicher Straße 220
71032 Böblingen
{nijkampe, martino}@de.ibm.com

**Abstract:** Today, decision making by users of front office applications happens without analytical information supporting this process. We propose as solution Embedded Analytics (EA) making analytical insight in context of the front office application available to users in an on demand fashion. "In context" means that only analytics relevant for decisions on the current User Interface (UI) screen is made available. "On demand" means that the user has the information accessible in "mouse-over" events which means the user decides when to consume which portion of the relevant analytical information. The underlying implementation uses the Unstructured Information Management Architecture (UIMA) to determine the context and then dynamically binds real-time lookup services for the delivery of the analytic insight to the application UI using Web 2.0 techniques in the UI. We explain the technologies and algorithms used for Embedded Analytics in context of business processes running in a SAP Netweaver CRM system.

## 1 Business Problem

Users of front office applications such as call center or customer support applications make millions and millions of decisions each day without analytical support. For example, if a support employee gets a new support ticket and needs to decide how much time should be used for problem resolution and which measures should be taken, this is done without analytical insight such as customer revenue, customer net profitability or customer potential. There are two barriers preventing the use of analytical information derived in BI systems: The first barrier is related to the constraints of the user group. Front office application users usually lack the BI analyst skills to work with complex BI tool sets to create reports. Running the daily business process operations has priority – there is no time to build and run analytic for each and every decision which needs to be made on a case by case basis s. The second barrier is the fact that the use of BI is not yet tailored to bridge the gap between the operational and analytical systems. This would require the real-time delivery of analytical information as a seamlessly consumable service within a business process. As a result, companies cannot optimize their front office departments because analytical insight derived in Business Intelligence (BI) systems or other systems is not available to users in these departments. A solution to this business problem must therefore deliver analytical information in context of the business process step the user performs without any additional skill set requirements at the time the information is needed in real-time.



**Trademarks:** IBM logos are trademarks or registered trademarks of the International Business Machines Corporation in the United States, other countries, or both. BAPI, SAP Netweaver, SAP CRM, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

## 2 Business Intelligence and Embedded Analytics

Business Intelligence[1] encompasses methods and technologies which support management in decision making by providing analytical insight into operational data. Looking at BI from a process perspective [GG00] then BI can be considered as a process creating analytical insight into the own company status and perspectives in the market place from heterogeneous internal and external source systems. Therefore, BI can be considered the discipline consisting of methods and processes to systematically analyze internal and external data in a electronic fashion with the goal to support management in strategic and tactical decision making supporting the overall company goals.

### 2.1 Business Intelligence – traditional view

Since the 1960's analytical solutions supporting decision making have been known by different names such as Decision Support Systems (DSS), Executive Information Systems (EIS), Data Warehouses (DW) and today they are known as BI solutions [HW05]. BI systems need to consume operational data from Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), Supply Chain Management (SCM) and many other application systems with an overall maximum size in three digit Tera-Byte volumes at large enterprises. The BI process is usually structured as shown in Figure 1. It extracts operational data from various source systems, cleanses and transforms it and then loads it into a Data Warehouse. This first step is known as Extract-Transform-Load (ETL) process step. Once the DW is populated data analysis is done using approaches such as Online Analytical Processing (OLAP) or Data Mining.

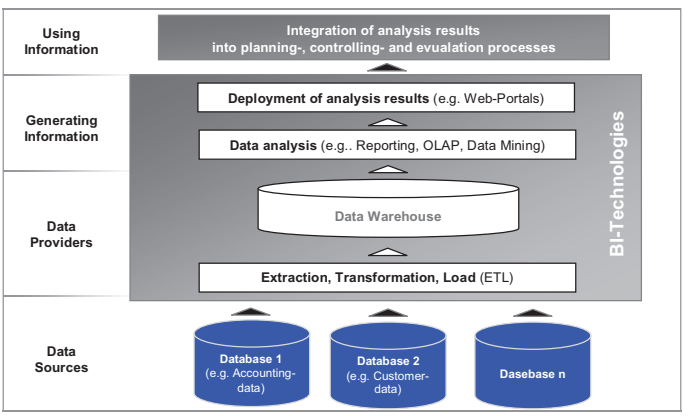



Figure 1: BI related aspects and technologies [Gen06]

The analytical results are summarized in BI reports which are sometimes deployed into intranet Portal applications used by managers to access the BI results supporting their decision making. Through the BI process operational data has been transformed into

[1] The Term „Business Intelligence" was created by the Analyst Howard Dresner in 1989.

analytical data. The key difference between operational and analytical data is that the former is used to perform a business process and the latter is used to support decision making to control the company.

Today, BI is a default component in an IT department in almost all companies and analysts project an increase in budget for BI solutions during the next year[2]. Even though BI is used in many companies, a study performed by SAP Labs concludes that BI so far reached only 30% of its potential. The study concludes in the best case only 3 out of 10 actors with a need for analytical information have access to it [Sch07]. The unresolved issue for many companies is the transfer of analytical results into their daily operations – in essence to the users of their front office applications. Simply put: The quality of the analytical information today is very high – but there is a lack of mechanisms to distribute this analytical information to the vast majority of users in need of it. This created the need for operational Business Intelligence as another sub-discipline in addition to strategic and tactical BI. The key characteristics of the three sub-disciplines are shown in Table 1.

Table 1: Subdisciplines of BI [Imh06]

| | Strategic BI | Tactical BI | Operational BI |
|---|---|---|---|
| Business focus | Achieve long-term organizational goals | Conduct short-term analysis to achieve strategic goals | Manage daily operations, integrate BI with operational systems |
| Primary users | Executives, analysts | Executives, analysts, line-of-business (LOB) managers | Line-of-business managers, operational users and systems |
| Timeframe | Months to years | Day(s) to weeks to months | Intra-day |
| Data | Historical metrics | Historical metrics | Right-time metrics |

While companies successfully use strategic and tactical BI already, operational BI is not yet there. Even worse, terms like “real-time” or “right-time” BI for operational BI solutions are imprecise from a BI process perspective as previously outlined. This means it is not clear whether or not just the ETL step is in real-time or the analytical processing within a DW or the entire process. In essence, it is not yet clear which process steps are performed in real-time which we further investigate in the next section.

### 2.2 Introduction of Embedded Analytics

Latency must consider regarding the overall BI process data latency, analytical latency and decision latency as shown in Figure 2. Several latencies accumulate between an original business event and the point where related decisions must be taken using information form the event.

[2] The Gartner Group projects an increase from 2.5 billion dollars in 2006 to 3 billion dollars for BI solutions in 2009. [Crm06]

The goal of operational BI is to increase agility and reduce the latency in the decision making process based on an as accurately as possible available data in order to improve agility supporting change in business requirements.

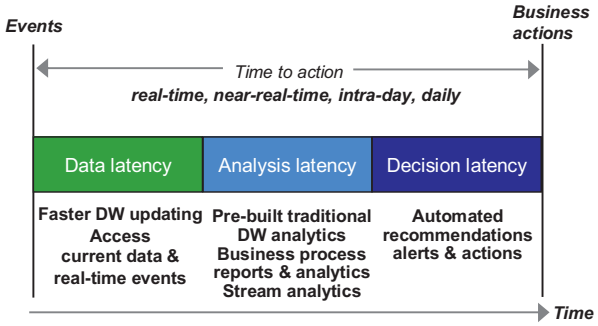



Figure 2: Latency in the BI process [WD08]

Responsible for applying the analytical insight in daily decision making are the users of the front office applications. For up to date analytical information naturally the desire arises to reduce the latency in analytical processing as much as possible with the ultimate goal to perform as close as possible to real-time. We therefore conclude that operational Business Intelligence is a trend towards a set of services, methods [WD08] and applications which monitor, analyze and optimize business processes on a daily basis. With this understanding of operational BI we can now define Embedded Analytics: Embedded Analytics (EA) is a processing environment where in a operational business process for optimization purposes real-time or near real-time analytical services support intra-day decision making by users of front office applications on a daily basis. At the core of EA is therefore the harmonization of the analytical with the operational application environment through an establishment of a direct communication channel between these two environments. This allows delivery of analytical information in context at the right time to the users of the operational applications. Separating Embedded Analytics into the term "Embedded" and "Analytics" as suggested by Eckerson [Eck06] we get a sharp focus on the essential meaning:

- Operational Process: A business process in execution
- Analytics: A process to convert data into analytical insight
- Embedding: The invocation of analytical insight from an operational process

In the following we therefore Embedded Analytics as "process-centric user analytics" with real-time latency.

### 2.3 The value of Embedded Analytics

The goals and therefore the potential of EA are the seamless distribution into the context of operational applications to allow "BI for the masses". Hence EA is simple to use in the context of the operational applications. A context switch from an operational application UI to a BI application UI is avoided because the analytical insight is delivered within the UI of the operational front office applications. This delivery reflects

the role and permissions of the user in the current context of the business process. Associating the BI results with operational processes enables the front office application users to make better decisions on a daily business due to improved, available business insight. This benefit materializes from two opposite directions: On the one end, potential problems can be detected before they become real problems and mitigations steps are possible preventing the real problem before it happens. On the other end, potential, temporary business opportunities can be identified on which a user can act in a daily step of a business process.

Implementing EA also requires a modularization of BI into BI analytical services. Once this step is done, these BI services can be more easily consumed in existing or new applications with the result of re-use and improved flexibility in the IT infrastructure. With EA business processes can be optimized and the operational and analytical environments become connected in a closed loop.

## 3 A Formal Reference Architecture for Embedded Analytics

*„You will waste your investment in SOA unless you have enterprise information that SOA can exploit ...“* Andrew White

According to A. Whites assumption a service-oriented architecture should incorporate not only functionality of online transaction processing systems (abbr. OLTP) such as ERP related services, but also analytical services. Being able to exploit valuable enterprise information in orchestrated processes is crucial to fully unfold the potential of business intelligence frameworks.

The multitude of involved components and their dependencies represents a challenging problem: How to address a broad diversity of architectural and technical aspects in a holistic and integrated manner [BD07]? One approach to solve this issue is the stepwise decomposition of operational and analytical functionality into architectural layers. Within this stack a layer provides services which consume elements of the layer beneath and provide functionality to the layer above. This procedure reflects the central idea of a ***SOA***: gradually abstracted services allow the simple composition of applications and processes.

In the scope of this paper embedded analytics is characterized as a method which can be used to provide a channel delivering context relevant business information (respectively a subset of metrics) to operational environments in a customizable way. In reference to the procedure described previously the following figure 3 shows the ***embedded analytics method*** decomposed into several components.

Notice that the proposed embedded analytics solution takes two different types of users into consideration: operational members of staff being responsible for the day-to-day business and business analysts who are able to identify trends and to express economic activities as a formal model.

From an architectural point of view a set of interacting components is required to realize an embedded analytics solution.

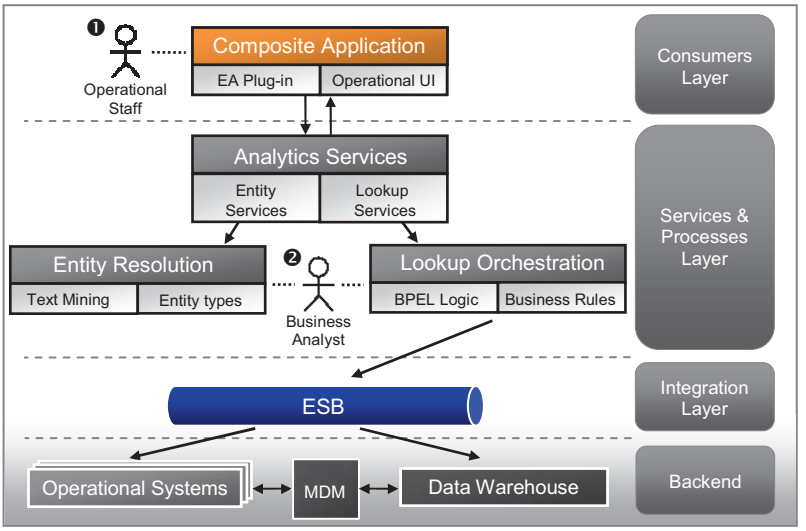



Figure 3: Proposed reference architecture for embedded analytics

Going from top to bottom, a ***composite application*** consists of the two crucial parts. The operational functionality required to manage the day-to-day business and an embedded analytics extension which provides analytical insight to support decisions instantly. In order to provide additional analytical information the embedded analytics ***plug-in*** transforms the operational user interface into a serialized representation and invokes an entity service to reveal the informative objects[3]. An entity is a type which has an identity and, if necessary, is used to transport additional attributes. Exemplary, a real-world entity such as a customer can be referred to in different ways (e.g. different name spellings). The goal of ***entity resolution*** (abbr. ER) is to resolve entities, by identifying associations between a set of varying representations and the informative object. Subsequently, ER algorithms such as text mining[4] are utilized to analyze the user interface and compare the given representation with a set of predefined entity types. Based on the retrieved list of entities the operational user interface is extended with anchors (e.g. adding mouse listeners on widgets). Whenever an event is triggered the embedded analytics plug-in calls the ***lookup service*** being associated with the given entity type. The lookup service inputs the information providing the identity of the requested entity and returns a refined representation which contains decision supporting facts. For example the lookup service could invoke a ***Business Process Execution Language (BPEL)*** process which retrieves additional customer data (e.g. historical revenue growth) delivered by a DW. In between a ***master data management*** system holds a single version of truth of master data to guarantee that the backend system can

[3] Informative in the sense of being useful for queries e.g. a product id which can be used to retrieve details.
[4] In this context we refer to the ability of text mining algorithms to determine which actual person or object a particular reference refers to, by looking at natural language text.

handle the lookup and associated entity. Viewed from a bottom up perspective operational[5] and analytical[6] systems provide services which are typically accessed in a centralized manner – this is the role, among others, of the ***enterprise service bus***.

Furthermore, the architecture incorporates a mechanism enabling business intelligence analysts to define the ***analytic logic*** in a customizable way. Business rules thus configured can be directly incorporated into reports used by the operational day-to-day business. For example figure 4 shows analytical logic defined as visual flow using BPEL as notation.

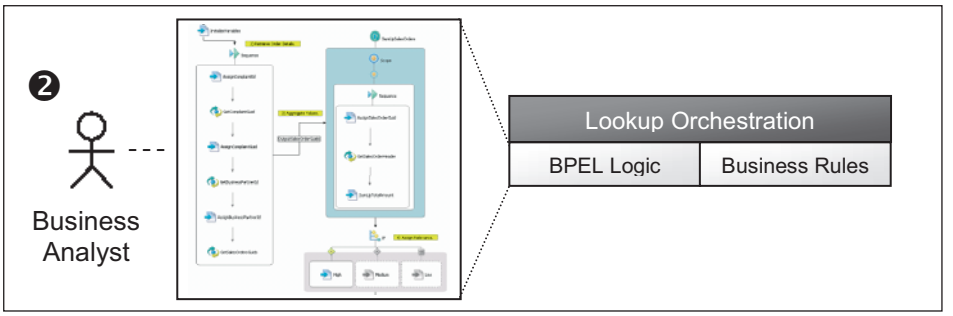



Figure 4: Analytical logic defined as a visual flow (BPEL)

Remarkably, this process of ***alignment*** between strategic/tactical guidelines and operational activities does not involve developers or administrators precisely because the logic itself can be defined by an analyst. In other words strategic and tactical goals are indirectly injected into the operational environment which affects the actual process execution and consequently optimizes the operational activities.

In addition it can be easily seen in figure 3 that the introduced layer required to define analytical logic acts as an ***abstraction layer***. Frequently, trends or substitutions are reshaping the IT landscape which affects the associated application interface. In order to react to these changes of the underlying infrastructure an analyst would ***redefine*** the analytical logic. However, the analytical output used by the embedded analytics services stays untouched. In fact this mechanism accepts mutable input parameter types while providing an immutable result – decoupling the operational or analytical systems from high level components. For instance replacing the Data Warehouse system does not affect the implementation of an arbitrary lookup service.

The advantages of this approach are as follows:

- the analytical insight of domain experts is directly injected into the daily business
- operational activities are aligned to tactical or strategic goals (e.g. business rules)
- decoupling of the underlying systems from the embedded analytics infrastructure
- a domain expert is able to modify to the analytics without involving a developer
- the visualization of analytical logic allows non-technical individuals to understand the factors which "drive the business"

[5] E.g. Enterprise-Resource-Planning or Customer-Relationship-Management Systems.
[6] Resp. Data Warehouse Systems.

# 4 Integrating Analytics into Complaint Management Processes

*„Once a good product was all you needed. Today, the totality of experience has equal impact.“* Duane Sharp

This chapter describes how to utilize an existing operational process or user interface as vehicle for predefined embedded analytics. The key idea is to deliver target-oriented analytical information (e.g. metrics) to the responsible employee in such a way that any crucial Customer Relationship Management (abbr. CRM) process can be improved by supporting a decision at hand. In order to increase employee effectiveness the following aspects are addressed:

- analytical results are computed automatically in real-time as needed
- the analytics are available in relevant portions or as whole to the employee “on demand” in an easy to use fashion, not flooding the employee
- the analytical insight is made available in the context of the user interface where the decision processing takes places

The idea is that an operational environment implicitly defines a context which can be used to establish a link between a set of business intelligence reports and the decision to be made. Besides providing concise reports as indirect decision support, the approach can be extended to offer an appropriate set of recommendable decisions (a subset of all possible decisions). Based on the analytical logic (e.g. logic which incorporates tactical business rules) the embedded analytics plug-in could manipulate the CRM user interface to only allow decisions with are conform to the tactical goals or policies.

Figure 5 shows an embedded analytics prototype which enriches the SAP CRM web form that handles customer complaints with highlighted annotations and analytical insight. In this case a regular HTML website represents the operational user interface.

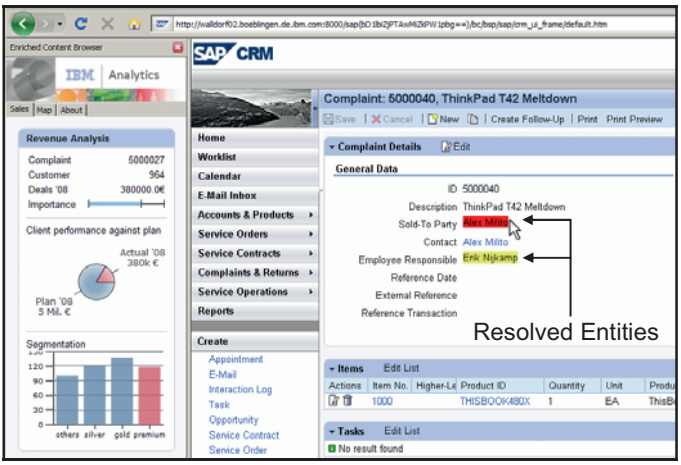



Figure 5: SAP CRM user interface extended with analytical information

The embedded analytics functionality is provided by a browser plug-in which analyzes the DOM tree and renders the analytical information. For example, in figure 3 the user pointed at the highlighted customer name and obtained additional revenue information.

Figure 6 describes the underlying method as a sequence of operations transforming a given input to output which is used by the subsequent operation.

1. The customer complaint form is analyzed for entities that are relevant to the users task (e.g. customer numbers, phone numbers)
   a. The browser plug-in draws the plain text content from the HTML document skipping all document mark-up information. However, the mark-up information can not simply be discarded as the relation between a text passage and its mark-up must be obtained. The document's plain text content is send to the entity services (UIMA annotators).
   b. The annotators process the plain text document, identify entities according to regex patterns and return a set of objects including the position within the HTML document.
2. The enriched document content is displayed in the web browser
   a. The original document and the text analysis results coming from the UIMA annotators are interwoven. The entities detected by the text analysis annotators are highlighted in different colors and extended with additional client side code that triggers one or multiple actions when the user moves the mouse over a detected entity.

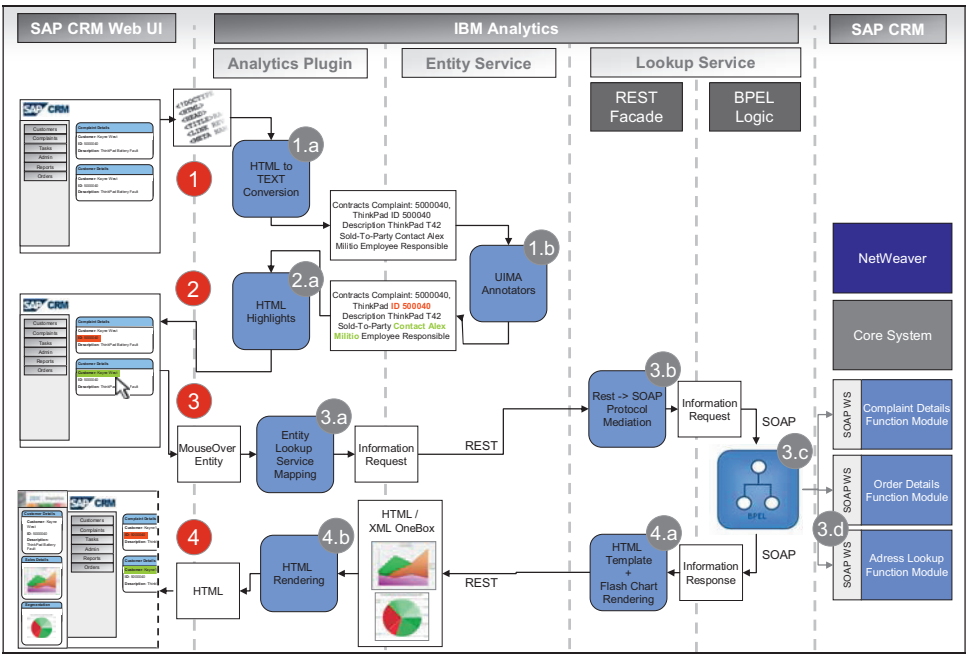



Figure 6: Analytical logic defined as visual flow

3. The browser renders the new document and the end-user can easily extract additional information from various back-end systems by simply moving the mouse over an entity.
   a. The user moves the mouse over a customer's name, which triggers a call-back function.
   b. In case the original function call was specified in the REST format, a transformation to a SOAP call is possible. This mediation can be useful as REST is very popular among web developers whereas SOAP is very common in SOA environments.
   c. The SOAP request is fed into a Workflow system where it kicks off a workflow process that could e.g. be specified in BPEL.
   d. The workflow process itself can contain some complex logic specifying which backend systems to call under what conditions and for what results.

4. The information gathered from one or multiple backend systems is displayed to the user and help's him/her to decide how to handle this customer complaint best.
    a. The output of the previously called business process is transformed into a mark-up document so that it can be displayed by a web browser (e.g. charts)
    b. The browser displays the information generated by the lookup service, the user can benefit from the additional information.

Notice that the mentioned Apache UIMA framework provides functionality which allows us to identify entities in unstructured information (e.g. HTML code). This prototype extracts the analytical information directly from the transactional SAP CRM instead of accessing a Data Warehouse for DW. Nevertheless this demonstration shows how embedded analytics can be used to optimize the complaint process by providing decision supporting information.

## 5 Summary and Future Work

In the context of the complaint management process in a SAP Netweaver CRM system EA delivered on its value promise which means the user get the analytical information in context at the right time on demand. As a net result, the user can now decide for which customer it is worth spending more time and efforts in resolving the complaint based on the customer revenue metrics and other analytical information. This improves customer satisfaction in the profitable and important customer segments and supports therefore key objectives of effective customer relationship management. We also conclude that Embedded Analytics will not replace the need for strategic and tactical BI solutions because it doesn't address the needs of the users with these requirements. It is in our perspective a complementary, new BI discipline enabling BI for the masses which do not have the BI analyst skill set.

If the DW provides access to accurate customer segmentation based on revenue metrics to a front office application, the DW needs to be able to provide this analytical insight in real-time. Building a DW system with seamlessly consumable services capable of returning this in a real-time fashion requires further research.

## References


[GG00] Grothe, M. / Gentsch, P.: Business Intelligence - Aus Informationen Wettbewerbsvorteile gewinnen, Addison-Wesley, P. 11-13, Munich 2000.

[HW05] Humm, B. / Wietek, F.: Architektur von Data Warehouses und Business Intelligence Systemen, Springer, Munich 2005.

[Gen06] Gentsch, P.: Business Intelligence for better decisions, http://www.intelligence-group.com/downloads/BI-for-better-decisions.pdf, 2006.

[Crm06] CRM Today: Business Intelligence Software Market to Reach $3 Billion in 2009,, http://www.crm2day.com/content/t6_librarynews_1.php?news_id=117297, 2006.

[Sch07] Schubert, L.: Embedded Analytics Are Not a Myth, in: SAP Insider, P. 12-16, 9/2004.

[Imh06] Imhoff, C.: Operational business intelligence, http://www.teradata.com/tdmo/v06n03/Viewpoints/EnterpriseView/OBI.aspx, 9/2006.

[WD08] White, C. / Davis, J.: Using Embedded Business Intelligence and Analytics for Near-Real-Time Decisions and Actions, http://www.beyeresearch.com/study/7561. 6/2008.

[Eck06] Eckerson, W. (2006): Embedded Analytics, http://download.101com.com/pub/tdwi/Files/TDWI%20Monograph%20Embedded%20Analytics.pdf, 5/2006.

[BD07] Berson, A.; Dubov, L.: Master Data Management and Customer Data Integration for a Global Enterprise, The McGraq-Hill Companies, 2007.

# An Architecture for Integrated Operational Business Intelligence

Dr. Ulrich Christ

SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf
ulrich.christ@sap.com

**Abstract:** In recent years, Operational Business Intelligence has emerged as an important trend in the Business Intelligence (BI) market. While sharing some obvious features with "traditional" data warehouse based BI, some of the requirements for the operational flavor are strikingly different justifying the question how, or even whether, Operational BI fits into a DW environment. We describe an architecture that, even though optimized for Operational BI, integrates smoothly into the broader data warehousing picture.

## 1 Introduction

While the term Business Intelligence is typically associated with systems and techniques that support the information need for strategic decisions – often based on historical data – Operational BI refers to the application of BI methods (and technology) to the vast number of low-level decisions to be taken in the daily operations of a business. Ideally, the respective reports or queries are tightly integrated into the business processes of a company to provide users with actionable just-in-time information when executing their task[1].

Typical examples would include CRM or call center applications, but also logistics or real-time analysis of technical processes can be named here. Due to its proximity to transactional processes and applications, Operational BI differs from Strategic BI in several aspects, in particular:

- Operational Integration as opposed to Information Integration – Operational BI reports typically have a narrow focus, supplementing a certain operational application; as a consequence, the need to integrate data from various applications (or even systems) is far less pronounced than in the realm of Strategic BI

[1] [Eck] provides a good overview of the topic

- Target group – due to its nature, a much higher number of users on all levels of an enterprise has access to Operational BI

- Latency – to support business decisions effectively, data has to be available for reporting much faster, ranging from within a few hours to sub second latency.

## 2 Common Approaches to Operational BI

It is a very natural and widely-spread approach to apply standard BI and data warehousing technology to analyze operational data. Particularly the question of whether to incorporate data for operational analyses into an existing data warehouse or not is probably the most fundamental one in discussions about Operational BI.

### 2.1 Direct Access to Application Persistences

Maybe the oldest and most straight-forward idea when it comes to Operational BI is to directly access data in the application systems. Today several BI vendors promote this approach as "data federation". While avoiding the need to replicate data, the main drawback are the resulting performance implications due to the mixed workload imposed on the operational system. Moreover, data will typically be stored in a write-optimized and thus highly normalized fashion which increases the complexity of BI queries even further.

### 2.1 The Data Warehouse

Data warehouses are a well established part of most IT infrastructures. In heterogeneous landscapes they serve as a data hub, integrating data from various application systems, taking care of data quality and providing a dedicated point of access to this data for analysis purposes. However, their role as a data hub also has drawbacks with respect to the requirements of Operational BI. First, data warehouses are generally loosely coupled with operational systems. They extract data from there in typically daily (or longer) intervals leading to significant latency issues in operational scenarios. To overcome these problems, (Near-) Real-time DW technologies like SAP NetWeaver BI Realtime Data Acquisition have been developed which allows a reduction of data latency to the range of one to five minutes. Yet, in many situations the data warehouse approach, which introduces additional data layers and processes that need to be maintained, might be considered too heavyweight.

From our point of view, the need to integrate data from various sources should be the decisive argument for using a data warehouse. In many scenarios, e.g. when the relevant data comes from a single application, a leaner approach would certainly be preferable. However, we would like to emphasize that in order to provide scalability and add flexibility, such a lean architecture should integrate into the large scale system landscape and be able to propagate data into a data warehouse.

### 2.3 The Operational Data Store

Within W.H. Inmon's concept of a Corporate Information Factory (CIF), Operational Data Stores (ODS) are architectural entities specifically designed for the purposes of Operational BI. They play a dual role, supporting both decision support and operational transaction processing[2]. Like data warehouses they *integrate* data from various systems. In fact, as figure 2.1 shows, they can be used as intermediate layers between operational systems and a data warehouse. However, they differ from data warehouses in three important aspects:

- their contents are *volatile*, meaning that they change at much higher frequencies than contents of a data warehouse
- they keep *detailed*, highly granular data and are
- *current valued* in the sense that they contain no (or little) historical data.

In particular the concentration on current data keeps the size of Operational Data Stores manageable and ensures acceptable query runtimes on fresh operational data.

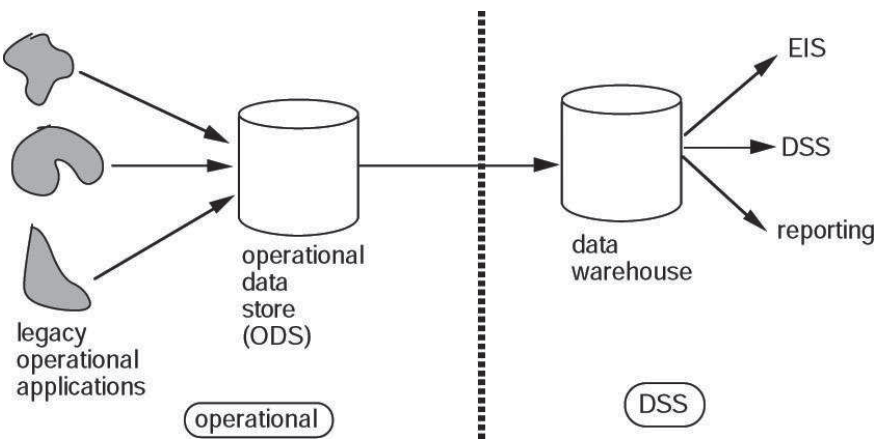



Figure 2.1: Inmon's Operational Data Store[2]

[2] for a detailed discussion of the subject, see [IIB]

# 3 Operational BI as Part of the Enterprise Data Warehouse Architecture

In this chapter we describe an architecture that allows to combine low data latency with excellent query response times. To this end we place an entity similar to the Operational Data Store inside the operational system and tie it closely to business transactions. In addition we make use of in-memory technology to achieve excellent query response times.

## 3.1 Motivation and Requirements

There is certainly no one-size-fits-all approach that supports all possible use cases in an optimal way. For relatively simple scenarios with small amounts of data, direct access to the operational sources is most likely appropriate. On the other end of the spectrum, complex scenarios will be best supported by data warehouses with specialized (near-) realtime technology like SAP NetWeaver BI Realtime Data Acquisition. Especially when integration of data from several operational systems is required – and consequently data quality issues tend to arise –, a central data hub fits very well in the overall architecture.

However, in between these extremes lies a number of scenarios that are of modest complexity and do not require integration, yet the underlying data volume makes direct access to operational data impractical. In these cases, replication into a data warehouse for mere performance reasons is certainly undesirable and a more lightweight and flexible solution would be preferable.

Overtime, as a company's IT landscape and processes change, though, it may turn out to be necessary to integrate the accumulated data into a data warehouse. We will take this into account and ensure that, like Inmon's Operational Data Store, our architecture can serve as an access point for extraction into a central data hub.

To summarize the requirements, we focus on a technology that

- enables excellent query response times with minimal data latency
- does not require a full-fledged data warehouse
- can act as an upstream persistence for a data warehouse

### 3.2 Large Scale Architecture

The architecture we describe in this paper resembles the notion of an Operational Data Store. It shares two of the technical properties in being able to keep volatile and detailed data with expected data latencies in the (sub-) second range. However, it is intended to be able to keep historical data over longer time intervals than Operational Data Stores. It is also closely coupled with operational applications and therefore does not serve as an integration hub.

Figure 3.1 below shows the positioning of our *Integrated ODS* within a typical landscape of operational and decision support systems. From a semantic perspective we would like to point out that the object should not be seen as tied to a specific application, which means that e.g. customer master data could be shared among several applications. We will elaborate on this in Chapter 4 "Reporting View" below.

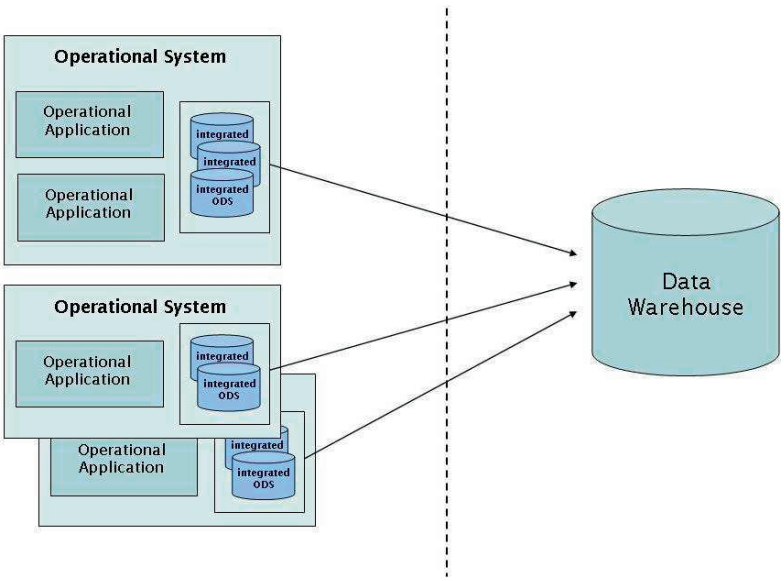



Figure 3.1: Positioning of the Integrated ODS

### 3.3 ETL Services in SAP Systems

BI experts commonly agree that it is not in general possible to *derive* a meaningful view suitable for BI reporting from a transactional schema automatically; in particular, join operations will generally not suffice to transform operational data into a format that suits reporting needs. Thus one can very well argue that provisioning of (transaction) data in a dedicated reporting format is part of the application logic.

Reflecting this within an SAP landscape, the NetWeaver BI Service API provides services and interfaces that enable SAP applications to pre-process data for BI purposes. *DataSources* provide a metadata description of a view on application data that is specifically designed for BI purposes. Generally they include an extraction program and additional information about the extraction process they support. One of these extraction processes works via the so called *Delta Queue*. When committing transactional changes to the database, an application can write corresponding records for reporting purposes to this container, from which they can be extracted asynchronously and in historical order to the SAP data warehouse. See figure 3.2 for a sketch of this architecture.

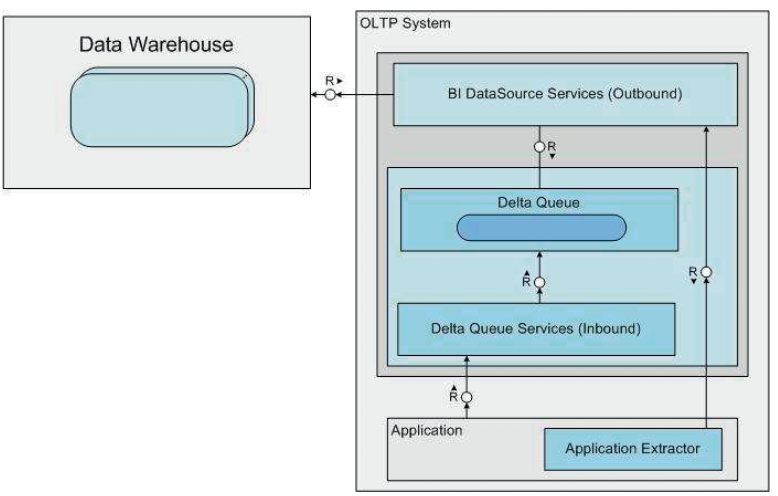



Figure 3.2: NetWeaver BI Delta Queue

### 3.4 Architecture Details

It is therefore a natural idea to enhance the *Delta Queue* to be capable of executing OLAP queries while keeping its role as an intermediate layer between operational persistence and data warehouse. To be able to hold and query large amounts of data and thus be able to operate independently of a data warehouse, SAP NetWeaver TREX provides excellent column based storage and aggregation technology, which is also used for the SAP BI Accelerator.

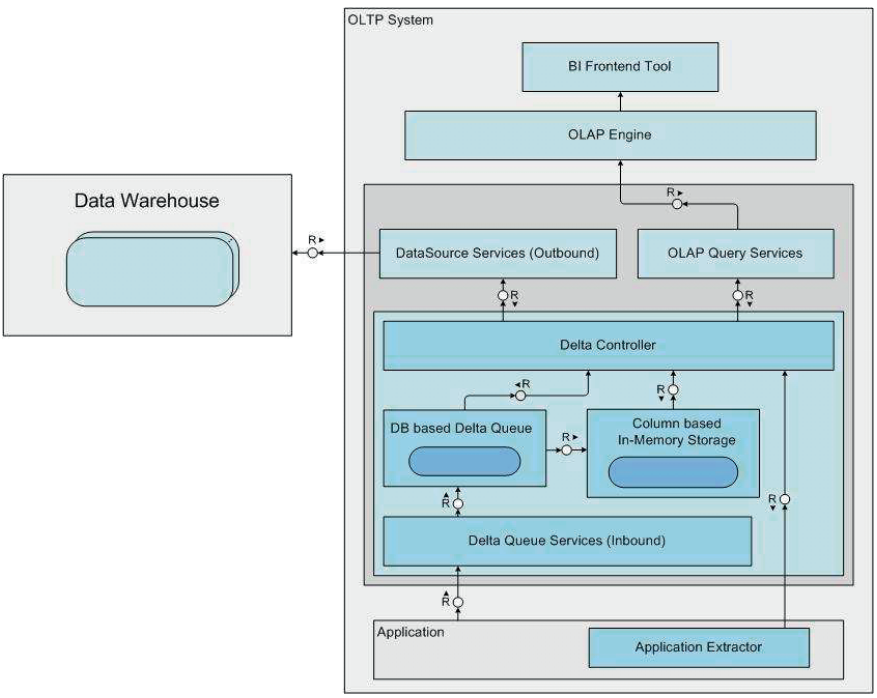



Figure 3.3: The Integrated ODS offers interfaces for OLAP tools as well as for replication into a data warehouse

Column based technologies are optimized for high speed read access and aggregation. However, they are less optimal for typical transactional updates consisting of few records and occurring at high frequencies. Besides the increased costs for continuous updates of the read-optimized structures, a certain communication overhead and a more complicated commit protocol have to be taken into account.

To minimize the effect on transaction runtime, we keep the original Delta Queue as a buffer receiving updates as part of the OLTP transaction. From there data is propagated to TREX's column based storage engine in intervals of about once per minute. An update frequency in that range allows keeping the size of the Delta Queue small and ensures that the communication overhead remains acceptable.

To accomplish both our goals of providing high speed query access and allowing consistent delta replication into the data warehouse, inside TREX data is organized in two layers. One, which will be used for typical Operational BI queries, is keeping the up-to-date data, i.e. the most recent record for every given semantical key. Upon execution of a BI query, the most recent updates from the Delta Queue are merged with the data in this layer to compute the query result. In this way, data latency is reduced to the (sub-) second range. Of course, some sophistication is required for this merge since we cannot simply transfer the cost saved during OLTP commit to the time of query execution. The second layer keeps track of historical changes of records which are needed to supply consistent delta information to connected data warehouses.

## 4 Reporting View

So far we have discussed processing of data for individual *DataSources*. To explain how to combine the resulting *ODSs* for reporting purposes, we consider a simple scenario involving Sales Order, Business Partner and Product data.

For each of these objects we have a corresponding *DataSource* and *integrated ODS*. To provide a meaningful reporting view combining transaction and master data, we need in addition a thin metadata layer which describes the relation between Sales Orders, Business Partners and Products – basically knowledge of foreign key relationships and specification of Sales Orders as the "transactional facts" suffices[3].

Figure 4.1: Combining master and transaction data to a Reporting View

If master data is harmonized over a suite of applications, the corresponding ODS's like Products or Business Partners can be shared with other reporting views, e.g. in a Delivery scenario.

[3] Recent developments in NW BI's Analytical Engine allow to derive the metadata for NW BI's Olap Processor at runtime from (in our specific case) the DataSource metadata.

## 5 Summary

The most common approaches to Operational BI, namely using data warehouse or data federation functionality, both are appropriate in a certain range of scenarios. However, they tend to have deficits when it comes to either performance or data latency and they are quite loosely coupled with business transactions. In particular, in scenarios with high data volume produced by a single application an integrated, lean and yet flexible approach is desirable.

We therefore argue in favour of an architecture that allows tight integration of BI functionality into business applications and consequently can resolve the performance vs. latency dilemma to a large extent. It should be considered a lightweight extension of data warehousing functionality in the operational system adding flexibility in scenarios where using a data hub seems inappropriate. However, we also emphasize that in order to achieve true flexibility – e.g. in case of changing requirements concerning data integration or lifecycle –, it is most important to ensure integration of the architecture with the data warehouse.

## Bibliography

[Eck] Eckerson, W.W.: Best Practices in Operational BI – Converging Analytical and Operational Processes. TDWI Best Practices Report, 2007

[IIB] Inmon, W.H.; Imhoff, C., Battas, G.: Building the Operational Data Store, Second Edition. John Wiley & Sons, 1999.

# Enhancing Business Intelligence with unstructured data

Alexander Lang, Maria Mera Ortiz, Stefan Abraham

Advanced Analytics Development
IBM Research and Development Germany
Schoenaicher Str 220
71032 Boeblingen
alexlang@de.ibm.com
mmera@de.ibm.com
stefana@de.ibm.com

**Abstract:** Quality early warning and proactive customer churn detection are two examples of applications that can benefit from insights gained from unstructured text data. The term "Unstructured Business Intelligence" describes methods and tools that enable data warehouse applications to use unstructured information. This article introduces the key components of Unstructured Business Intelligence and describes its realization within IBM InfoSphere Warehouse Enterprise Edition.

## 1 Unstructured Business Intelligence

### 1.1 Motivation

In a recent TDWI survey [Ru07], data management professionals were asked *Which types of data and source systems feed your data warehouse three years from now?* The respondents expected a huge increase in unstructured data as a source of warehouse information in the next three years. This included email, call center transcripts, documents from content management systems, and extranet content from forums or blogs.

This article will show how text analysis technology can transform this unstructured, textual data into meaningful pieces of information that can be used within Business Intelligence applications. Unstructured data can improve the quality of existing BI analytics, or, in some cases, is the key enabler for new types of insights. Typical business scenarios include:

- *Improve Quality Early Warning*: Internal problem reports, customer email or call center transcripts can yield valid information about emerging product problems. Today, companies try to capture these insights using a fixed set of categories within "problem taxonomies". Such taxonomies typically suffer from granularity problems: if they contain only high-level categories, they can't capture the actual reason for a problem. However, if they try to capture all possible problems, they become too unwieldy to use for front-line workers, who just stick to the categories they know (especially in a high-stress environment such as a call center). Thus, the actual reason for a defect is often buried within technician comments or call center logs. As a result, a company may detect that there is a problem with a certain product, but doesn't know which part causes the problem, and therefore can't take the right action: deciding on a product recall, or checking other products that use the offending part. On the contrary, Figure 1 shows a sample report showing correlated terms extracted from customer complaints for a certain car model, which provides direct insight in likely problem spots.

- *Reduce customer churn*: Companies in the telecommunication sector already have elaborate predictive analytic models for customer churn, based on structured data. However, once the customer's unhappiness with a certain service shows up in the structured data (for example, a decline in the number of long-distance calls made), it may already be too late. By analyzing each customer contact with the company, be it email or call center records, a company can earlier detect angry, unhappy customers, or customers that explicitly reference a competitor, and include that into their churn model. This allows for taking action at the first sign of customer discontent.

- *Reputation management*: Blogs, news articles and consumer portals increasingly affect customer's buying decisions, especially for consumer goods. Analyzing this extranet content helps to answer questions like *How do people talk about my company and my products – compared to my competition?* or *What companies are associated with "cool" technology?* – for example, hybrid technology in the automotive domain. This can serve as an "External Quality Early Warning" system, as not every unhappy customer bothers to write to the company, but whose forum entries may turn away quite a few prospective customers.

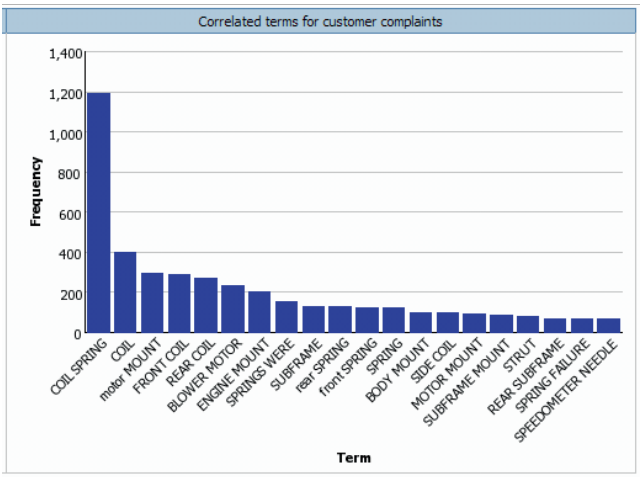



Figure 1: Analysis of automotive customer complaints

In all of the above scenarios, text is the main type of unstructured data. There may be the need to analyze semi-structured text, i.e. text containing markup elements like XML, or other data types of audio and video. However, we see that the bulk of content that is relevant for today's applications comes as free-form text from technician notes, customer comments through CRM applications or email, or relevant snippets from news services. Thus, the remainder of this article will focus on this type of text.

### 1.2 Definition: Unstructured Business Intelligence

Using text data for Business Intelligence comprises three steps:

1. Introduce dedicated text analysis components into the ETL flows that feed the data warehouse. These text analysis components extract information from the unstructured data, such as product names, product codes, indicators for problems or expressions of customer sentiment.

2. Enrich existing warehouse schemas and cube definitions with additional columns or dimensions for the extracted information. This requires that the text analysis technology delivers results that can be readily integrated into a relational schema.

3. Use the existing BI infrastructure on the enriched warehouse. Ideally, reporting, data mining and predictive analytics should not need to care that the additional information was originally extracted from an unstructured source.

We use the term *Unstructured Business Intelligence* (or *Unstructured BI*) to describe methods and tools that enable data warehouse applications to use unstructured information. This includes

- Text analysis components that extract information with sufficiently high quality to be used by automatic downstream processing.

- Approaches to create efficient relational schemas for text analysis results that enable both reporting and data mining.

- Analytics components that can combine structured and previously unstructured data to yield better results and more insight.

Unstructured Business Intelligence includes approaches that are often described as *Text Mining*. However, we think that text mining has too close a connotation with data mining, and leaves out important aspects such as data preparation, relational schema building and the use of text in "traditional" business intelligence reports.

### 1.3 Challenges

Using textual data in Business Intelligence presents some challenges to "typical" data warehouse infrastructure. Typical requirements include:

- *Dealing with misspellings and acronyms*: In a project with a Telecommunications customer, we started with an Excel sheet that contained the names of products, call tariffs and contract options. However, this sheet didn't include the variants used by call center employees. Adding these company-specific acronyms, for example *NOK* for *Nokia,* more than doubled the number of concepts that got extracted from call center records. Moreover, robust text analysis technology is required to deal with frequent misspellings (e.g. *Nioka*).

- *Advanced data cleansing:* Especially dealing with extranet content requires data cleansing and preprocessing capabilities that are typically beyond the capabilities of "traditional" cleansing products. For example, analyzing hotel reviews from sites such as tripadvisor [Tr08] requires to focus only on the actual customer reviews, not on the advertisements or the description provided by the hotel itself. Increasingly, this data cleansing becomes a "make or buy" decision, with companies such as kapow technologies [Ka08] offering screen scraping tools and content feed services that provide such targeted information.

- *Confidence values*: Text analysis sometimes lacks the context to decide on ambiguous terms. For example, the acronym *WPS* may refer to either *IBM WebSphere Portal Server* or *IBM WebSphere Process Server*. Some text analysis components are able to attach confidence values to the extracted information. However, there are currently few BI algorithms that make use of this information.

- *Agile text analysis configuration:* Text analysis technology typically has to be adapted to each domain, enterprise, and business scenario. Product names must be imported from Excel sheets or product databases and enriched with company-specific variants. Regular expression rules need to be created for product codes, taxonomies need to be developed for relevant topics. The key item for the success of an *Unstructured BI* project is the buy-in of domain experts and business users. Only they are able to provide the insights necessary for configuration. The IT department may take over some of the tasks (e.g., writing regular expressions), but only the business users know the relevant "business lingo". Hence, a potentially large user group must be able to influence the configuration in an agile manner. Otherwise, the resources will become outdated, yielding inferior results, and eventually, user rejection of the whole solution.

## 2 Text analysis: from text to structure

### 2.1 Information Extraction

Information Extraction is an area of natural language processing that is concerned with extracting concepts (so called entities) and relationships between these concepts from unstructured text. These results lend themselves for storing within a relational database, and are thus more relevant to *Unstructured BI* than technologies such as document clustering or classification. Relevant information extraction tasks are:

- *Named Entity Recognition (NER)*: extract person or place names, monetary expressions, problem indicators
- *Relationship Detection*: detect relationships, based on named entities, such as *part X causes problem Y*
- *Co-reference resolution*: identify expressions across a document that refer to the same entity, such as the hotel *Best Western* in *I liked my stay at the* ***Best Western***. ***It*** *has very bright rooms.* ***The hotel*** *also features…*

### 2.2 List-based and rule-based named entity recognition

Named Entity Recognition tasks are of varying complexity: *The Grosvenor Inn did not meet my expectations* contains both a hotel name and a negative customer sentiment. Whereas the hotel name can be detected by a match against a fixed list of hotels, the correct analysis of the negative sentiment requires some grammatical understanding to correctly pick up the negation.

One approach to Named Entity Recognition is *list-based extraction*. This is advisable for entities that can be exhaustively listed, such as employee names (taken, for example, from the company LDAP), or product names and their attributes. Some domains, such as Life Science already have "official" domain vocabulary, e.g. SnoMed CT [Sn07].

One advantage of list-based extraction is that it often comes from "trusted" sources. That means its creation and maintenance can be automated to a certain degree, e.g. by doing a "batch update" every time new products are introduced. Moreover, the extraction results are immediately plausible to the end user: the machine detects exactly the terms that are in the list. As mentioned earlier, the most labor-intensive part in using list-based extraction is the enhancement of the list with company-specific variants and acronyms.

Some types of entities, such as telephone numbers or monetary expressions can't be listed exhaustively. For these entities, *rule-based extraction* is the proper approach. These may either be regular expression rules over characters, or higher-level rules, such as regular expressions over previously detected entities. One advantage of rules is generalization: one rule may cover a large range of entities. Another advantage is that rules can take the document context into account. This is crucial for tasks like sentiment detection, where a negation word such as "not" flips the sentiment of a whole sentence.

The key challenge for rules is their complexity: users need help to create and maintain rules. For many people, writing regular expressions is as obscure as programming in Java – but these are often the people with the appropriate domain knowledge. Hence, configuration tooling is necessary that hides the intricacies of linguistics and rule languages from non-technical users. For complex tasks such as sentiment detection, even that is not enough. Creating the syntactical rules necessary for negation and other sentiment-affecting phenomena will overwhelm non-Linguists. In these cases, a combined approach is in order: the solution contains pre-packaged grammar rules, which are not meant to be changed by business users, but lets users modify the list of positive and negative terms, or the products and product features that should be related to an extracted sentiment.

## 2.3 Using BI analytics for Information extraction tasks

In "traditional" computational linguistics, relationship detection and co-reference resolution is often approached through elaborate rule sets. However, such rule sets are often very specific to a particular domain, or assume a formal writing style that often does not exist in the type of text relevant for *Unstructured BI*. In our experience, it is often beneficial to accept some loss in extraction precision for higher robustness against "real-world" text. This can be achieved by relying on co-occurrence heuristics and statistics, and lends itself easily to the machinery available within existing BI components. Relationship detection between parts and problems can be done by extracting parts and problems through NER, and applying data mining, namely, Association rule mining [AIS93], on the result to find parts and problems that occur within the same document in a statistically significant amount. Co-reference resolution can be done by associating all product features or customer sentiments within a certain document snippet to the concept (e.g., the hotel) found within that snippet. The right snippet length may vary from one sentence up to a paragraph, depending on the type of document, e.g. a consumer forum vs. a website with product reviews.

## 2.4 Decision factors for choosing information extraction technology

In general, choosing the right set of information extraction components depends on:

- *Business scenario:* How complex are the entities that need to be detected? For example, a purely quantitative analysis of product mentions in a call center doesn't require as elaborate processing as detecting dissatisfied customers within the same documents.

- *Configuration effort:* As stated in chapter 1, text analysis needs to be adapted to an enterprise, with business users playing a key role in the process. Do the expected business improvements merit the (ongoing) configuration effort for the desired technology?

- *Focused vs. explorative solution approach:* In a *focused solution* approach, the goal of the text analysis configuration is to extract the relevant concepts with high precision. This requires more effort for creation and testing the configuration, but allows the downstream BI analysis users to pose specific questions like: *Show the top 10 car parts that occur in repair reports for automobile make X.* In the *explorative solution* approach, text analysis detects terms in a very broad fashion, and doesn't attempt to match them against a certain concept, for example, by detecting all combinations of an adjective, followed by a noun. This requires more elaborate statistical analysis or data mining post processing to detect statistically significant concepts and relationships. Still, the results may show irrelevant terms, and the user can't do a concept-driven drill down like in focused solutions. However, an explorative solution is more suited for "open-ended" questions like *What are the causes for failure X?* where the root cause may be a car part, a certain tool used, or even a certain repair shop. Moreover, such an approach keeps the text analysis configuration effort to a minimum, and is able to pick up emerging concepts on the fly, without the need for a human to add them to a list of concepts.

# 3 Unstructured Information Management Architecture (UIMA)

## 3.1 Motivation

Scenarios such as company reputation management typically require a set of text analysis components to work together. For example, customer satisfaction analysis in the hospitality sector requires both list-based extraction to detect hotel names and hotel amenities, and rule-based extraction to detect customer sentiment. Chances are that the expression *Best Western Shanghai* will be tagged correctly as a hotel name, but the term *best* may trigger a positive sentiment rule as well, so that *Best Western* is tagged as a customer sentiment. To spot these inconsistencies, and apply subsequent cleansing steps such as *remove all customer sentiments that are totally included within a concept,* the text analysis components must represent their results in a common way. Moreover, a framework should manage the task to orchestrate different analysis components (which may come from different sources), rather than forcing solution builders to re-implement these capabilities.These requirements were the driving force behind the Unstructured Information Management Architecture (UIMA). UIMA is a software architecture that supports the rapid development, integration and deployment of unstructured information management technologies, including, but not limited to, text analysis [UI04]. UIMA started as an IBM initiative, but has now gone open source, and is enhanced through the Apache community [UI08]. The *UIMA Working Group* is a network of scientific organizations and companies that use UIMA in research projects and professional development.

### 3.2 Core concepts

*Analysis Engines* are the central building blocks within UIMA. An analysis engine contains one or more *annotators,* each implementing one specific text analysis functionality, or other analysis engines. This recursive packaging allows to build complex analysis engines out of simpler ones.

Analysis Engines work off a common data structure, the *Common Analysis Structure (CAS).* Each analysis engine stores its results within the CAS as typed feature structures. *Annotations* are a special subclass of feature structures. They contain a specific start and end position within a document and lend themselves easily for specifying information extraction results, e.g. as a *Hotel* annotation that covers positions 5 to 26 in the document *The Best Western Shanghai welcomes you.* This open data representation allows individual teams to develop an analysis step that is "best of breed" in what it's doing, and which builds on results delivered through prior annotators rather than recomputing them. Figure 2 shows several annotators working together for named entity recognition, grammatical parsing and relationship detection. Note that the *Relationship Annotator* can detect relationships without looking at the actual document text, but by analyzing pre-existing concepts and grammar.

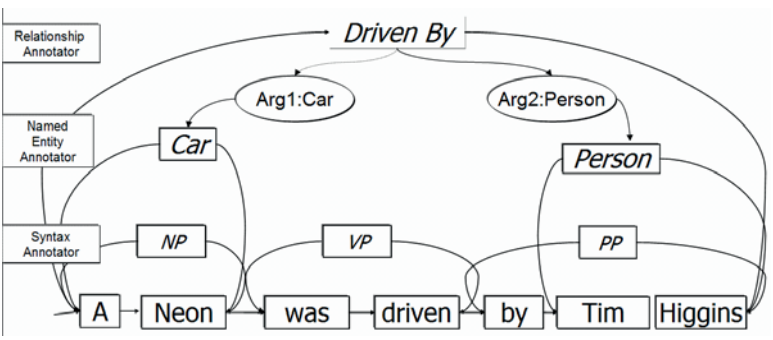


Figure 2: UIMA analysis process

### 3.3 Text analysis configuration workflow

UIMA aims to ease the integration of text analysis components, but does not provide text analysis configuration capabilities suitable for business users. Figure 3 describes this configuration as an iterative, ongoing process, with the following distinct phases:

**Explore Data:** The goal of this step is to understand "what's in the data", and to understand whether the textual data can actually help with the business problem. Tools that support the domain expert in this step are components that suggest potentially relevant terms from the documents, based on document statistics. Another way to explore the data are text search queries. The results may contain "signal terms" that can be used in regular expressions to improve their quality, or, by using fuzzy search, to find frequent typos and variants to include into dictionaries.

**Create configuration:** In this step, the domain expert decides on the extraction approach to use (e.g., rule-based vs. list-based) and creates the appropriate resources. This step requires easy-to-use editors, which are able to re-use the insights gained in the explore data step.

**Evaluate:** In this step, the domain expert examines the effectiveness of the rules and lists on sample, real-life data. This requires tools to run the analysis and to compare results, which is important to gauge the impact of configuration changes.

**Deploy:** In this step, the created rules and resources are deployed to the target system. Note that this is the only step that does not require any business users, but should be done through the IT department.

**Monitor:** In this step, feedback obtained in the production system over time must be applied to the resources to ensure that the configuration remains up to date. This is especially relevant in the focused solution approach. Monitoring the extraction recall through BI reports (e.g., *How many products have been detected on average per call center transcript*) can help detect concept drift: emerging concepts that are not reflected in lists or rules, and thus decrease the overall recall.

The *domain expert* in the process must be familiar with the business goals of the unstructured BI initiative, and with the documents that are to be used. He must work with front-line employees creating these documents to capture company-specific jargon.

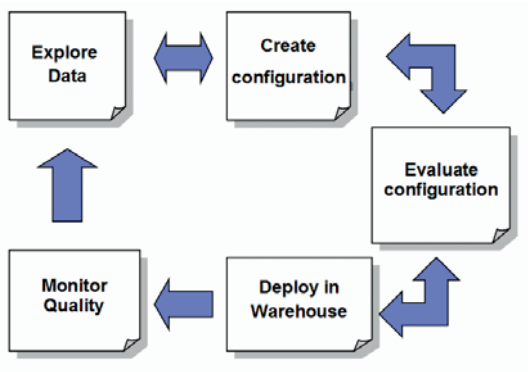



Figure 3: Text analysis configuration workflow

# 4 Unstructured Business Intelligence with InfoSphere Warehouse

## 4.1 InfoSphere Warehouse Enterprise Edition

InfoSphere Warehouse Enterprise Edition [In07] is a complete solution to design and build an enterprise data warehouse. In addition to “traditional” ETL and OLAP cube modeling, it provides integrated data mining and visualization. The data mining capabilities cover both discovery analytics through association analysis, sequential patterns and clustering and predictive analytics through various classification and regression algorithms.

Building up a suitable warehouse through ETL steps or creating and visualizing a data mining model for business insight is done graphically by defining *flows* within the *Design Studio,* an Eclipse-based design environment. These flows contain operators that either represent SQL commands or higher-level tasks such as text analysis. They are then deployed to a flow execution environment within IBM WebSphere Application Server. Depending on the operator, the actual tasks are pushed down to the underlying database, or executed as Java components with the application server.

## 4.2 InfoSphere Warehouse enhancements for unstructured BI

InfoSphere Warehouse 9.5 introduced key capabilities for Unstructured BI. First of all, text analytics became a first class citizen within ETL flows by introducing three new flow operators:

- *Dictionary lookup* runs a UIMA analysis engine for list-based extraction. It performs linguistic preprocessing of the text for more than 20 languages to make sure that plural forms or inflections of concepts are also detected. This way, a list entry for "wiring harness" also detects "wiring harnesses" and "wire harness", and “Fertighaus” also detects “Fertighäuser” (German for *pre-fabricated home*). Note that a simple “stemming” approach to plural detection, which simply cuts off word endings, would have problems matching the resulting stem *Fertighäus-* back to *Fertighaus*.

- *Regular expression* runs a UIMA analysis engine for rule-based extraction. It is capable of detecting concepts based on regular expressions over characters. Expressiveness and syntax conform to the java.uti.regex.Pattern class of Java 5 [Ja08]

- *Text analyzer* runs a custom UIMA analysis engine. This analysis engine may have been custom-built for a certain solution, or can be an analysis engine provided by other IBM products or the Apache community.

Above operators process the text contained in a CHAR, VARCHAR or CLOB column of a database table, and produce one or more output columns.

To create dictionaries and regular expression rules, the Design Studio was enhanced with dedicated editors that hide the underlying linguistic processing, or the actual rule language from the domain expert. A taxonomy editor allows assigning the extracted concepts to higher-level categories. This is especially relevant when using text analysis results in an OLAP context, as it allows describing additional aggregation levels over the extracted concepts. For example, a dictionary with IT terms like Java and Windows enables a report like "What are the TOP 10 IT terms?". However, creating a taxonomy with the paths *IT-Skills -> Programming Languages -> Java* and *IT-Skills -> Operating systems -> Windows,* allows for reports like *What are the Top 10 IT-Skills?* or *What are the Top 10 programming languages?*. By separating the taxonomy from the dictionary, different applications can impose their "view" on the extracted concepts, without requiring any changes to the text analysis itself.

The configured analysis engines run as Java components within the application server. Several analysis engines can run in parallel on the same database table to enhance document throughput.

## 5 Example scenario: Quality Early Warning

The Office of Defects Investigation is an office within the National Highway Traffic Safety Administration (NHTSA) in the United States. It maintains a web site where consumers can file complaints about vehicles, equipment (e.g., motorcycle helmets), child restraints or tires [Od08a]. The contents of the complaint database are publicly available and contain 511,068 unique complaint records. Each record contains 46 structured fields, plus an unstructured complaint description, which can contain up to 2048 characters [Od08b]. As Figure 4 shows, the complaint description has some "typical" characteristics for unstructured database content: it is in all-uppercase format, and does not exceed one paragraph in length.

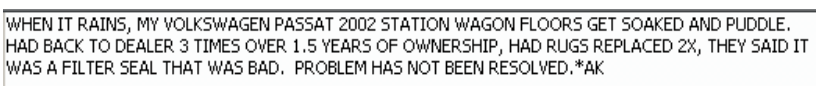

WHEN IT RAINS, MY VOLKSWAGEN PASSAT 2002 STATION WAGON FLOORS GET SOAKED AND PUDDLE. HAD BACK TO DEALER 3 TIMES OVER 1.5 YEARS OF OWNERSHIP, HAD RUGS REPLACED 2X, THEY SAID IT WAS A FILTER SEAL THAT WAS BAD. PROBLEM HAS NOT BEEN RESOLVED.*AK

Figure 4: Sample complaint description

While the structured fields contain descriptive information about the complaint, such as whether the vehicle was involved in a crash, or the number of people injured, they contain little information that point to the cause of the problem. The complaint may be categorized, but categories like *TIRES*, *AIR BAGS* and *OTHER* are not granular enough to describe the problem adequately.

This example scenario shows how InfoSphere Warehouse can extract relevant information from the complaint descriptions, which allows business analysts to investigate potential causes of problems for certain vehicles. In this example, the Quality Early Warning application is built as a set of reports within IBM Cognos 8 BI server. As the results of both text analysis and subsequent data mining are stored in relational tables, the application could be implemented in any BI tool, or custom application.

### 5.1 Text Analysis Configuration

In the *Explore Data* phase, we use InfoSphere Warehouse to find frequent terms within the complaint descriptions. These terms are extracted based on grammatical patterns such as *Noun-Noun* or *Adjective-Noun*. The domain expert selects the appropriate patterns, and InfoSphere Warehouse shows the results in a list or a cloud view, as in [Fig X]. In our example, we focus on grammatical patterns that match car parts, such as *Noun* (e.g., engine) or *Noun-Noun* (e.g. windshield wiper). As the underlined terms in figure 5 show, the results of frequent term analysis with above patterns would provide a good starting point for creating a dictionary with car parts. A domain expert could select relevant terms, add them to a dictionary, and enrich the entries with additional variants.

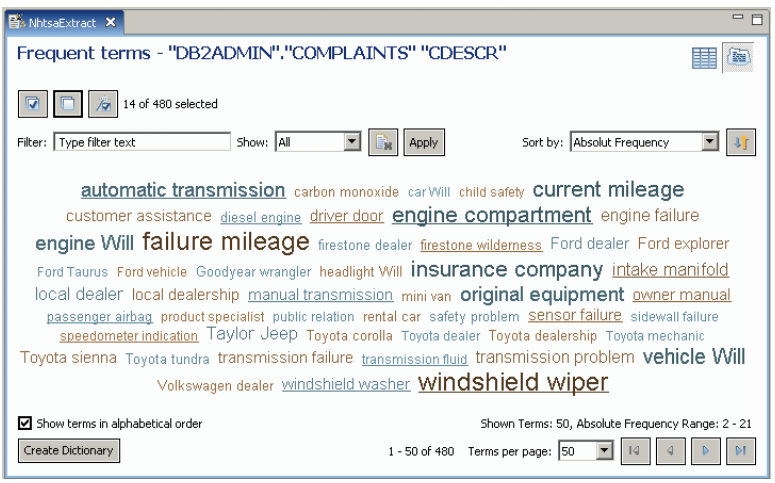



Figure 5: Frequent noun-noun combinations

However, in the example scenario, we chose to take an explorative solution approach. Rather than creating a dictionary specifically with car parts, we add all extracted noun-noun combinations and single noun combinations to a dictionary. Hence, the Create Configuration phase merely consists of a single mouse click on the *Create Dictionary* button shown in figure 5. This approach yields a sizeable list of entries, but the dictionary lookup operator compiles this list into an efficient binary representation, which provides a lookup time linear to the length of the lookup term, not linear to the length of the list, as in a naïve implementation. This allows to process several documents per second, even if the list grows to hundreds of thousands of entries.

In an explorative solution approach, the *evaluate* phase is used to check whether the dictionary contains entries from all relevant grammatical patterns. Depending on the document content, it may be beneficial to create additional dictionaries containing verbs that signify actions, or adjectives that describe some product features or circumstances. Figure 6 shows the result of dictionary lookup on a sample complaint. The tab *Analysis Results* contains an aggregated results view, which shows how many terms got extracted for each document. This allows the domain expert to gauge the recall of the current configuration, and to specifically check documents that contain zero or few results whether they contain terms that are not captured by the current configuration. On the other hand, documents with an exceptionally high number of terms may point to "stopwords" that are contained in the dictionary.

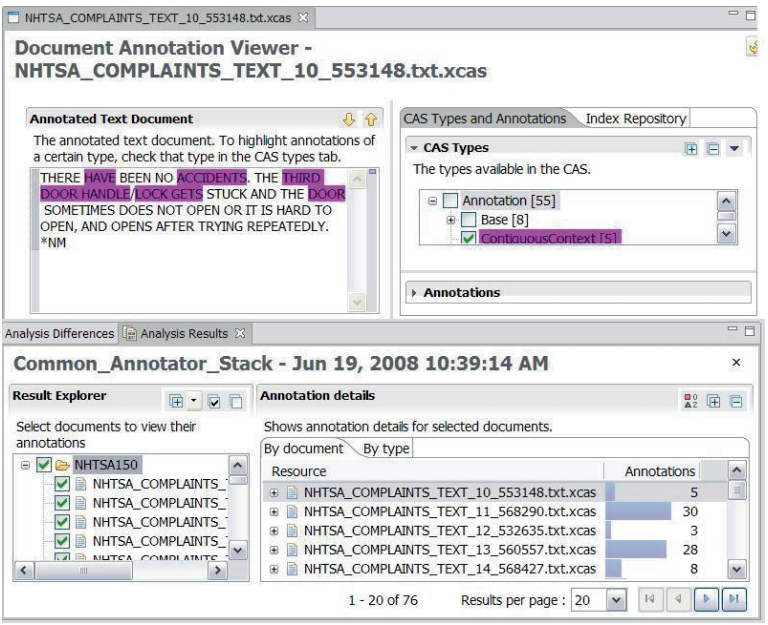



Figure 6: Text analysis result overview

In the *Deploy* phase, the created dictionary is used within a Dictionary Lookup operator of an ETL flow (figure 7). The text column CDESRC from the complaints database is connected to the input port of the dictionary lookup operator. This operator is configured to use the dictionary created in the evaluate step, as seen on the *properties* tab. The result is a table that contains one record per extracted term. The column CMPLID from the original table is routed through the operator as a join key with the original complaint.

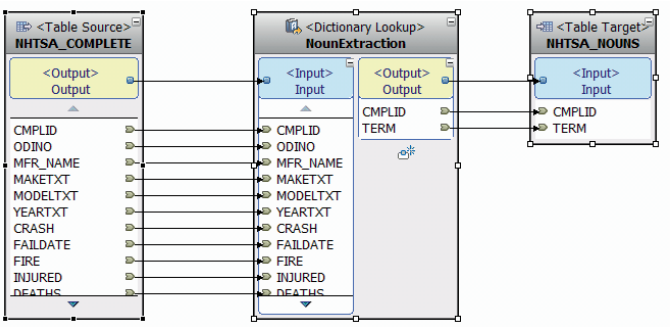



Figure 7: Text analysis ETL flow

The result table can now be used within a Business Intelligence application such as IBM Cognos 8 BI server. For the example, we chose car manufacturers, car models and the term list as dimensions for the data cube. This allows the domain expert to start with an overview how many complaints exist for each manufacturer. She can then drill-down to the car models of a certain manufacturer to see models that have a high number of complaints. Up to this path, the information is derived solely from the existing structured information. A further drilldown to a certain car model yields the insights gained from analyzing the car complaints. Figure 8 contains a report that shows the most frequently occurring terms for the car model Ford Taurus. However, showing only the most frequent terms for a particular model is not enough: entries such as CAR occur across all complaints, and don't yield insight. This underlines our statement made in chapter 2: the explorative solution approach needs analytical post processing of the results to detect terms that are correlated with the subject under discussion, in this case, the Ford Taurus.

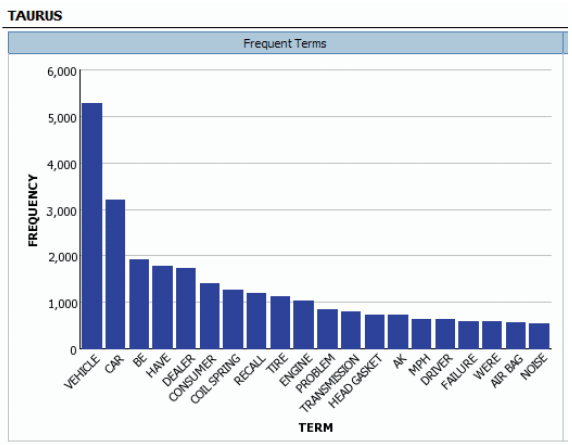



Figure 8: Most frequent terms in complaint descriptions for Ford Taurus

## 5.2 Combining text analysis and data mining

To find correlated instead of frequent terms in our example, we use Associations analysis that is part of InfoSphere Warehouse Data Mining. The goal of association analysis is to find items that are consistently associated with each other in a statistically significant way [AIS93]. Association analysis is typically used for *Market-Basket-Analysis*, which analyzes purchase transactions to discover combinations of goods that are often purchased together.

In our scenario, the goal of the association mining function is to find terms that are highly correlated with a certain car model. The results are rules of the form [rule body] ==> [rule head], e.g. [ventilator oil] ==> [TAURUS]. Figure 9 shows a flow that uses the text analysis results to build an association rule model. This model is built within all models of a certain car make (e.g. Ford), and attempts to find terms that are highly correlated with a certain model (e.g., Taurus). The latter is done by applying a rule filter that only accepts rules that contain Taurus in the rule head.

The extractor operator shown in figure 9 stores the detected rules back into a relational table. This way, the results can be visualized with "traditional" BI, and both administrators and users of BI applications don't need to be aware that the results were detected using data mining.

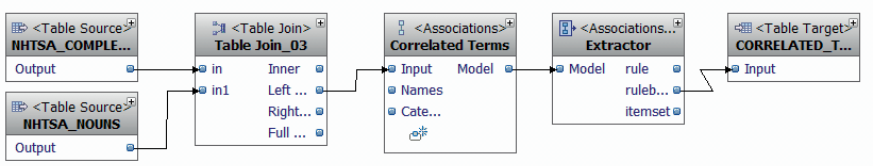



Figure 9: Using association analysis to detect correlated terms

Figure 1 contains a report that shows the most correlated terms for Ford Taurus, as detected by association analysis. The terms are much more specific – "stopwords" such as car and vehicle do not occur anymore. Many of the results focus around the *coil spring*, with both *front coil* [spring] and *rear coil* [spring] being affected. The second problem spot appears to be certain *mounts*, such as the *motor mount* or the *subframe mount*. The domain expert can now drill down into the actual documents that contain a certain term to understand whether there indeed is a problem.

The goal of this example scenario is not to take the business user completely out of the loop, but to keep down the information that (s)he has to consume to a manageable size: instead of reading through all 14000 complaints for the Ford Taurus, the business analyst can focus on the 80 that contain the term *body mount*. If the resulting document set is still too large (e.g. for *coil spring*, which still yields 1200 documents), the domain expert can focus on a sample set, or do a further drill-down on the term. This drill-down action triggers the creation of a new association model for the documents that contain, say, coil spring. The base set used to derive the association rules is now the set of all Taurus documents, not the set of all Ford documents. This way, the system can guide the user along relevant, correlated terms to specific "problem spots".

## 6 Summary

The data warehouse will experience a paradigm shift in the coming years: structured data remains important, but the ability to harness unstructured data will become key to support and enhance new business intelligence applications. This article has introduced the term *Unstructured Business Intelligence* to describe the methods and tools necessary to integrate unstructured data into the warehouse. In the context of a Quality Early Warning scenario, the article showed how IBM InfoSphere Warehouse enables building Unstructured BI applications. Unstructured BI includes text analysis components that perform information extraction, and tools and processes that allow the agile configuration of these components. But the work doesn't stop there: advanced data cleansing approaches and downstream analytic steps that can combine structured and previously unstructured data are equally relevant to deliver a complete Unstructured BI solution.

## 7 References


[AIS93] Agrawal R., Imielinski T, Swami A: Mining Associations between Sets of Items in Massive Databases. In Proceedings of the 1993 ACM-SIGMOD International Conference on Management of Data, 1993; pp. 207-216

[In07] Official product page:http://www.ibm.com/software/data/db2/dwe/enterprise.html

[Ja08] J2SE 5 documentation: http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html

[Ka08] Kapow Technologies, http://www.kapowtech.com

[Od08a] ODI home page: http://www-odi.nhtsa.dot.gov/index.cfm

[Od08b] Database schema is available at http://www-odi.nhtsa.dot.gov/downloads/folders/Complaints/CMPL.txt

[Ru07] Russom, P.: BI Search and Text Analytics, TDWI Best Practices Report, 2007; pp. 9-11

[Sn07] http://www.ihtsdo.org/our-standards

[Tr08] Tripadvisor Travel Web Site; http://www.tripadvisor.com

[UI04] Unstructured Information Management, Special Issue of IBM Systems Journal, Vol.43,No. 3, 2004

[UI08] Website of the Apache incubator project: http://incubator.apache.org/uima/

# Kompressionstechniken für spaltenorientierte BI-Accelerator-Lösungen

Christian Lemke[1,2] Kai-Uwe Sattler[2] Franz Färber[1]
[1]SAP AG, Walldorf, Germany
[2]FG Datenbanken & Informationssysteme, TU Ilmenau, Germany

**Abstract:** BI-Accelerator-Lösungen wie SAP's TREX ermöglichen durch die Kombination von spaltenorientierter Datenorganisation, hauptspeicherbasierter Verarbeitung und skalierbarer Multiserver-Architektur eine deutliche Beschleunigung bei der Verarbeitung komplexer OLAP-Anfragen in riesigen Data Warehouses. Durch den Einsatz von Datenkompressionstechniken lässt sich der Speicherbedarf der Spalten und damit auch die Verarbeitungszeit weiter reduzieren. In diesem Beitrag untersuchen wir daher Verfahren zur Spaltenkompression und deren Implementierung in TREX. Da für eine effektive Kompression eine Sortierung der Werte pro Spalte notwendig ist, stellen wir weiterhin Strategien zur Optimierung der Spaltenreihenfolge für die Sortierung vor.

## 1 Einführung

In den vergangenen Jahren sind die Anforderungen an Data-Warehouse- und OLAP-Technologien ständig gestiegen. Heutzutage müssen Giga-, Tera- oder Petabytes an Daten verwaltet und komplexe analytische Anfragen von mehreren hundert Nutzern gleichzeitig beantwortet werden können. Weiterhin wächst der Bedarf von Kunden nach Ad-hoc- bzw. Realtime-Auswertungen, die eine Vorberechnung von Reports unmöglich machen.

Betrachtet man klassische relationale Datenbanksysteme, die noch immer die dominierende Technologie im Backend-Bereich darstellen, so stellt der externe Speicherzugriff den limitierenden Faktor bei der Erfüllung dieser Anforderungen dar. Trotz aller Forschritte bei Festplattenspeicher klafft immer noch eine große Lücke zwischen den Zugriffszeiten im Hauptspeicher und auf Externspeicher. Zur Überwindung dieser Zugriffslücke werden aktuell drei wesentliche Ansätze verfolgt:

1. Die vom Externspeicher zu lesenden Daten werden auf das nötigste beschränkt und so schnell wie möglich gelesen. Neben geeigneten Indexstrukturen besitzen insbesondere spaltenorientierte Speicherorganisationsformen (so genannte *Column-Stores*) großes Potential [SAB+05, BZN05].

2. Durch das Halten und Verarbeiten der Daten im *Hauptspeicher* lassen sich I/O-Zugriffe komplett vermeiden [DKO+84].

3. Schließlich kann die Verarbeitungskapazität einer großen Zahl günstiger Serversysteme (Blades) durch *parallele Verarbeitung* der Anfragen ausgenutzt werden.

Im Bereich von Business-Intelligence-Technologien werden diese Ideen gegenwärtig im Rahmen so genannter BI-Accelerator-Lösungen kombiniert. Ein Beispiel hierfür ist SAP's NetWeaver BI Accelerator TREX [LLR06]. TREX basiert auf einer skalierbaren Multiserver-Architektur aus kostengünstigen Blades. Die Verarbeitung findet komplett im Hauptspeicher statt, wobei Fakten- und Dimensionstabellen spaltenorientiert organisiert (vertikal partitioniert) und die Spalten wiederum horizontal über die einzelnen Knoten partitioniert sind. Ungenutzte Spalten werden mittels einer LRU-Strategie aus dem Speicher verdrängt und können ggf. später wieder vom Externspeicher nachgeladen werden. Die Hauptspeicher- und parallele Verarbeitung erlaubt die interaktive Beantwortung von OLAP-Anfragen ohne Voraggregation.

Ein einfache Verwaltung der Daten im Hauptspeicher stößt jedoch speziell bei sehr großen Warehouse-Installationen schnell an ihre Grenzen, da RAM im Vergleich zu Festplattenspeicher teuer ist und nicht beliebig vergrößert werden kann. Daher werden geeignete Kompressionstechniken benötigt, die das Datenvolumen verringern und damit auch die Ausnutzung von CPU-Caches verbessern.

In diesem Beitrag diskutieren wir geeignete Kompressionstechniken für spaltenorientierte Speicherorganisation im Kontext des BI-Accelerators TREX. Da eine effektive Komprimierung die Sortierung der Spalten erfordert, hat die Reihenfolge der Spaltensortierung einen großen Einfluss auf die erzielbare Kompressionsrate. Wir untersuchen daher weiterhin das Problem der optimalen Spaltenreihenfolge und diskutieren geeignete Heuristiken. Die vorgestellten Techniken werden anschließend einer Evaluierung unterzogen.

## 2 Verwandte Arbeiten

Zur Speicherung und Organisation von Daten in Datenbanken existieren zwei grundlegende Konzepte: zeilenorientierte und spaltenorientierte Datenbanken (Row-Stores und Column-Stores). In letzter Zeit wurden mehrere Untersuchungen [HLAM06, HBND06, AMH08, HD08] durchgeführt, die beide Ansätze unter Berücksichtigung verschiedener Kriterien vergleichen und für das jeweilige Einsatzszenario Empfehlungen geben. Wird beispielsweise nur ein sehr kleiner Teil der vorhandenen Attribute in einer Anfrage verwendet und ändern sich die Daten relativ selten, dann ist ein Column-Store vorzuziehen, weil hier nur die benötigten Spalten geladen und verarbeitet werden müssen. Zwei bekannte Vertreter spaltenorientierter Datenbanken sind C-Store [SAB+05] und MonetDB/X100 [BZN05]. Im Gegensatz zu [HLAM06, HD08], wo die beiden Organisationsformen nur auf der Ebene des Plattenzugriffs und nicht hinsichtlich angepasster Operatoren verglichen werden, konzentriert sich [AMH08] auf die Anfrageausführung. Dahingehend wird in der Arbeit versucht, einen Column-Store unter Zuhilfenahme unterschiedlicher Herangehensweisen in einem Row-Store zu emulieren und herauszufinden, welche spaltenorientierten Optimierungen den größten Einfluss auf die Geschwindigkeit haben. Einen anderen Ansatz verfolgen [HBND06], die einen Row-Store auf lesenden Zugriff optimieren und in einem gemeinsamen Framework einer spaltenorientierten Datenbank gegenüberstellen.

Um auch mit großen Datenmengen effizient umgehen zu können, ist es notwendig die

Daten zu komprimieren und auf den komprimierten Daten zu arbeiten. Der Einsatz von Kompression in Datenbanken ist schon seit Anfang an Gegenstand der Forschung, wobei das Ziel früher Arbeiten [Als75, Cor85] hauptsächlich die Reduzierung des Speicherplatzbedarfs und des Datentransfers war. Spätere Veröffentlichungen [GS91, WKHM00] konzentrierten sich mehr auf die höhere Geschwindigkeit, die sich durch das Arbeiten auf komprimierten Daten erreichen lässt. So müssen bei Operationen wie Verbund oder Selektion nur noch kleinere Datenmengen verglichen werden und durch den zusätzlich verfügbaren Speicher können mehr häufig genutzte Seiten im Zwischenspeicher vorgehalten werden.

Neben vielen Arbeiten [Rv93, GRS98, CGK01, RS06, HRSD07, RSQ+08, SAB+05, ZHNB06] , die sich mit der Entwicklung neuer und der Zusammenfassung vorhandener Kompressionstechniken befassen, gibt es auch einige Veröffentlichungen [IW94, WKHM00, CGK01, AMF06], die auf das Problem der Integration von Kompression in Datenbanken eingehen. Obwohl die meisten neueren Publikationen [WKHM00, CGK01, SAB+05, AMF06, BZN05, ZHNB06] nur leichtgewichtige Kompressionstechniken vorschlagen, weil sonst die Kosten für die Dekompression den Gewinn durch den schnelleren Datentransfer übersteigen würden, setzen [RS06, HRSD07, RSQ+08] auf eine modifizierte Huffman-Kodierung [Huf52]. Sie zeigen, dass trotz einer schwergewichtigen Kodierung eine effiziente Anfragebearbeitung möglich ist.

Ein weiterer Aspekt, in dem sich die Techniken unterscheiden, besteht in der Granularität der Dekompression. Während erste Implementierungen in kommerziellen, relationalen Datenbanksystemen nur ganze Seiten entpacken konnten, was bei vielen ungleichmäßig verteilten Einzelzugriffen auf mehrere Seiten einen sehr großen Overhead darstellt, untersuchten [Cor85, IW94, GRS98] bereits die Vorteile einer zeilenweisen Dekompression. OLAP-Anfragen verwenden jedoch häufig nur einen kleinen Teil der Spalten, wodurch es nötig wird, auch effizient auf einzelne Werte zugreifen zu können. Da auch normale Datenbankanfragen davon profitieren, empfehlen besonders aktuelle Arbeiten [Als75, GS91, GRS98, WKHM00, CGK01] Kompressionstechniken, die einen feingranularen Zugriff unterstützen. Dazu gehören auch die neueren Publikationen [SAB+05, AMF06, ZHNB06, Aba07], die spezielle Kompressionsverfahren für eine spaltenorientierte Organisation präsentieren und teilweise auch deren Integration beschreiben. Neben der Kodierung von NULL-Werten, die abhängig von der Häufigkeit erfolgt, schlägt [Aba07] weitere Anwendungsszenarios für Column-Stores vor, die über Data Warehousing und OLAP hinausgehen.

Für eine optimale Kompression in einem Column-Store spielt abgesehen von der Kompressionstechnik auch die Reihenfolge der Zeilen eine große Rolle. [OR86] ist eine frühe Arbeit, die sich mit dem Umsortieren von Daten beschäftigt, um eine bessere Kompressionsrate zu erzielen. Der Fokus liegt hier jedoch auf einer zeilenorientierten Organisation und auf der Kompression von Nullwerten. Wir hingegen gehen davon aus, dass es nicht nur einen, sondern mehrere häufige Werte in einer Spalte geben kann, die optimal angeordnet werden müssen. Im Kontext von Hybrid OLAP (HOLAP), einer Kombination von relationalem OLAP (ROLAP) und multidimensionalem OLAP (MOLAP), stellt [KL03] einige Heuristiken vor, um eine effiziente Organisation der Daten zu erzielen. In einem HOLAP System werden dichte Regionen in einem mehrdimensionalem Array gespeichert,

Dictionary

| pos | value |
|---|---|
| 0 | Aachen |
| 1 | Karlsruhe |
| 2 | Leipzig |
| 3 | Münster |

IndexVector

| pos | value |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 3 |

Abbildung 1: Beispiel für das *domain coding*.

IndexVector (uncompressed)

| pos | value |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 3 |

IndexVector (indirect coded, values)

| pos | value |
|---|---|
| 0/1/2 | 0 |
| 3 | 1 |
| 4 | 0 |
| 5 | 0 |
| 6 | 2 |
| 7 | 3 |

IndexVector (indirect coded, offsets) / Integer (start pos)

| pos | value | pos |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | |
| 2 | 0 | |
| 3 | 1 | |
| 0 | | 2 |
| 1 | | |
| 2 | | |
| 3 | | |

Abbildung 2: Beispiel für das *indirect coding* mit einer Blockgröße von 4 Werten.

um schnell auf sie zugreifen zu können, und Daten aus dünn besetzten Regionen in einem relationalen Schema abgelegt, um Speicherplatz zu sparen.

## 3 Techniken zur Spaltenkompression

Wie bereits erwähnt, liegt der Fokus dieser Arbeit auf leichtgewichtigen Kompressionstechniken für Column-Stores und deren Implementierung in SAP's TREX zur weitergehenden Evaluierung im Abschnitt 5. Die vorgestellten Verfahren erlauben den effizienten Zugriff sowohl auf einzelne Werte als auch auf Blöcke von Werten, wobei Letzteres durch optimierte Algorithmen besonders schnell durchgeführt werden kann. Als Basis für alle folgenden Techniken wird das *domain coding* [WKHM00, AMF06, ZHNB06] verwendet, bei dem die in einer Spalte vorkommenden Werte in einem Wörterbuch (Dictionary) sortiert abgelegt und dann nur noch bit-komprimierte Verweise (IndexVector) gespeichert werden. Dadurch benötigt die Kodierung mit minimalen Bitlängen von $n$ Ganzzahlen und $u$ unterschiedlichen Werten nur $n\lceil \log_2 u \rceil$ Bits . Abbildung 1 zeigt das domain coding anhand der Beispieldaten *'Aachen, Aachen, Aachen, Karlsruhe, Aachen, Aachen, Leipzig, Münster'*, wobei jeder Wert im IndexVector durch zwei Bits dargestellt wird und die kursiv dargestellten Spalten nur logisch vorhanden sind, da sie sich implizit durch den Kontext ergeben. Das Verwenden von Ganzzahlen statt den Originalwerten bringt darüber hinaus auch Geschwindigkeitsvorteile, weil das zu verarbeitende Datenvolumen kleiner ist und die Prozessoren auf diesen Datentyp optimiert sind. So passen beispielsweise mehr Daten in den viel schnelleren Cache und mehrere Werte können mit speziellen Prozessorbefehlen parallel bearbeitet werden (SIMD-Prinzip).

Die einfachste Kompressionstechnik ist *prefix coding*, bei der gleiche Werte am Anfang (Präfix $p$) weggelassen und stattdessen ein Wert und die Häufigkeit gespeichert werden. Für eine Spalte mit $n$ Elementen und $u_{col}$ unterschiedlichen Werten sind auf diese Weise $(n-p)\lceil \log_2 u_{col} \rceil + 64$ Bits erforderlich. In Abbildung 3 sieht man links den IndexVector der Beispieldaten unkomprimiert und rechts nach der Anwendung des prefix codings.

Wenn nun der häufigste Wert nicht nur am Anfang steht, sondern auch unregelmäßig zwischen den anderen Werten sehr oft vorkommt, dann lässt sich durch das *sparse co-*

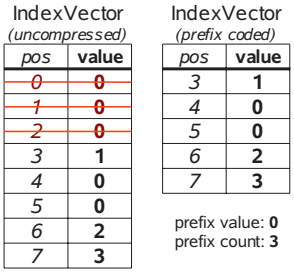



Abbildung 3: Example for prefix coding.

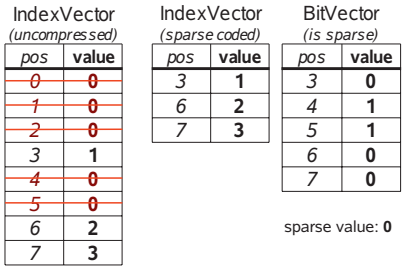



Abbildung 4: Example for sparse coding.

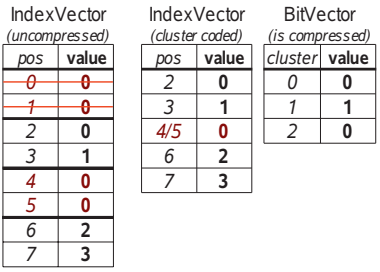



Abbildung 5: Example for cluster coding with a block size of two values.

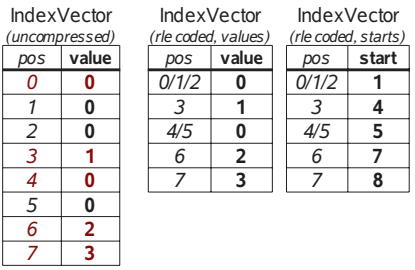



Abbildung 6: Example for run length coding.

*ding* (Abbildung 4) eine gute Kompression erreichen. Hier werden alle Vorkommen $f$ des häufigsten Wertes entfernt und in einem Bitvektor die Positionen gespeichert. Weiterhin ist es möglich, für den Bitvektor das prefix coding anzuwenden, wodurch bei einem großen Präfix $p$ noch zusätzlich viel eingespart werden kann. Bei dieser Technik beträgt der Speicherbedarf $(n-f)\lceil \log_2 u_{col} \rceil + (n-p) + 32$ Bits.

Alle im Folgenden beschriebenen Verfahren nutzen das Prinzip vom prefix coding und arbeiten auf Böcken von Daten, in denen möglichst wenig verschiedene Werte vorkommen dürfen, um eine gute Kompressionsrate zu erzielen. So werden beim *cluster coding* ausschließlich Blöcke mit einem unterschiedlichen Wert komprimiert, indem nur der auftretende Wert gespeichert wird. Weiterhin muss man sich in einem Bitvektor merken, welche Blöcke komprimiert wurden, um das Original wieder herstellen zu können. Die Bestimmung der optimalen Blöckgröße und deren Einfluss auf die Kompressionsrate und Geschwindigkeit wird in diesem Beitrag aus Zeigründen nicht weiter untersucht. Die Anzahl der Elemente sollte auf jeden Fall eine Zweierpotenz sein, weil dann statt Multiplikation und Modulo die schnellere Bitverschiebung und -verknüpfung verwendet werden kann. Abbildung 5 zeigt ein Beispiel, in dem ein Block aus zwei Werten besteht und die komprimierten Elemente dunkelrot dargestellt sind.

Enthalten die Datenblöcke mehr als einen aber trotzdem nur wenig unterschiedliche Werte, dann kommt *indirect coding* zum Einsatz. Hier wird für lohnenswerte Blöcke das *domain coding* verwendet, was eine weitere Indirektionsstufe einführt und für jeden Block ein ei-

genes Mini-Wörterbuch erforderlich macht. Um die Anzahl der Wörterbücher und somit den Speicherbedarf zu reduzieren, können aufeinanderfolgende Blöcke ein Wörterbuch weiter benutzen, falls sie durch neue Einträge nicht die Anzahl der zur Repräsentation benötigten Bits erhöhen. Bei einer Spalte mit $u_{col}$ unterschiedlichen Werten wird ein Block mit $k$ Einträgen und $u_{block}$ unterschiedlichen Werten genau dann als lohnenswert bezeichnet, wenn die Größe des Wörterbuches und der Referenzen kleiner ist als die unkomprimierten Daten (nur *domain coding*):

$$u_{block}\lceil \log_2 u_{col} \rceil + k\lceil \log_2 u_{block} \rceil < k\lceil \log_2 u_{col} \rceil$$

Die in der Implementierung verwendeten Datenstrukturen sind anhand eines Beispiels in Abbildung 2 veranschaulicht, wobei ein Block aus vier Werten besteht und die komprimierten Elemente dunkelrot dargestellt sind. Die Wörterbücher und die unkomprimierten Daten werden in dem mittleren IndexVector (*values*) gespeichert und pro Block über einen Startposition (*start pos*) adressiert. Komprimierte Blöcken besitzen in der rechten Datenstruktur zusätzlich einen IndexVector (*offsets*), der die Verweise auf die einzelnen Werte enthält.

Die letzte hier vorgestellte Kompressionstechnik *run length coding* ist eine leicht modifizierte Variante vom *run-length encoding* [Gol66], die gleiche aufeinanderfolgende Werte zusammenfasst, indem sie nur noch einen Wert und die Anzahl des Auftretens speichert. Wenn nun zu einem Wert die Positionen des Auftretens ermittelt werden sollen, wäre es notwendig, die Häufigkeiten der vorhergehenden Werte aufzusummieren. Da dies jedoch recht aufwendig werden kann, haben wir uns entschlossen, auf Kosten der Kompressionsrate die Startposition und nicht die Anzahl zu merken. Für die Beispieldaten ist dies in Abbildung 6 illustriert.

In der folgenden Tabelle wird noch einmal zusammengefasst, in welchen Fällen welche Verfahren eingesetzt werden können:

| **Fall (Werteverteilung)** | **Kompressionstechniken** |
|---|---|
| *ein* sehr häufiger Wert *am Anfang* | prefix coding |
| *ein* sehr häufiger Wert *weit verstreut* | sparse coding |
| viele Blöcke mit nur *einem* Wert | cluster, run length coding |
| viele Blöcke mit *wenig verschiedenen* Werten | indirect coding |
| wenig verschiedene, *zusammenhängende* Werte | run length coding |
| *sehr viele* verschiedene Werte | nur domain coding |

## 4 Optimierung der Spaltenreihenfolge

Alle im vorherigen Abschnitt vorgestellten Techniken können nur dann eine optimale Kompressionsrate erzielen, wenn die Daten nach bestimmten Kriterien sortiert sind. So profitieren bis auf das run length coding alle Verfahren davon, die Vorkommen des häufigsten Wertes an den Anfang zu verschieben und diese somit nicht mitkodieren zu müssen. Für cluster, indirect und run length coding ist es weiterhin notwendig die Daten

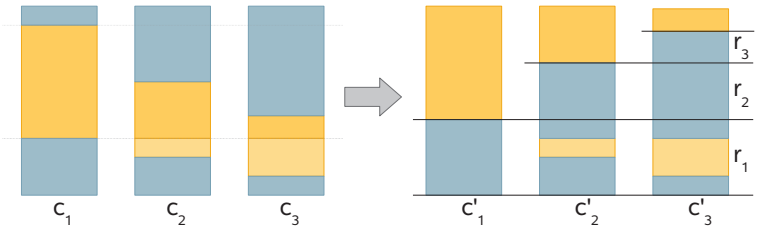



Abbildung 7: Beispiel für das Verschieben des häufigsten Wertes an den Anfang.

so anzuordnen, dass Blöcke mit möglichst wenig verschiedenen Werte entstehen. Aufgrund des mit der Anzahl der Spalten exponentiell anwachsenden Lösungsraumes ist das Suchen nach einer optimalen Lösung für dieses Optimierungsproblem praktisch undurchführbar. Bereits [Als75] hat zeigt, dass das Sortieren unter Verwendung von RLE ein NP-vollständiges Problem ist. Als Konsequenz wurden im Zuge dieser Arbeit mehrere Greedy-Heuristiken entwickelt, die einen Kompromiss zwischen der Laufzeit der Sortierung und der erzielten Speichereinsparung darstellen.

Der gesamte Optimierungsprozess einer Tabelle läuft in vier Schritten ab:

1. Überprüfung, ob eine Sortierung notwendig ist
2. Bestimmung lohnenswerter Spalten (Kandidaten)
3. Anwendung einer Heuristik
4. Bestimmung der besten Kompressionstechnik für jeden Kandidaten

Im ersten Schritt wird überprüft, ob sich genügend Daten verändert haben und damit eine neue Sortierung die Kompressionsrate verbessern kann. Danach werden alle Spalten bestimmt, die mehr als einen aber nicht zu viele unterschiedliche Werte besitzen, da sonst eine Umordnung keine Verbesserungen mit sich bringen würde. Nachdem eine der gleich vorgestellten Heuristiken mit der Optimierung fertig ist, wird für jeden Kandidaten die beste Kompressionstechnik bestimmt und angewendet.

Die einfachste Heuristik wird vom *FValueOptimizer* implementiert und verschiebt den häufigsten Wert jeder Spalte an den Anfang, wobei die Abhängigkeit der Spalten untereinander beachtet werden muss. Die Spalten können nicht unabhängig voneinander sortiert werden, da die Position der Werte die Zugehörigkeit zu einer Zeile ausdrückt und im Falle einer freien Sortierung ein teures Mapping von Spaltenwerten zu Zeilen erforderlich wäre. Abbildung 7 zeigt ein kleines Beispiel mit drei Spalten gleicher Bitbreite $(c_1, c_2, c_3)$, in dem die Spalte 1 zuerst sortiert wird, weil dort im Vergleich zu den anderen Spalten die Häufigkeit des häufigsten Wertes (gelber Bereich) und somit die Einsparung am größten ist. Die Bereiche $(r_1, r_2, r_3)$, die durch das Verschieben entstehen, werden im weiteren Verlauf als Restbereiche bezeichnet und können durch fortgeschrittene Verfahren weiter sortiert werden, ohne dass die bestehende Ordnung zerstört wird.

Alle Heuristiken, die auf dem FValueOptimizer aufbauen, sind durch den *FValueBlockOptimizer* realisiert und unterscheiden sich allein durch die Ermittlung der Reihenfolge, in der die Spalten sortiert werden. Um in den Restbereichen möglichst große, zusam-

menhängende Bereiche mit gleichen Werten zu erzeugen, erfolgt die Sortierung der Werte absteigend nach der Häufigkeit ihres Vorkommens. Falls diese Bereiche größer als die in den Kompressionstechniken verwendeten Blockgröße sind, werden sie zu der Liste der aktuellen Restbereiche hinzugefügt und in der nächsten Spalte mit berücksichtigt. Die folgende Tabelle fasst mögliche Strategien zusammen:

| **Strategie** | **Sortierreihenfolge** |
|---|---|
| reversed | umgekehrte Verschiebereihenfolge |
| single value blocks | $\sim$ Anzahl der Blöcke mit einem unterschiedlichen Wert |
| average distinct values | $\sim \varnothing$ Anzahl unterschiedlicher Werte pro Block |
| valuable blocks | $\sim$ Anzahl lohnenswerter (komprimierbarer) Blöcke |

Die *reversed* Strategie sortiert die Restbereiche entgegengesetzt der Reihenfolge, in welcher der häufigste Wert an den Anfang verschoben wird, da auf diese Weise die bis jetzt am wenigsten optimierten Spalten bevorzugt werden. Wenn nun eine Gleichverteilung und eine identische Anzahl unterschiedlicher Werte in allen Spalten angenommen wird, dann sinkt nach dem Entfernen eines großen Präfixes die Wahrscheinlichkeit große Bereiche mit gleichen Werten formen zu können und damit eine hohe Kompressionsrate zu erreichen. Während *single value blocks* (s. v. b.) versucht, die Kompression mit cluster coding zu verbessern, dienen die Maße bei *average distinct values* (a. d. v.) und *valuable blocks* (v. b.) dazu, eine gute Sortierreihenfolge für die Spalten beim Einsatz von indirect coding zu bestimmen. In den letzten drei Strategien wird wie beim FValueOptimizer die Anzahl der Bits eingerechnet, die zur Kodierung der Werte in den einzelnen Spalten notwendig sind, um die jeweilige Einsparung abzuschätzen.

Einen etwas anderen Ansatz verfolgen die durch den *BlockOptimizer* implementierten Heuristiken. Hier fängt die Sortierung unabhängig von dem häufigsten Wert in den einzelnen Spalten mit einem großen Restbereich an, der die gesamten Daten repräsentiert. Die Sortierreihenfolge bestimmt sich aus der Anzahl der unterschiedlichen Werte einer Spalte, wobei entweder mit dem Minimum (*min value count*) oder Maximum (*max value count*) angefangen wird.

In unserer Implementierung der vorgestellten Heuristiken werden bei der Sortierung einer Spalte nicht die Werte in allen Spalten getauscht, sondern nur die Einträge in einem Mapping, das die neue Zeilenreihenfolge bestimmt. Nach der Umordnung einer Spalte auf diese Weise kann man die Daten nicht mehr blockweise bearbeiten, sondern es werden viele Einzelzugriffe notwendig, die viel teuerer sind. Aus diesem Grund werden die häufigsten Werte im FValueOptimizer und die Maße im FValueBlockOptimizer nicht in jedem Schritt neu berechnet.

## 5 Evaluierung

Das Ziel dieses Abschnittes ist die Evaluierung der vorgestellten Heuristiken mit Hilfe der modifizierten TPC-H Daten und Anfragen aus [HRSD07]. Dort wird eine Projektion mit neun Spalten und 32 Millionen Zeilen aus den TPC-H Tabellen *customer*, *supplier*, *line-*

| Heuristik | Größe in KB | Laufzeit in ms |
|---|---|---|
| *keine* | *321 245* | - |
| fvalue | 262 527 | 1 886 |
| fvalue[1] | 285 170 | - |
| fvalue[2] | 262 500 | 7 647 |
| reversed | 136 042 | 6 322 |
| single value blocks (s. v. b.) | 160 898 | 8 329 |
| single value blocks (s. v. b.)[3] | 163 263 | 19 628 |
| average distinct values (a. d. v.) | 136 671 | 10 103 |
| average distinct values (a. d. v.)[3] | 150 540 | 19 204 |
| valuable blocks (v. b.) | 136 692 | 9 669 |
| min value count (min v. c.) | 204 609 | 211 264 |
| *max value count (max v. c.)* | *108 090* | 2 030 |

[1] für alle Attribute wurde *prefix coding* und nicht die beste Kompressionstechnik verwendet

[2] die häufigsten Werte wurden in jedem Schritt für den aktuellen Bereich neu bestimmt

[3] die Maße wurden in jedem Schritt für die aktuellen Restbereiche neu bestimmt

Tabelle 1: Erreichte Größe und für die Optimierung benötigte Zeit aller vorgestellten Heuristiken.

*item* und *order* erzeugt, wobei zusätzlich eine ungleichmäßige Datenverteilung gewählt wurde, um den Vorteil der Huffman-Kodierung zu zeigen. So entspricht die Verteilung der Länder von Lieferanten und Kunden dem kanadischen Handelsverkehr der Welthandelsorganisation. Darüber hinaus umfassen die Datumsangaben zu 99% Wochentage, wovon 40% auf Weihnachten und Muttertag fallen, und die Jahre von 3 bis 9999, wobei 99% im Bereich von 1995 und 2005 liegen.

Weil in [HRSD07] weiterhin eine andere Architektur (Row-Store) verwendet wird, können die Ergebnisse nicht ohne weiteres verglichen werden, was jedoch auch nicht Gegenstand dieser Arbeit ist.

Die Tabelle 1 enthält für alle vorgestellten Heuristiken die erreichte Größe in *KB* und die für die Optimierung benötigte Zeit in *Millisekunden*. Bei dem *cluster coding* und *indirect coding* wurde eine Blockgröße von 1024 Elementen verwendet. Bereits das Verschieben des häufigsten Wertes an den Anfang und die Verwendung von *prefix coding* bringt eine Einsparung von 11% (36 075 KB). Mit den optimalen Kompressionstechniken sind es sogar 18% (weitere 22 643 KB). Die Messungen zeigen ebenfalls, dass der Mehraufwand, in jedem Schritt den häufigsten Wert neu zu bestimmen, sich nicht lohnt, da die Verbesserung in der Kompression vernachlässigbar ist.

Sehr gute Ergebnisse erreicht auch die sehr einfache *reversed* Heuristik, die den Speicherbedarf bei den Testdaten um 58% (185 203 KB) senkt! Wie bei dem FValueOptimizer wurde auch für den FValueBlockOptimizer die Bestimmung der Maße in jedem Schritt implementiert, um mögliche Verbesserungen oder Verschlechterungen analysieren zu können. Im schlimmsten Fall hat sich die Kompressionsrate um 4% (von 136 671 KB auf 150 540 KB) verschlechtert.

Die längste Laufzeit wurde für die *min value count* Heuristik gemessen, welche die meis-

| Heuristik | nur domain coding | prefix coding | sparse coding | cluster coding | indirect coding | run length coding |
|---|---|---|---|---|---|---|
| *keine* | 9 | - | - | - | - | - |
| fvalue | 5 | 2 | 1 | - | 1 | - |
| reversed | 2 | 2 | 1 | - | 1 | 3 |
| s. v. b. | 1 | 1 | 1 | 2 | 4 | - |
| a. d. v. | 2 | 1 | 1 | 1 | 1 | 3 |
| v. b. | 2 | 2 | 1 | - | 1 | 3 |
| min v. c. | 3 | - | - | - | 2 | 4 |
| max v. c. | 2 | - | 2 | - | 2 | 3 |

Tabelle 2: Anzahl der Attribute, welche die jeweilige Kompressionstechnik verwenden.

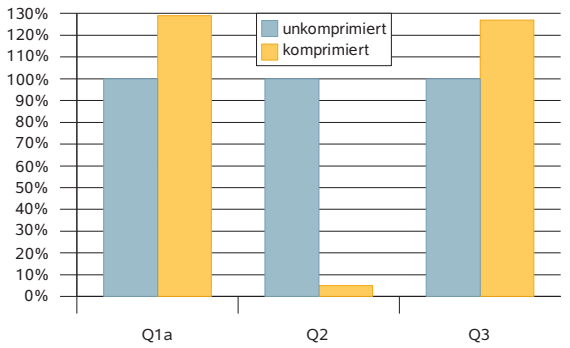



Abbildung 8: Vergleich von Anfragen aus [HRSD07] mit und ohne weitere Kompression.

ten Restbereiche bearbeitet und dafür viele teure Einzelzugriffe benötigt hat. Die beste Kompressionsrate (66%) hingegen wurde von *max value count* erzielt, wobei das Ergebnis relativ stark von einer bestehenden Ordnung abhängig ist, da nur relativ wenig Restbereiche und Werte sortiert wurden. So konnten mit einer guten Vorsortierung auch 74% (82 613 KB) erreicht werden.

Als Nächstes zeigt die Tabelle 2 wie viele Attribute die jeweilige Kompressionstechnik verwenden, nachdem die entsprechende Heuristik angewendet wurde. Hier ist deutlich zu sehen, dass *cluster coding* und *sparse coding* sehr selten automatisch ausgewählt wurden. Die Ursache liegt in dem Overhead, der beim ersten Verfahren für große Bereiche mit gleichen Werten auftritt, wenn für jeden Block der enthaltene Wert wiederholt gespeichert werden muss, was beispielsweise beim *run lenght coding* nur einmal für den gesamten Bereich erforderlich ist. Im Gegensatz dazu amortisiert sich der unter Umständen sehr große Bitvektor beim *sparse coding* erst, wenn ein Wert sehr häufig und verstreut vorkommt. Da für die meisten Attribute die Daten nach der Optimierung jedoch sortiert vorliegen, sind die anderen Kompressionsverfahren effektiver.

Zuletzt wird in Abbildung 8 die Geschwindigkeit von verschiedenen Anfragen mit und

ohne weitere Kompression verglichen, wobei die Laufzeit einer Anfrage unter ausschließlicher Verwendung von *domain coding* als 100% angegeben ist und die Daten mit der *min value count* Heuristik sortiert wurden, weil damit die höchste Kompressionsrate erzielt wurde. In der Anfrage *Q1a* wird eine Gruppierung und Aggregation durchgeführt, im Gegensatz zu den anderen beiden Anfragen, die außerdem zwei Gleichheitsprädikate (*Q2*) beziehungsweise ein Bereichsprädikat (*Q3*) anwenden. Es zeigt sich, dass die zusätzliche Kompression in den meisten Fällen die Anfrageverarbeitung kaum ausbremst, sondern sogar beschleunigen kann, wenn man die Eigenschaften einzelner Kompressionstechniken geschickt ausnutzt.

## 6 Zusammenfassung

In diesem Beitrag haben wir Techniken der Datenkompression für Column-Stores am Beispiel des BI-Accelerators TREX vorgestellt und evaluiert. Die auf der Idee des domain coding basierenden Verfahren ermöglichen eine effizienten Zugriff auf Datenblöcke wie auch auf einzelne Werte. Zur Erzielung einer hohen Kompressionsrate werden die Spaltenwerte zu Blöcken zusammengefasst, die nur wenige verschiedene Werte enthalten. Aus der dafür notwendigen Sortierung leitet sich die Aufgabe der Bestimmung einer optimalen Reihenfolge der Spalten bei der Sortierung ab. Da es sich hierbei um ein NP-vollständiges Problem handelt, haben wir in dieser Arbeit verschiedene Greedy-Lösungen vorgestellt. Die Ergebnisse der Evaluierung zeigen, dass mit unseren Heuristiken gute Kompressionsraten erzielt werden können und dass auch die Verarbeitung von Anfragen davon profitiert.

## Literatur


[Aba07] D. J. Abadi. Column-Stores For Wide and Sparse Data. In *Proc. CIDR*, Seiten 292–297, 2007.

[Als75] P. A. Alsberg. Space and Time Savings Through Large Data Base Compression and Dynamic Restructuring. *Proc. IEEE*, 63(8):1114–1122, 1975.

[AMF06] D. J. Abadi, S. R. Madden und M. C. Ferreira. Integrating Compression and Execution in Column-Oriented Database Systems. In *Proc. SIGMOD*, Seiten 671–682, 2006.

[AMH08] D. J. Abadi, S. R. Madden und N. Hachem. Column-Stores vs. Row-Stores: How Different Are They Really? In *Proc. SIGMOD*, Seiten 967–980, 2008.

[BZN05] P. Boncz, M. Zukowski und N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proc. CIDR*, Seiten 225–237, 2005.

[CGK01] Z. Chen, J. Gehrke und F. Korn. Query Optimization In Compressed Database Systems. *SIGMOD Rec.*, 30(2):271–282, 2001.

[Cor85] G. V. Cormack. Data Compression on a Database System. *Commun. ACM*, 28(12):1336–1342, 1985.

[DKO+84] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. Stonebraker und D. Wood. Implementation Techniques for Main Memory Database Systems. *SIGMOD Rec.*, 14(2):1–8, 1984.

[Gol66] S. W. Golomb. Run-Length Encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[GRS98] J. Goldstein, R. Ramakrishnan und U. Shaft. Compressing Relations and Indexes. In *Proc. 14th ICDE*, Seiten 370–379, 1998.

[GS91] G. Graefe und L. D. Shapiro. Data Compression and Database Performance. In *Proc. ACM/IEEE-CS Symp. on Applied Computing*, Seiten 22–27, 1991.

[HBND06] A. Halverson, J. L. Beckmann, J. F. Naughton und D. J. DeWitt. A Comparison of C-Store and Row-Store in a Common Framework. Bericht 1570, University of Wisconsin-Madison, 2006.

[HD08] A. L. Holloway und D. J. DeWitt. Read-Optimized Databases, In Depth. In *Proc. 34th VLDB*, 2008.

[HLAM06] S. Harizopoulos, V. Liang, D. J. Abadi und S. R. Madden. Performance Tradeoffs in Read-Optimized Databases. In *Proc. 32nd VLDB*, Seiten 487–498, 2006.

[HRSD07] A. L. Holloway, V. Raman, G. Swart und D. J. DeWitt. How to Barter Bits for Chronons: Compression and Bandwidth Trade Offs for Database Scans. In *Proc. SIGMOD*, Seiten 389–400, 2007.

[Huf52] D. Huffman. A Method for Construction of Minimum-Redundancy Codes. *Proc. IRE*, 40(9):1098–1101, 1952.

[IW94] B. R. Iyer und D. Wilhite. Data Compression Support in Databases. In *Proc. 20th VLDB*, Seiten 695–704, 1994.

[KL03] O. Kaser und D. Lemire. Attribute Value Reordering For Efficient Hybrid OLAP. In *Proc. 6th DOLAP*, Seiten 1–8, 2003.

[LLR06] T. Legler, W. Lehner und A. Ross. Data Mining with the SAP NetWeaver BI Accelerator. In *Proc. 32nd VLDB*, Seiten 1059–1068, 2006.

[OR86] F. Olken und D. Rotem. Rearranging Data to Maximize the Efficiency of Compression. In *Proc. 5th PODS*, Seiten 78–90, 1986.

[RS06] V. Raman und G. Swart. How to Wring a Table Dry: Entropy Compression of Relations and Querying of Compressed Relations. In *Proc. 32nd VLDB*, Seiten 858–869, 2006.

[RSQ+08] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang und R. Sidle. Constant-Time Query Processing. In *Proc. 24th ICDE*, Seiten 60–69, 2008.

[Rv93] M. A. Roth und S. J. van Horn. Database Compression. *SIGMOD Rec.*, 22(3):31–39, 1993.

[SAB+05] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. C. Ferreira, E. Lau, A. Lin, S. R. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran und S. Zdonik. C-Store: A Column-oriented DBMS. In *Proc. 31st VLDB*, Seiten 553–564, 2005.

[WKHM00] T. Westmann, D. Kossmann, S. Helmer und G. Moerkotte. The Implementation and Performance of Compressed Databases. *SIGMOD Rec.*, 29(3):55–67, 2000.

[ZHNB06] M. Zukowski, S. Héman, N. Nes und P. Boncz. Super-Scalar RAM-CPU Cache Compression. In *Proc. 22nd ICDE*, Seite 59, 2006.

# Pre-Caching hochdimensionaler Aggregate mit relationaler Technologie

Jens Albrecht, Marc Fiedler, Jürgen Görlich, Matthias Lemm und Thomas Ruf
{jens.albrecht, marc.fiedler, juergen.goerlich, matthias.lemm, thomas.ruf}@gfk.com
GfK Retail and Technology GmbH, Nordwestring 101, 90319 Nürnberg

**Abstract:** Die GfK Retail and Technology produziert Berichte für einige hundert Warengruppen auf Basis eines zentralen Data Warehouse. Die umfassenden und detailreichen, d.h. hochdimensionalen Auswertungen schränken die Einsatzfähigkeit von materialisierten Sichten stark ein. Cache-Techniken können aufgrund kontinuierlicher Änderungen am Datenbestand ebenfalls nicht eingesetzt werden. Um Zugriffe auf die Rohdaten dennoch vermeiden zu können, wurde ein kombinierter Ansatz entwickelt. Die benötigten Aggregate werden a priori in einen Aggregat-Cache geladen. Um eine hohe Wiederverwendbarkeit und einfache Wartbarkeit zu ermöglichen, werden die Aggregate nach den Hauptdimensionen zerlegt abgelegt. Das Verfahren unterstützt auch die Berechnung nicht-additiver Kennzahlen. Es wurde basierend auf einem relationalen Datenbanksystem realisiert und ist produktiv im Einsatz.

## 1 Einleitung

Die GfK-Gruppe ist eines der weltweit führenden Marktforschungsunternehmen mit einem Gesamtumsatz von über 1,1 Milliarden €. Im Geschäftsfeld „Retail and Technology“ stellt die GfK ihren Kunden umfassende Marktberichte zu technischen Gebrauchsgütern auf internationaler Basis zur Verfügung. Der Kundenkreis der GfK Retail and Technology (GfK RT) umfasst vornehmlich international operierende Markenartikelhersteller und Handelshäuser.

Die Basis des Reportings bilden Verkaufs-, Preis- und Bestandsdaten aus etwa 350.000 Geschäften in über 70 Ländern. Diese Daten werden im Rahmen eines umfangreichen ETL-Prozesses konsolidiert. Am Ende des Datenproduktionsprozesses werden die berichtsfertig konsolidierten Daten in die Auswertungsdatenbank, die Reporting Base, geladen. Die Reporting Base ist ein ROLAP-basiertes Data Warehouse auf Basis eines Oracle Real Application Clusters. Ein detaillierterer Überblick über das Auswertungssystem der GfK RT wird in [BG09] gegeben.

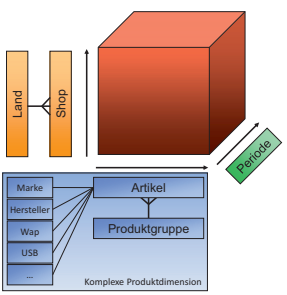



Abbildung 1: GfK RT Datenmodell

Der auswertungsbezogene Datenbestand ist nach einem Star-Schema organisiert. Das Datenmodell basiert auf 3 Hauptdimensionen, die Informationen zu Geschäften, Produkten und der Zeit enthalten (Ab-

bildung 1). Auf Basis dieser Dimensionen werden Fakten wie Verkaufsmenge, Preis und Bestand erhoben, aus denen verschiedene additive und nicht additive Kennzahlen (z.B. Umsätze, Marktanteile, Distributionen) berechnet werden können. Für die Auswertungen werden Geschäfte hierarchisch Ländern und Vertriebskanälen zugeordnet. Produkte sind nach Warengruppen, Kategorien und Sektoren organisiert. Eine Besonderheit ist die Vielzahl der dimensionalen Detailattribute, die insbesondere bei den Produkten für die Auswertung herangezogen werden. Abhängig von der Warengruppe werden bis zu 100 Warengruppen-spezifische Produktmerkmale erhoben. Neben der wichtigen Information über Marke/Hersteller können das bei Mobiltelefonen Eigenschaften wie WAP-Funktion, USB-Funktion und Kameraauflösung sein, bei Waschmaschinen hingegen Energieeffizienzklasse und Schleuderdrehzahl. Damit ergibt sich, dass der Auswertungsraum pro Warengruppe allein durch die Produktmerkmale potenziell 100-dimensional ist.

Aufgrund der Vorverdichtung auf Wochen- bzw. Monatsebene im Dateneingang ist die Größe des Data Warehouse mit knapp einem Terabyte noch überschaubar. Was die Datenverarbeitung in der GfK RT jedoch extrem anspruchsvoll macht, sind Masse und Komplexität der Berichte. Mehrere hundert Kunden erhalten nationale und internationale Standardberichte zu über 400 Warengruppen. Dafür werden jeden Monat über 100.000 Berichtsdateien und individualisierte Datenbanken produziert. Zusätzlich erhalten die Kunden einen Online-Zugang, der es ihnen ermöglicht, internationale Berichte online aus dem Data Warehouse System abzurufen.

Die Standardberichte zeichnen sich durch einen sehr hohen Detaillierungsgrad in Bezug auf die Produktmerkmale aus. Abbildung 2 zeigt einen typischen Seitenriss aus dem Bereich Mobiltelefone. Zu einem bestimmten marktrelevanten Aspekt der Warengruppe – in diesem Fall die eingebaute Kamera – sind hier die wesentlichen Merkmalsausprägungen dargestellt. Da verschiedene Merkmale in unterschiedlichen Kombinationen auftreten, sind in der Regel mehrere SQL-Anweisungen nötig, um einen Seitenriss zu berechnen.

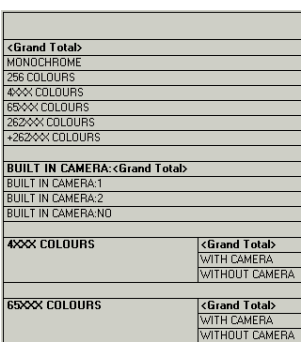



Abbildung 2: Beispiel eines Seitenrisses

Ein Standardbericht enthält zum Teil über 100 solcher Seitenrisse, die wiederum für bis zu 70 Länder pro Vertriebskanal ausgerechnet und meist zusätzlich noch nach Preisklassen unterteilt werden. Aufgrund der Kombinatorik kann die konventionelle Produktion eines Standardberichtes leicht über 100.000 SQL-Anweisungen umfassen, womit alle etablierten Systeme performancemäßig an ihre Grenzen stoßen. Online-Berichte werden im navigierenden Zugriff eher seitenweise abgerufen, so dass die Zahl der Anweisungen damit nicht so hoch ist. Die Antwortzeit-Anforderungen liegen dafür generell im Sekundenbereich, unabhängig vom Datenumfang. Sowohl die Batch-Produktion tausender Berichte als auch der Online-Zugang stellen somit enorme Herausforderungen in Bezug auf die Performance dar.

## 2 Performance-Optimierung durch redundante Datenstrukturen

In den vergangenen 10 Jahren wurde sehr viel zum Thema Performance-Optimierung in Data Warehouse Systemen veröffentlicht. Beschränkt man sich auf Methoden, die Performance durch Redundanz erzielen sollen, gibt es die Möglichkeit, entweder durch Vorberechnung von Aggregaten in Form materialisierter Sichten oder durch Cache-Techniken den Zugriff auf die Rohdaten zu vermeiden.

*Materialisierte Sichten* mit Anfragereformulierung haben den großen Vorteil, dass sie bereits in kommerziellen Datenbanksystemen wie Oracle verfügbar sind [AF06]. Eine besondere Stärke ist die universelle Wiederverwendbarkeit materialisierter Sichten. Die Auswahl geeigneter Aggregationsniveaus, aus denen viele Anfragen ableitbar sind, stellt jedoch ein Kernproblem dieses Ansatzes dar ([GL01, ACN00]). Aufgrund der Vielzahl der Auswertungsdimensionen in der GfK RT kommen pro Warengruppe bis zu $2^{100}$ verschiedene Aggregationsebenen in Betracht. Das würde entweder zu wenigen sehr feingranularen Materialisierungen oder einer großen Anzahl von Sichten gröberer Granularität zur Darstellung der häufigsten Merkmalskombinationen führen. De facto bedeutet das, dass in dieser Dimension nicht sinnvoll aggregiert werden kann. Nur die Geschäfte können auf Land- und Kanalebene voraggregiert werden, da deutlich weniger mit Geschäftsmerkmalen gearbeitet wird. Diese Art von Aggregationstabellen kann aber nur für die additiv berechenbaren Kennzahlen (Verkaufsmengen, Umsätze, Preise) eingesetzt werden. Die für die Marktforschung besonders wichtigen Distributionskennzahlen hingegen messen den Anteil der Geschäfte, die ein bestimmtes Produktsegment führen (z.B. Nokia-Telefone mit 3-Mega-Pixel-Kamera), so dass in keiner Dimension ein sinnvolles Aggregationsniveau für eine materialisierte Sicht gefunden werden kann. Da durch den Produktionsprozess in der GfK ständig neue Datenpakete pro Warengruppe, Land und Periode in die Reporting Base übertragen werden, würde die Verwendung von materialisierten Sichten oberhalb dieser Hierarchiestufen auch zu einem massiven Wartungsproblem führen.

*Data Caching* kann eingesetzt werden, um nahezu alle Arten von Anfragen zu beschleunigen. Allerdings müssen die nötigen Informationen erst einmal im Cache sein, und dann muss entscheidbar sein, unter welchen Voraussetzungen Anfragen auf Basis von Data Caches berechnet werden können. Typischerweise wird der Cache-Inhalt ähnlich einer materialisierten Sicht deskriptiv mit Hilfe einer definierenden Anfrage beschrieben [LGZ04]. Größter Vorteil des Cachings gegenüber materialisierten Sichten ist die dynamische Anpassung auf die Lastsituation [ABK+03]. Besonders problematisch ist hingegen die Entscheidung, wann der Cache invalidiert werden muss. Je höher das Aggregationsniveau des Anfrageergebnisses, desto häufiger muss es verworfen und wieder komplett neu berechnet werden. Der Result Cache in Oracle 11g verwirft den Cache-Inhalt beispielsweise bei jeder Änderung an einer der zugrundeliegenden Relationen, unabhängig davon, ob der Cache-Inhalt überhaupt betroffen ist.

Obwohl beide Techniken in verschiedensten Ausprägungen bereits Einzug in kommerzielle Datenbanksysteme gefunden haben, waren die damit erreichbaren Ergebnisse nicht ausreichend, um den Performance-Anforderungen des umfassenden und detailreichen Reportings in der GfK RT gerecht zu werden. Daraus ergab sich die Notwendigkeit, einen eigenen Ansatz zu finden, der die Stärken von vorberechneten Aggregaten mit Cache-

Techniken verbindet, ohne jedoch die Schwächen zu übernehmen. Erste Überlegungen in dieser Richtung wurden in [TAL06] veröffentlicht. Inzwischen wurde das Verfahren so weit entwickelt, dass es produktiv eingesetzt wird. Ziel der nachfolgenden Abschnitte ist es, einen Überblick über die Realisierung des Pre-Caching-Verfahrens und die damit erzielten Ergebnisse im praktischen Einsatz zu geben.

## 3 Pre-Caching anwendungsseitig vorberechneter Aggregate

In der GfK RT wurde mit der Aggregate Base ein Pre-Caching-System für die Realisierung folgender Design-Ziele entwickelt:

1. Die Aggregate müssen a priori bereit gestellt werden können. Ein rein dynamischer Ansatz hätte zur Folge, dass der erste Zugriff immer zu lange dauert. Da die Standardberichte periodisch mit den gleichen Segmentationen produziert werden und vorab bekannt sind, kann durch eine Vorab-Bereitstellung dieses Problem vermieden werden.
2. Es muss möglich sein, Aggregate dynamisch hinzuzufügen. Die Berichte ändern sich von Zeit zu Zeit, und es werden ad hoc neue Berichte erstellt. In diesem Fall muss das Verfahren fehlende Aggregate erkennen, nachberechnen und automatisch in den Cache aufnehmen.
3. Der Cache muss mit häufigen Änderungen umgehen können. In der GfK RT stellt die Kombination aus Land und Warengruppe eine Produktionseinheit dar, auf welcher pro Periode Daten in die Reporting Base eingespielt werden. Ein Aggregat, das vielleicht 50 Länder und 20 Warengruppen umfasst, würde damit in einer Berichtsperiode sehr häufig invalidiert.
4. Die Aggregate sollen in relationalen Strukturen abgelegt werden können, damit auf konventionelle Datenbanktechnologie mit ihren Vorteilen in Bezug auf Skalierbarkeit und Verfügbarkeit zurückgegriffen werden kann.

### 3.1 Das Aggregationsmodell

Um die Grundidee des Cache-Aufbaus und Zugriffs zu illustrieren, ist in Abbildung 3 oben ein einfacher Bericht dargestellt. Ausgegeben wird die Verkaufsmenge (*Sales Units*) der beiden Warengruppen *Mobilephones* und *Smartphones* in den Ländern Frankreich, England und Russland in der Periode März 2007. Neben der Gesamtsumme sind zusätzlich die Verkäufe nach Marken aufgeschlüsselt.

Um die Wiederverwendbarkeit und Wartbarkeit zu gewährleisten, werden die Aggregate nach den Hauptdimensionen Land, Produktgruppe und Periode zerlegt. Wie in Abbildung 3 unten angedeutet, wird beispielsweise der Gesamtwert für Nokia intern pro Land und Warengruppe berechnet und abgespeichert. Der Gesamtwert für den Bericht muss dann

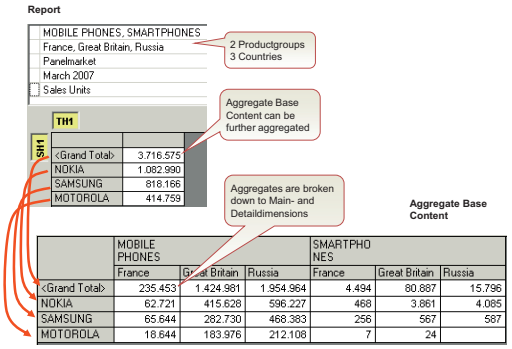



Abbildung 3: Berechnung von Kennzahlen mit Hilfe der Aggregate Base

nicht mehr aus Rohdaten, sondern nur noch aus 6 Aggregatzellen berechnet werden. Da in den Berichten beliebige Kombinationen von Produktmerkmalen gebildet werden können, müssen diese gesondert behandelt werden. Jede Aggregatzelle ist neben den Hauptdimensionen beschrieben durch ein Tupel von (Attribut:Wert)-Pärchen, z.B. (Marke:Nokia) oder (Marke:Nokia;UMTS:Ja;Kamera:Ja). Diese potenziell hochdimensionalen Merkmalskombinationen werden auf einen eindimensionalen Schlüssel abgebildet. Damit können die heterogen strukturierten Aggregate in einer einzigen relationalen Faktentabelle abgelegt werden.

Formal ausgedrückt wird in der Aggregate Base ein standardisierter Satz von Basiskennzahlen $F = (F_1, \ldots, F_f)$ auf Granularität von Haupt- und Detaildimensionsattributen abgelegt. Die Hauptdimensionen der Aggregate Base $M$ werden gebildet aus geeigneten Kategorienattributen eines relationalen Star- Schemas: $M = \{M_1, \ldots, M_m\}$. In der GfK ist $m = 3$ und die Kategorienattribute sind Land, Produktgruppe und Periode.

Die Detaildimensionen $D$ dienen der Darstellung semi-strukturierter Detailinformationen von beliebigen Kombinationen der Merkmale aus den Hauptdimensionen: $D = \{D_1, \ldots, D_d\}$ mit $d \leq m$. Über eine Abbildungsrelation $k$ wird jede mögliche Kombination von (Attribut:Wert)-Pärchen der Detaildimensionen auf genau einen numerischen Schlüssel abgebildet. $k$ ist also eine bijektive Funktion:

$$k\ \{(\text{Attribut:Wert})_1 \ldots (\text{Attribut:Wert})_n\} \rightarrow \mathbb{N}$$

Für die GfK gilt $d = 2$, da nur Merkmalskombinationen aus der Produkt- und Geschäftsdimension berücksichtigt werden (PKey, SKey). Für eine Datenzelle aus dem Beispiel (Abbildung 3) ergibt sich der folgende Eintrag in die Aggregate Base:

$$\underbrace{(\text{France}}_{\text{Land}}, \underbrace{\text{Mobilephones}}_{\text{Produktgruppe}}, \underbrace{\text{March 2007}}_{\text{Periode}}, \underbrace{337439}_{\text{PKey: k((Marke:Nokia))}}, \underbrace{337434}_{\text{SKey: k((Grand Total))}}, \underbrace{62721}_{\text{Sales Units}}, \ldots)$$

Das Aggregationsmodell der Aggregate Base reduziert somit die Dimensionalität eines $m + r$-dimensionalen Datenraumes auf $m + d$ Dimensionen mit $d << r$. [1] Durch diese

[1] $m$ Anzahl der Hauptdimensionen, $r$ Anzahl der Merkmalsattribute, $d$ Anzahl der Detaildimensionen

Art von Dimensionsreduktion wird die Modellierung durch ein kompaktes relationales Datenschema der Form $(M, D, F)$ ermöglicht.

### 3.2 Berechnung von Kennzahlen auf Basis der Aggregate Base

Erster Schritt beim Zugriff auf Kennzahlen der Aggregate Base ist die Übersetzung des Auswertungsraumes eines Berichtes auf die Hauptdimensionswerte des Datenraumes der Aggregate Base (Addressierung). Im Beispiel aus Abbildung 3 werden zunächst die Warengruppen (Mobilephones und Smartphones), die Länder (Frankreich, England und Russland) und die Periode (März 2007) aufgelöst. Im zweiten Schritt müssen die Detailinformationen, z.B. (Marke:Nokia) aus der Produktinformation und der *Grand Total* aus der Geschäftsdimension auf ihre korrespondierenden numerischen Werte übersetzt werden (*Key Mapping*).

Aggregate können sowohl direkt abgegriffen werden (Punktzugriff), z.B. die Verkäufe für Mobiltelefone der Marke Nokia in Frankreich im März 2007, als auch weiter verdichtet werden, wie beispielsweise das Länder- und Warengruppenkumulat für die Marke Nokia (Abbildung 3). Die weitere Verdichtung von hinterlegten Aggregaten findet unter Nutzung dimensionaler Hierarchien oder über frei definierbare Gruppen für die Attribute der Hauptdimensionen statt. Eine Aggregation über die Attribute der Detaildimensionen ist nicht möglich, da der Schlüsselwert $k$ auch die Werteausprägungen kapselt. Jede weitere Verdichtung der materialisierten Basiskennzahlen $F$ wird ausschließlich über die Aggregationsfunktion SUM realisiert, auch wenn die eigentliche Urberechnung der Basiskennzahlen nicht notwendigerweise additiv ist. Eine nicht-additive Metrik ist beispielsweise die *Distribution*. Die Distribution prozentuiert die Anzahl von Geschäften, die Produkte mit bestimmten Merkmalen führen, auf die Anzahl aller beobachteten Geschäfte. Diese beiden Basiskennzahlen werden in der Aggregate Base abgelegt. Auch wenn die Basiskennzahlen nicht additiv berechenbar sind, so können die Aggregatzellen der Aggregate Base zumindest über Länder hinweg summiert werden, da Geschäfte eindeutig einem Land zugeordnet werden und somit deren Anzahl überschneidungsfrei weiter verdichtet werden kann.

Mit Hilfe der Aggregate Base können Reports mit beliebig heterogenen Seitenrissen (unterschiedlichste Kombinationen von Produkt- und Geschäftsmerkmalen) auf Basis sehr weniger SQL-Anweisungen komplett berechnet werden. Im einfachsten Fall ( Berechnung der Verkaufsmenge) ist bei Zugriff auf die Aggregate Base lediglich eine SQL-Anweisung nötig, die Aggregate über die Hauptdimensionsattribute verdichtet und auf die PKey- und SKey-Werte, die durch den Seitenriss addressiert werden, einschränkt. Im Beispiel aus Abbildung 3 werden insgesamt 4 PKeys und 1 SKey (*Grand Total*) für die angefragten Hauptdimensionsattribute aus der Aggregate Base angefordert. Der gesamte Report wird auf Basis von

$$\underbrace{3}_{\text{Land}} \times \underbrace{2}_{\text{Produktgruppe}} \times \underbrace{1}_{\text{Periode}} \times \underbrace{4}_{\text{PKey}} \times \underbrace{1}_{\text{SKey}} = 24$$

Aggregaten der Aggregate Base berechnet.

Beim Zugriff auf die Aggregate Base wird ein optimistischer Ansatz mit „Vollständigkeitsvorbehalt“ realisiert, d.h. bei jedem Zugriff auf Aggregate wird davon ausgegangen, dass diese bereits vorhanden sind. Eine Kennzahl kann nur dann im Reporting weiter verwendet werden, wenn sie das Kriterium der Vollständigkeit erfüllt. Ein Aggregat ist genau dann vollständig aus der Aggregate Base berechenbar, wenn die Ergebniskardinalität gleich der Erwartungskardinalität ist. Für die Vollständigkeitsprüfung müssen also Ergebniskardinalität und Erwartungskardinalität ermittelt werden. Die Ergebniskardinalität wird über einen einfachen COUNT(*) direkt beim Zugriff auf die Aggregate Base ermittelt. Für die Bestimmung der Erwartungskardinalität wird eine Kontroll-Relation herangezogen. Diese ist eine *Factless Fact Table*, die nur aus den Hauptdimensionen $M = \{M_1, \ldots, M_m\}$ der Aggregate Base besteht. In dieser Relation wird hinterlegt, ob für eine Kombination aus (Land, Produktgruppe, Periode) Rohdaten geladen sind. Ist das der Fall, so muss auch ein entsprechendes Aggregat vorhanden sein. Die Kontroll-Relation ist extrem kompakt, so dass die Zugriffskosten für die Bestimmung der Erwartungskardinalität völlig vernachlässigbar sind.

In dem Bericht aus Abbildung 3 ist der Erwartungswert für jede Ergebniszelle 6 (3 Länder $\times$ 2 Warengruppen $\times$ 1 Periode). Würden die Aggregatzellen von Mobilephones/Frankreich aus irgendeinem Grund fehlen, wäre die Ergebniskardinalität 5 und der mit der Aggregate Base errechnete Wert unvollständig. In einem solchen Fall werden die entsprechenden Aggregate dynamisch nachgerechnet und in die Aggregate Base eingetragen.

### 3.3 Befüllung der Aggregate Base

Die Vorberechnung von Aggregaten für die Aggregate Base ist Bestandteil des Ladeprozesses für die Rohdaten. Dazu wird ein speziell definierter Versorgungsbericht verwendet, der die im Standard-Reporting verwendeten Seitenrisse enthält. Da die Daten periodisch auf Ebene von Land und Produktgruppe geladen werden, findet das Berechnen der Aggregate auf einem sehr überschaubaren Datenbestand statt. Tests haben erwiesen, dass bereits bei einmaliger Berechnung eines komplexen Berichtes das Pre-Caching lohnenswert ist. Die Zeit für die Berechnung der Aggregate und für die Auswertung des Berichtes ist insgesamt geringer, als wenn der Bericht vollständig auf Basis der Rohdaten berechnet wird. Die initiale Vorberechnung amortisiert sich insbesondere dann, wenn Aggregate sehr häufig wiederverwendet werden können.

### 3.4 NULL-Wert-Kompression bei Verwendung mehrerer Detaildimensionen

Die Berechnung der Erwartungskardinalität für die Vollständigkeitsprüfung auf Grundlage der Hauptdimensionsattribute erfordert zusätzlich die Speicherung von NULL-Aggregaten. NULL-Aggregate entstehen bei der Befüllung der Aggregate Base immer dann, wenn für eine bestimmte Merkmalskombination keine Rohdatensätze existieren. Das trifft beispielsweise für Marken zu, die in einem Land nicht verkauft werden. In der Aggregate Base

werden diese NULL-Werte permanent abgelegt. Somit ist sichergestellt, dass die Ergebniskardinalität bei Vollständigkeit auch tatsächlich korrekt ermittelt werden kann.

Das direkte Ablegen von NULL-Aggregaten ist bei einer Detaildimension noch akzeptabel, im mehrdimensionalen Fall führt dieses Vorgehen jedoch zu einem nicht tolerierbaren Anwachsen des Datenvolumens. Bei der GfK RT ist die Detaildimension zweidimensional und besteht aus PKey und SKey. Typische Bestandteile des SKeys sind Geschäftsmerkmale wie der Vertriebskanal oder die Umsatzgrößenklasse. Da in den Standardberichten alle Produktmerkmalskombinationen mit allen Geschäftsmerkmalskombinationen ausmultipliziert werden, ist der errechnete Datenraum beträchtlich. Untersuchungen haben gezeigt, dass über 90% der Aggregatzellen NULL-Werte enthalten.

Um nicht für jede mögliche Kombination von PKey- und SKey-Werten ein NULL-Aggregat speichern zu müssen, werden sämtliche Werte der Detaildimensionen zur NULL-Wert-Speicherung linearisiert (Abbildung 4). Statt des kompletten Datenraumes werden also in einer gesonderten Relation nur zwei Listen für die Detaildimensionen abgelegt, aus denen sich der leere Datenraum wieder aufspannen lässt. Dieser Raum leerer Aggregatzellen wird dann mit den Aggregatzellen, die tatsächlich Zahlenwerte beinhalten, überlagert und zusammenaggregiert. Die Rekonstruktion ist zwar nicht kostenlos, aber deutlich günstiger als der Zugriff auf unkomprimierte NULL-Aggregate.

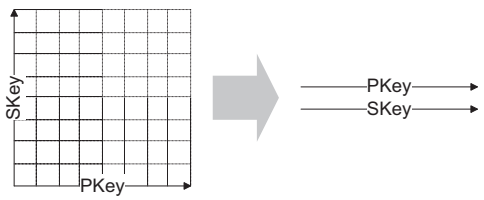
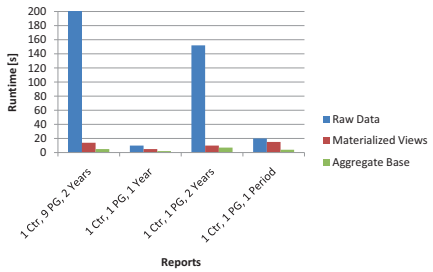



Abbildung 4: Zweidimensionale NULL-Wert-Kompression

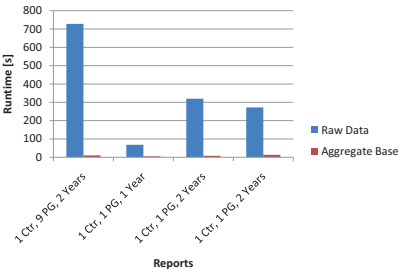
Die Kompressionsrate ist umso größer, je mehr Aggregatzellen auf einmal ausgerechnet werden. Werden 1000 PKey-Werte mit 50 SKey-Werten kombiniert, so werden für die Information, dass diese $1000 \times 50$ Zellen berechnet wurden, lediglich 1050 NULL-Werte abgelegt. Werden nur 2 PKey-Werte und 2 SKey-Werte kombiniert, sind für diese 4 Zellen auch in der linearisierten Darstellung 4 NULL-Werte nötig. Zusätzlich müssen noch die Zellen mit Zahlenwerten abgelegt werden. In diesem Fall ist die Linearisierung also sogar schädlich. Die NULL-Wert-Komprimierung ist deswegen besonders profitabel bei der initialen Befüllung der Aggregate Base durch einen umfangreichen Versorgungsbericht. In diesem Fall werden leicht Kompressionsraten von über 90% erreicht. Für dynamische Nachberechnungen wird auf die NULL-Wert-Kompression hingegen verzichtet.

### 3.5 Evaluation

Ein wesentliches Ziel bei der Entwicklung der Aggregate Base war es, eine Performancesteigerung für das interaktive Reporting zu erreichen. Abbildung 5 a) stellt die Laufzeiten unterschiedlich komplexer Berichte bei der Berechnung von Verkaufswerten gegenüber. Für den Vergleich mit materialisierten Sichten wurde eine Aggregationstabelle angelegt, welche die Geschäftsdimension auf Länder-Ebene beschränkt auf wenige häu-




a) Laufzeit für Sales Units




b) Laufzeit für Distribution

Abbildung 5: Laufzeiten unterschiedlicher Berichte auf Basis von Rohdaten, materialisierter Sichten und Aggregate Base

fige Geschäftsmerkmale hoch aggregiert vorhält, auf der Produktebene aber auf Basisgranularität bleibt, um alle Merkmalskombinationen in den Berichten abdecken zu können. Während die Laufzeiten für die Urberechnung von Kennzahlen aus Rohdaten für den interaktiven Betrieb nicht akzeptabel sind, kann bei Verwendung von vorverdichteten Daten, insbesondere bei sehr umfangreichen Fusionen aus Ländern und/oder Warengruppen und längeren Zeiträumen, eine erhebliche Performancesteigerung erzielt werden. Die Berechnung aus der Aggregate Base ist meist um Faktor 2-3 schneller als die Berechnung basierend auf materialisierten Sichten. Dieser Performancevorteil resultiert aus der Tatsache, dass im Gegensatz zu materialisierten Sichten auch für einen komplexen heterogen strukturierten Bericht meist nur eine einzige SQL-Anweisung für die Berechnung aus der Aggregate Base notwendig ist. Hinzu kommt, dass nur eine begrenzte Anzahl von Anfragen tatsächlich durch materialisierte Sichten unterstützt werden kann. Ursache dafür ist das starre relationale Schema materialisierter Sichten, welches deren universelle Nutzbarkeit einschränkt. Je komplexer die Urberechnung von Kennzahlen ist, desto höher ist der Nutzen der Aggregate Base. Die Berechnung von Distributionen ist ein solches Beispiel (Abbildung 5 b)). Die Distributionsberechnung ist zum einen besonders aufwändig und zum anderen nicht durch Nutzung materialisierter Sichten optimierbar. Im Vergleich zur Urberechnung ist bei gefüllter Aggregate Base eine Beschleunigung um das bis zu 100-fache erreichbar.

## 4 Zusammenfassung

Für die speziellen Anforderungen in der GfK Retail and Technology wurde ein neuartiges Pre-Caching-Verfahren entwickelt. Vorab bekannte Aggregate für die Produktion von Standardberichten werden mit der Bereitstellung der Rohdaten automatisch berechnet und in die Aggregate Base eingefügt, damit bereits der erste Zugriff schnell erfolgen kann. Fehlende Aggregate werden zur Berichtslaufzeit durch eine automatische Vollständig-

keitsprüfung erkannt und können dynamisch ergänzt werden.

Die Aggregate werden in relationalen Strukturen in einer Oracle-Datenbank abgelegt. Die Verwendung relationaler Strukturen für die Speicherung von Aggregaten, die für beliebige Merkmalskombinationen aus der Produkt- und Geschäftsdimension berechnet werden, setzt eine Dimensionsreduktion des Datenraumes voraus. Dadurch wird die Speicherung und der Zugriff auf Aggregate beliebiger Granularität in den Detaildimensionen ermöglicht. Der Zugriff erfolgt anwendungsseitig über einen in Eigenentwicklung entstandenen Analyse-Server, der sowohl vom Online-Frontend angesprochen werden kann als auch die Batch-Produktion von Berichten ermöglicht.

Das System ist produktiv im Einsatz. Die erreichten Laufzeitverbesserungen reichen je nach Berichtsumfang und Komplexität von Faktor 2 bis zu einigen Größenordnungen.

## Literatur


[ABK+03] Mehmet Altinel, Christof Bornhövd, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh und Berthold Reinwald. Cache Tables: Paving the Way for an Adaptive Database Cache. In *Proceedings of 29th International Conference on Very Large Data Bases, (VLDB 2003, September 9-12, Berlin, Germany)*, Seiten 718–729, 2003.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri und Vivek R. Narasayya. Automated Selection of Materialized Views and Indexes in SQL Databases. In *Proceedings of 26th International Conference on Very Large Data Bases, (VLDB 2000, September 10-14, Cairo, Egypt)*, Seiten 496–505, 2000.

[AF06] Jens Albrecht und Marc Fiedler. Datenbank-Tuning – einige Aspekte am Beispiel von Oracle 10g. *Datenbank-Spektrum*, 16:26–33, 2006.

[BG09] Andreas Bauer und Holger Günzel, Hrsg. *Data-Warehouse-Systeme Architektur, Entwicklung, Anwendung*. dpunkt.verlag, 3. Auflage, 2009.

[GL01] Jonathan Goldstein und Per-Åke Larson. Optimizing Queries Using Materialized Views: A Practical, Scalable Solution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, (SIGMOD 2001, May 21-24, Santa Barbara, USA)*, Seiten 331 – 342, 2001.

[LGZ04] Per-Åke Larson, Jonathan Goldstein und Jingren Zhou. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In *Proceedings of the 20th International Conference on Data Engineering, (ICDE 2004, March 30 - April 2, Boston, USA)*, Seiten 177–189, 2004.

[TAL06] Maik Thiele, Jens Albrecht und Wolfgang Lehner. Optimistic Coarse-Grained Cache Semantics for Data Marts. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management, (SSDBM 2006, July 3-5, Vienna, Austria)*, Seiten 311–320, 2006.

# Index Challenges in Native XML Database Systems

Henrik Loeser[1], Matthias Nicola[2], Jana Fitzgerald[2],

1: IBM Deutschland Research & Development
Schönaicher Str. 220, D-71032 Böblingen

2: IBM Silicon Valley Lab.
555 Bailey Ave, San Jose, CA 95123, USA
Email: hloeser@de.ibm.com, {mnicola, jfitzge}@us.ibm.com

**Abstract.** Today, more and more enterprises process XML data, many of them already in XML database systems. Once systems start to grow in size, scalability becomes an issue. One of the core operations during insert processing is the index maintenance. Typical relational systems only have few indexes per table, however, XML database users already look at creating hundreds of XML indexes per XML column. A similar situation is present during query processing when many indexes are present which the system can choose from and has to combine them for optimal performance. This creates new challenges for the database system – unknown in the traditional relational world. In this paper, we discuss those index challenges and possible solutions.

**Keywords.** XML, Indexing, Database Systems, Scalability, Performance

## 1 Introduction

Today, more and more enterprises process XML data. That data comes from different sources and is often managed by separate applications and stored in separate databases. There are also scenarios where XML data with a very high degree of variability is stored in a single database or even a single XML column. This creates challenges for native XML database systems. In this paper, we focus on the challenges related to maintaining XML indexes in insert-intensive applications.

The majority of existing work on XML indexes focuses on query performance and not on insert performance. In fact, the majority of research papers that propose XML index structures do not evaluate the cost of building and maintaining those indexes. They typically use a query-only workload to evaluate the query performance only. However, real-world XML applications often consist of a mix of read and write operations, and even a moderate amount of insert, update and delete operations can adversely affect overall system performance if the index structures are heavy and require costly maintenance.

In section 2, we first describe typical XML usage patterns in real-world applications, characteristics of popular industry standard XML formats, and the resulting index requirements. Then we describe a specific application scenario – financial application logging – and the index challenges that it creates. In section 3 we explain how XML data can be indexed in DB2 and how optimized index processing can address the challenges posed by the application logging scenario. Section 4 presents a performance evaluation of different approaches for maintaining XML indexes. We conclude the paper with a summary in section 5.

## 2 XML Index Requirements in Real-World Application Scenarios

In this section we discuss common characteristics of XML applications and how these characteristics frequently require the use of a large number of indexes.

### 2.1 XML Representation of Business Objects

XML is commonly used as a format for business objects, or messages that represent or transport these business objects. Common examples include purchase orders, trades, customer records, sales records, patient records, claim forms, and other business artifacts. In many cases these business objects can be very diverse and variable and it requires a large number of attributes to model them. A classic example is customer demographics such as age, geographical location, marital status, ethnicity, income range, profession, favorite newspaper, favorite sport, holiday preferences, hobbies, current car, and many other attributes. Another example is product characteristics such as color, size, weight, length, volume, material, water resistance, power requirement, resolution, operating temperature, adapters, display type, battery capacity, and many more. A business with a diverse set of products can easily require hundreds or several thousands of attributes to model their product portfolio. However, any particular product typically only has a subset of these attributes, often between 10 and 100. Such data is also known as having *sparsely populated attributes*.

Examples of such diverse and feature-rich products include electronics, telecommunication services, and financial investment products. It's easy to see that a similar richness in attributes exists in many other industries as well, such as patient records in health care or tax forms in government.

Such rich application scenarios and business requirements have existed long before the emergence of XML and are not fundamentally new. In fact, they have frequently posed challenges in relational database modeling. One such problem is that the number of attributes might exceed the maximum row length or the maximum number of columns in a relational table. Sometimes this can be addressed by splitting the attributes over multiple tables and multiple rows. However, this often leads to a less intuitive database design and additional un-natural normalization. This in turn makes

inserting and querying the business data more challenging for application developers or business reporting analysts.

Another problem with large numbers of attributes is that the set of attributes typically changes over time. This is the case whenever products, services, or business processes change, i.e. all the time. In many companies, modifications to relational schemas, such as adding or dropping columns in a table, are strictly regulated and can require lengthy approval processes. It is not uncommon that this slows down the introduction of new products or services. It can also delay or even prevent the availability of new insights from enriched business reports.

A potential but undesirable "solution" for dealing with evolving and large sets of attributes is the use of name-value pairs. This approach is often implemented as a relational table with three columns (`id`, `name`, `value`). In this design the attributes are not expressed as column *names*, but as *values* in the column "`name`". This approach has significant inherent drawbacks, including:

- It's very difficult and often impossible to define business rules and constraints for Name/Value Pairs. This is because the meaning of the entries in the column "`value`" changes from row to row.
- For the same reasons, the column statistics that usually aid relational optimizers in the selection of good access plans suddenly have a different meaning. Hence optimizing complex queries for name-value pair data is often challenging.
- Business objects are now represented in a form that is no longer understood by business experts. Hence, writing queries against name-value pair data is complex and often requires many self-joins. Reporting queries with many predicates prove to be particularly difficult.
- Name-value pair tables handle all data as strings (text). Since the `value` column can contain arbitrary data values, it cannot be typed as integer, decimal, date, or timestamp. This also means that any indexes and comparisons treat the data values as strings. If you search for cars with a price greater than "5000", you will also find cars with prices such as "600" or "900", because these strings are greater than the string "5000". You can solve this with appropriate cast operations in your queries, but those often preclude the use of indexes.

XML allows for a very natural solution of these problems and for a much more intuitive representation of business objects. A key feature of XML is that XML elements and attributes can be optional. If certain properties do not apply to a particular business object, the corresponding elements and attributes are simply absent, and not even represented by NULL values. This means that there does not need to be a predefined and reserved storage location for every attribute for every business object. This allows for a more compact representation and intuitive representation of business objects that is also easier to exchange between applications or organizations.

For these reasons, XML is being adopted in practically every industry to model business objects that have large and evolving sets of sparsely populated attributes.

Companies in every industry have launched consortia to define and standardize XML Schemas for the representation and exchange of data in their industry [3].

For example, the world's leading financial companies have developed more than a dozen XML vocabularies to standardize their industry's data processing [8]. FpML (Financial Products Markup Language), FIXML (Financial Information Exchange), SwiftML, IFX, MISMO (Mortgage Industry Standards Maintenance Organization), OFX, and XBRL (Extensible Business Reporting Language) are among the most popular. FIXML is an industry-standard XML Schema for trade-related messages such as trade capture reports, buy/sell orders, and many others [7]. The FIX protocol is used by more than 150 leading financial companies worldwide. The XML version, FIXML, has been developed to improve extensibility, application layer independence, message validation, and robustness. FIXML also enables straight-through processing, which reduces operating costs and improves the quality and timeliness of information [2].

The FIXML schema defines a very large number of optional elements and attributes. It consists of 41 XML Schema documents and contains 1310 type definitions, 619 element declarations, and 2593 attribute declarations. The vast majority of those are optional and only a small subset of them are present in any given instance document. Table  shows that other industry standard XML Schemas define similarly large numbers of types, elements, and attribute.

| | Version | Types | Elements | Attributes | XSD Files |
|---|---|---|---|---|---|
| ACORD | XMLife_2.16.01 | 1369 | 9378 | 1275 | 4 |
| ARTS | 1.0 - 3.0 | 4825 | 6305 | 2011 | 32 |
| CDISC | 00-9-03 | 98 | 84 | 71 | 1 |
| FpML | 4.2 | 686 | 1867 | 196 | 23 |
| FIXML | 4.4 | 1310 | 619 | 2593 | 41 |
| HL7CDA | 3 | 1953 | 945 | 477 | 6 |
| IRS1120 | 2006v3.3 | 3415 | 11591 | 2632 | 600 |
| MISMO | 2.3 - 2.4 | 2899 | 1087 | 13733 | 31 |
| MCJE/NIEM | 1 | 415 | 936 | 46 | 7 |
| OTA | 2003/5 | 27293 | 24893 | 43141 | 234 |
| STAR | 5.0.4 | 5846 | 77319 | 625 | 192 |
| TWIST | 3.1 | 1016 | 2314 | 20 | 19 |
| UBL | 2 | 682 | 2665 | 253 | 43 |
| UNIFI | 1.01 - 2.01 | 5082 | 9747 | 127 | 71 |
| XBRL | 10/25/2006 | 1858 | 2847 | 383 | 45 |

**Table 1: Characteristics of Selected Industry Standard XML Schemas**

Even if many of the defined elements and attributes are used rarely, when they occur they do carry significant business information. To find and exploit this business information and to gain valuable insight from it, it is a common requirement to define

XML path indexes for a large number of these elements and attributes. In specific applications we have seen requirements to index anywhere from a dozen to a few hundred selected elements and attributes. At the same time, simply indexing *all* elements and attributes is an undesirable approach due to its significant overhead for insert, update, and delete operations.

## 2.2 A Concrete Scenario: Financial Application Logging

We have recently worked with several companies in the finance and insurance industry who have brought up requirements for *financial application logging*. Their requirements were remarkably similar and describe a recurring usage scenario for XML databases. In this section we describe the concrete requirements of one specific company. To protect their identity we simply call them *XYZ Bank* in this paper.

**Overview**
The internet banking system at XYZ Bank is required to log every event in any of their internet banking applications. Events include clicks that take a user to a new web page or dialog, entry of user data, as well as every click that initiates a banking transaction. This "logging" happens across a set of diverse applications such as checking accounts, loans, investment management, and others. Currently, these applications write "log entries" into a relational database table. This table contains columns for the application id (INTEGER) and a timestamp (TIMESTAMP) as well as a VARCHAR column that contains the actual log entries in a proprietary string format. This string format contains a concatenation of dozens of individual values. This format was chosen because the information captured in the log entries can vary widely from one event to the next. It's the number of different types of events as well as the diversity of the banking applications which lead to hundreds of possible attributes that can potentially occur in an event. However, any particular instance of an event only carries a few dozen attributes.

The purpose of the application logging includes the support of troubleshooting and problem resolution as well as auditing and compliance regulations for certain applications. Each log entry (event record) contains a fixed set of "header" fields, such as user ID, application ID, session ID, date, and timestamp. These fields exist for every log entry and can easily be stored in fixed relational columns. However, the body of each log entry is highly variable and application dependent, and cannot be mapped to a reasonable relational database schema. Additionally, different application owners require autonomy and flexibility in deciding what information to include in the log records. They need to be able to change existing applications or introduce new applications at any time without causing schema changes in the logging infrastructure.

**The Problem**
The key problem is that the variable part of the log entries (VARCHAR) is hard to query with adequate performance. Current relational database technology does not allow easy indexing of individual pieces of strings in a VARCHAR column. The applications that read this data typically use SQL "LIKE" predicates on these

VARCHAR values. This results in limited queryability, limited index usage, and sub-optimal performance.

**The Desired Solution**
The IT department at XYZ Bank has decided to use XML as the new data format for the event records. The benefits of XML for this scenario include that XML tags allow proper labeling of all data fields in a log entry, easy extensibility whenever applications add or change certain fields in the event records, and the ability to index and query individual elements and attributes in the log records.

**The Challenge**
The application logging workload is very insert heavy. There are approximately 10M to 20M inserts in a 24-hour day, with peak insert rates of 500 events per second. The XML documents that represent the events are 4kb to 20kb in size. The documents have a "header" with identical structure for every document, and a "body" which is highly variable. Due to the variability, XYZ Bank requires a very large number of XML indexes (at least 100). Many of these indexes contain entries for only a few percent of the rows in the table, some indexes less than 1%. Hence, although the number of indexes is large, many indexes are quite small compared to the data. This is because unlike relational indexes, XML indexes in DB2 contain zero, one, or multiple index entries for each document (row), depending on how often the indexed path exists in a particular document. Nevertheless, due to the large number of indexes, the cost of maintaining these indexes during the high insert rate was a key concern for XYZ Bank.

There will be no offline maintenance window for this system since the logging of the internet application has to run 24 x 7. Applications that read the data for troubleshooting and auditing purposes have to support at least 100 users who expect query response times of 1 sec or less. However, the query rates are significantly lower than the insert rates.

In the following we describe how the insert and indexing requirements of this scenario can be fulfilled.

## 3 XML Insert Processing and Index Maintenance

In this section we review the XML index support in DB2 for Linux, UNIX and Windows and discuss different strategies for maintaining XML indexes efficiently.

### 3.1 XML Indexes in DB2

Native XML databases and XML-enabled databases have employed different approaches towards indexing XML data, often as a consequence of the different XML

storage approaches[1]. Some vendors have chosen to use a "shred and index all" approach where the XML data (stored in a large object in various forms) is first mapped into relational side tables and then either partially or fully indexed in these tables [6]. This can potentially lead to problems caused by the additional processing effort, database logging overhead, and additional space consumption. Hence we believe that a "shred and index all" approach is not an efficient solution to the performance and indexing requirements of the application logging scenario and would not scale to the required database size.

In DB2 a different approach to XML indexing was chosen. Users can index selected elements and attributes and avoid the additional overhead for indexing data items that don't need to be indexed. In DB2, an XML index is created by specifying an XPath-like expression called `XMLPATTERN` that identifies the nodes to be indexed. The specified pattern expression can contain namespace declarations similar to XQuery prolog. It can also contain wildcards and the descendant-or-self axis.

Figure 1 shows how an index on the `applicationID` element within an `event`'s `header` is created. The values are represented as numbers because the data type `DOUBLE` is chosen for the index. DB2 allows for the creation of XML indexes with different data types [5]. DB2 does not attempt to derive the data type from an XML Schema because an application may not have an XML Schema or may store documents for different XML Schemas in the same column.

```
CREATE INDEX appIDidx ON appLogs (log)
GENERATE KEY USING XMLPATTERN
'/event/header/applicationID' AS SQL DOUBLE
```

**Figure 1: Example of an XML Index Definition in DB2**

When a `CREATE INDEX` statement is issued, the syntax is checked, including the `XMLPATTERN` syntax. Then the index metadata is stored in the system catalog and the physical index structures, including metadata on their own *for runtime use*, are created on disk in the index object. It is noted that every index stands by itself and no combined metadata for all indexes on a table or column is available. This creates additional challenges as we will see later.

[1] It is also worth noting that not all of the so-called "native XML storage" techniques would support the financial application logging scenario. The reason is that XML schema flexibility is required, i.e., the ability to store any well-formed XML document without knowing its schema in advance. Systems that use optimized shredding of XML into (object-) relational structures either need the XML schema to generate more efficient mappings or have to employ a general and less efficient mapping approach.

## 3.2 Differences to Relational Index Maintenance

In relational database systems it is a possible technique to chain the existing indexes in a linked list per table. The reason is that during insert, delete and possibly update operations every single index that is defined on a table needs to be maintained. For every index, one key needs to be processed for every row. Thus, walking over a chain of indexes is a straight-forward way of processing the index maintenance for all affected indexes.

The situation is quite different with XML indexes in DB2. If the path defined by the `XMLPATTERN` does not exist in a given document, no key for that specific index is produced. Hence, an insert or delete of the document does not affect this index. While the insert or delete of a relational requires *all* relational indexes for the table to be updated, the insert or delete of an XML document may affect only a subset of the defined indexes. This is a significant difference. If a very large number of indexes is defined, as in the application logging scenario described earlier, it is critical for performance to efficiently identify the subset of indexes that need to be updated when a new document of unknown structure is being inserted.

If the `XMLPATTERN` defined by the index exists exactly once in a given document, the index will have one index entry for that document. If a document contains multiple occurrences of the indexed path, then multiple keys are processed for a single index as a result of an insert or delete operation. Update operations may or may not affect the defined XML indexes, depending on whether the indexed paths are modified or not.

What is even more different for maintaining XML indexes is how the index keys are produced. For relational data the entire row is present at once. Additionally, a relational row has a given size and a fixed number of fields and fits into a data page. Thus, keys can be easily extracted from the row and one index key per index can be generated. XML documents are variable in nature and in the absence of a schema their structure is not known until parsed. XML documents are parsed and the resulting data is often available token by token, or in smaller or bigger chunks. Additionally, XML documents can be large and can span many data pages. A naïve implementation might first receive and insert the entire document, and then navigate the document to extract all required index keys. However the two-step process must be avoided for performance reasons. Instead, index keys need to be obtained concurrently with the data processing. More precisely, index keys need to be generated while the XML document is being parsed and formatted into the data pages. This process needs to work in a streaming fashion, so that index entries and data pages are produced for the "beginning" of a large document while the "end" of the document has not been reached and the full structure and size of the document is still partially unknown.

In order to find the relevant parts of the document, the different `XLMPATTERN`s are applied to the document. For relational data the indexed columns are clearly identified as part of the index definition. But, the definition of XPath-based indexes over XML data can contain * and // in the `XMLPATTERN`. This means that the exact location of

the index key values is not always known in advance. For example, while the index definition in Figure 1 uses a fully-specified path, a database administrator could have used any of the XMLPATTERNs shown in Figure 2. And Figure 2 is not even an exhaustive list of all possible XMLPATTERNs that include the applicationID element.

```
//applicationID
/event/*/applicationID
/event//applicationID
//header/applicationID
//*/applicationID
//header//applicationID
```

**Figure 2: Other Examples of XMLPATTERNs for an XML Index in DB2**

Maintaining indexes that are defined with wildcards and descendent-or-self axis require finding and extracting relevant index key data from the XML documents. This adds to the complexity of index maintenance and can have impact on the scalability and performance.

## 3.3 XML Index Maintenance Strategies

In this section, we first discuss general requirements for efficient XML index maintenance and then describe two possible approaches. Subsequently we look into techniques for improving one of the described approaches in order to meet very high performance and index requirements such as in the application logging scenario.

**Chaining vs. Trees**

For the decision about how to best manage indexes over XML data, some key requirements as well as overall system architecture need to be taken into account. For the design, the following general principles apply:

- The design and code should be kept simple, so that it can be maintained and is less prone to bugs.
- The memory consumption of the data structure needs to be carefully considered as databases can have thousands of tables and indexes, as well as hundreds or thousands of concurrent insert, update and delete transactions.
- As index maintenance is critical to the insert, update and delete performance, it needs to perform and scale well.

In addition, a balance between setup and execution costs needs to be kept as we will explain in the following. Every index is an object of its own, including some metadata, and only at runtime of an insert, update or delete operation the set of participating indexes can be determined. It is only at this time that combined metadata for the participating indexes can be computed, e.g., to optimize the actual index

maintenance[2]. We can therefore distinguish between a runtime preparation/setup phase for index maintenance and its actual execution. Because of the large variety of XML documents and formats that DB2 strives to supporting, it is not possible to assume that either only very large or only very small documents are processed. However, we do observe that applications that process large numbers of small documents (less than 50kb) are a lot more common than applications that process large documents (multiple or even hundreds of MB). Additionally, many applications require high insertion rates. Therefore, it makes sense to optimize for small to medium-sized documents. This means that setup costs for performing the index maintenance for a single document should be kept small as the runtime part will be small as well.

Critical for the performance of XML index maintenance is to efficiently identify the subset of indexes that are affected by the insertion of a new document. This requires matching of the `XMLPATTERN`s in the index definitions against the nodes of an incoming XML document. When considering data structures for this process, trees come to mind. The paths from all `XMLPATTERN`s of all indexes that are defined on the XML column could make up the branches in a pattern tree with some nodes having attached actions (generate key). The actions could be both on leaf and non-leaf nodes as it is possible to index both `/event/header` (atomize the entire subtree) and `/event/header/applicationID`. It is possible to navigate this pattern tree in parallel with and based on the current context of the inserted document. To give an example, with the insertion of a root element `event`, the context pointer would move to the node `event` in the tree. When a child element `header` and later `applicationID` is seen, we first move to `header`, then to `applicationID` in the tree. The attached action to `applicationID` would direct us to generate an index key for the index in Figure 1.
Such a tree can provide very good runtime performance because the current context of the incoming document immediately determines the indexes to be maintained. In contrast to it, the setup costs to create such a tree are high and the integration of patterns with wildcards would make it complex to handle. For each step of every pattern some processing is needed to construct the tree because the tree is assembled from all patterns which need to be inserted into the tree, step by step resulting in the tree's nodes. Wildcards would either need to be expanded with all possible combinations or handled separately.

Another approach is similar to the handling of relational indexes, i.e. to manage index information on a dedicated chain, i.e., a linked list. For every node of the incoming document, the chain would be traversed and every index pattern compared to the current node. Because indexes are already chained in DB2, the setup cost is minimal. The downside is that the run time cost is high and on the order of $n \times m$, where $n$ is

[2] We distinguish between index metadata in the catalogs – seen by the user – and that metadata stored in the index object's header (b-tree) itself. For performance reason only the metadata from the index object, not from the catalog is used which might not be obvious.

the number of indexes and m the number of nodes in the incoming document. In the following we discuss techniques to reduce this run time cost.

**Starting from the Leaves**

To determine whether an index key needs to be generated we have to check whether the XMLPATTERN matches the current node and its path. This could be done in an ongoing fashion top-down, similar to the pattern tree, or bottom-up for the currently processed node. In the bottom-up approach, we first check the last step in the pattern against the current node's name. If it matches, the node's parent is checked against the parent step in the pattern and so forth till the root is reached and we have a match.

Since DB2 internally uses 32 bit integer values (stringIDs) to encode node names, the comparison of the names is fast. Because the most common case is a non-match of the current node, the pattern checking usually is only a single integer comparison per index and node (as opposed to comparing entire paths). However, considering the potentially large number of nodes per document and the required number of indexes, the effort per document is still high.

**Late In, Early Out – Aggregated Path Information**

When looking at applications and their XML documents we observe that usually only leaf nodes are indexed because they are the nodes where relevant business data is located in a document tree. This observation can be used to speed up index maintenance because for any given node in the document the system only needs to check for index matches if there are any XMLPATTERNs of the same depth. During the index definition at DDL time the database system can analyze the XPath regarding its minimum and maximum level and for the presence of wildcards.
For the path /event/header/applicationID both the minimum and maximum path level is 3 because it is an absolute path. For the path /event//applicationID the minimum level is 2 and the maximum is the largest supported document depth, which is 125 in DB2.

At runtime, when the information for all present indexes is available and the environment for index maintenance is being initialized, the aggregated minimum and maximum path level across all participating indexes can be computed. Later, during the actual insert processing when a document is parsed, we only need to check for matching indexes if the current node of the document is within the minimum and maximum level of the defined indexes.

Given that many XML documents fan out over the first few levels and that wildcards are rarely used in indexes –at least for first steps in a pattern– we can avoid index processing for those levels with our "Late In, Early Out" strategy. We start as late as possible (minimum) and try to get out of path matching as early as possible (maximum).

# 4 Performance Evaluation

If only few indexes are present – typically up to 20 in environments with only one document type per XML column – then the above described strategy based on index chaining with some of the discussed performance enhancements works fine and has a good balance between setup costs and runtime costs. However, it will not scale well for the extreme use case with possibly hundreds of indexes.

**Test Scenario:**
Since the actual XML data of the financial application logging scenario was not available to us in sufficient quantities, we designed a test scenario that exhibits similar characteristics. This test scenario is based on the open-source benchmark TPoX [9] which simulates a financial online brokerage scenario. It exercises XML inserts and XML index maintenance (among other things). The TPoX data set includes data compliant with FIXML, the XML implementation of the Financial Information eXchange (FIX) standard which contains hundreds of optional attributes and elements. These characteristics as well as document sizes ranging between 1KB and 20 KB in size make TPoX a suitable substitute to mimic the insert and index requirements that we found at XYZ Bank.

For our tests we defined different numbers of indexes (60, 110, 160, and 210), such that for any given document there are always 10 indexes that match nodes in the document, plus 50, 100, 150, or 200 indexes that do not match any nodes in the document. The number of non-matching indexes is the variable aspect in our experiments.

Each of the matching indexes requires an update after each document insert, whereas any non-matching index does not need to be updated during a document insert. Using the multi-user TPoX insert workload, we executed tests in which 100 concurrent users inserted 7000 documents each, i.e. 700,000 documents total. The TPoX workload driver captures the throughput in inserts per minute as well as other statistics [4].

**Test equipment and configuration**
Tests were run on the following hardware:

- **Processor:** 4 dual-core 1.9 GHz POWER5 processors
- **Memory:** 32 GB
- **Operating System:** AIX v5.3
- **Storage:** 32 external disks spread over two DS4700 disk array subsystems with a capacity of 72 GB per disk

## 4.1 Simple Index Chaining

In a first set of tests we measured the performance of the simple chaining approach (described in the beginning of section 3.3) with the “Late in, Early out” enhancement applied. The results for different numbers of indexes are shown in Figure 3.

The vertical bars in Figure 3 show the throughput in inserts/min for different numbers of non-matching indexes. Not surprisingly, the highest insert throughput of 96,463 inserts per minute was achieved for the smallest number of indexes, i.e. just 10 indexes which match nodes in every document. Then we increased the number of indexes so that for any given document there are 50, 100, 150 or 200 indexes that did not match nodes. For the largest number of indexes the insert throughput was reduced by 41% to 56,615 inserts per minute. The percentage by which the throughput is reduced is shown by the curve in Figure 3. The throughput reduction is caused by increasing CPU consumption to compare nodes of the incoming XML documents against the chained list of XML indexes.

Clearly, for a very insert-intensive scenario such as the financial application logging, the performance overhead of the large number indexes is not acceptable. Thus, further optimizations are required to meet the performance requirements.

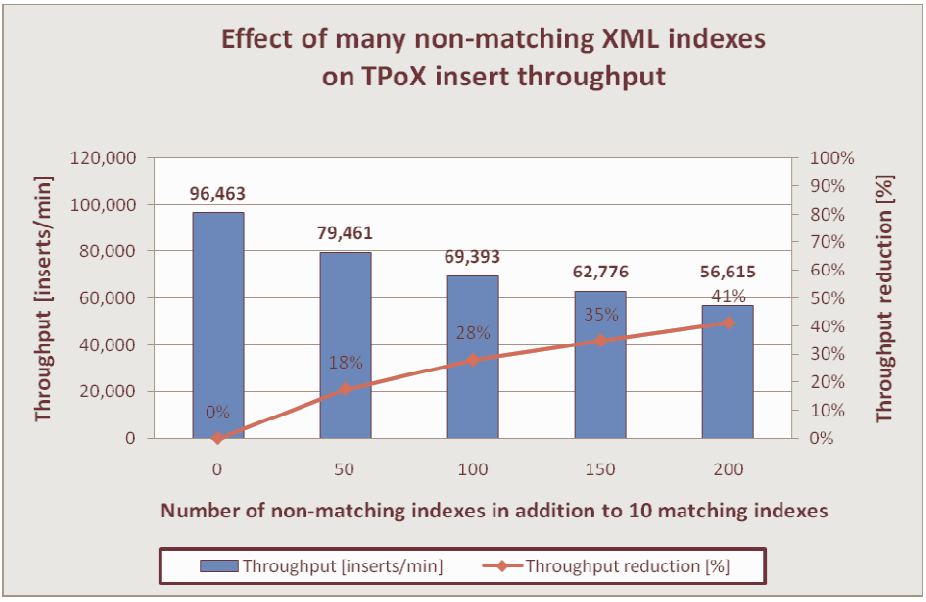



**Figure 3: Insert Performance relative to the number of Indexes (simple index chaining approach)**

## 4.2 Multi-level Chaining

The pattern tree approach discussed earlier is expected to meet the performance requirements for the runtime execution part. The reason is that at runtime we would move up and down in the tree based on the context of the incoming XML data. Non-matching indexes impact the navigation slightly because they add additional choices

when moving to a child node in the tree and contribute to the overall size of the pattern tree. However, during the runtime setup when the metadata from the physical objects can be combined, the more patterns are added to such a tree during this phase, the more time it takes. Thus, pattern trees are not our preferred choice as they don't keep the desired balance between setup and execution costs.

We mentioned earlier that we found chains to work well for a small number of indexes. In situations where a large number of indexes is required, documents are typically highly variable and the elements and attributes that need to be indexed are distributed across several different levels in the documents. We can use this observation to split the index chain and manage it in smaller chunks. Thereby, we can get back to the short chains which are efficient and meeting the simplicity requirement. One obvious way of splitting up the index chain is by the matching level, i.e., the document depth that the `XMLPATTERN` of an index definition points to. For every possible level, in DB2 up to 125 levels, we maintain a separate chain. In the application logging scenario the documents vary in structure and depths with a typical maximum depth of 10 to 15, with indexed nodes being located anywhere from level 2 to 15. This means that the list of indexes is split into 14 chains. To account for indexes with * and //, the chain for level 0 is used and is always checked. Level 0 can be used for this special purpose because it does not contain any other indexes. The multi-level chain design is illustrated in Figure 4. Additionally, the "Late in, Early out" optimization is used to process the chains.

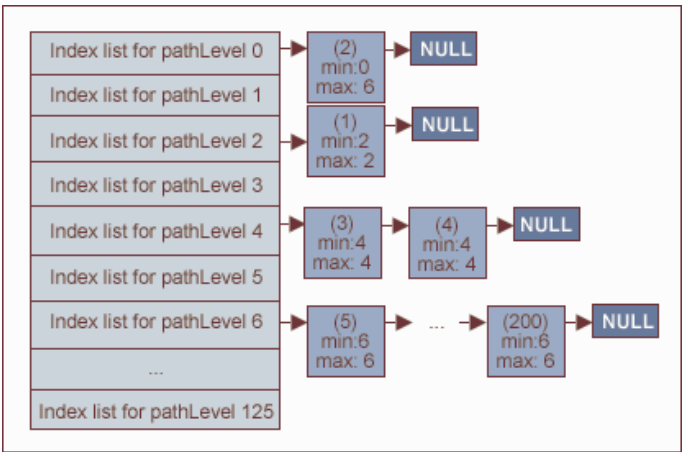



**Figure 4: XML Index lists for each path level**

**Performance results**

With the new approach of managing the indexes in separate chains for each path level we repeated the insert performance tests. As illustrated in Figure 5 the maximum overhead of index maintenance is reduced to 6%. Also for more moderate numbers of non-matching indexes, such as 50, the impact of indexes that do not match nodes in every document is as low as 2%. What is also visible is that even the case with zero non-matching indexes benefits from the separation.

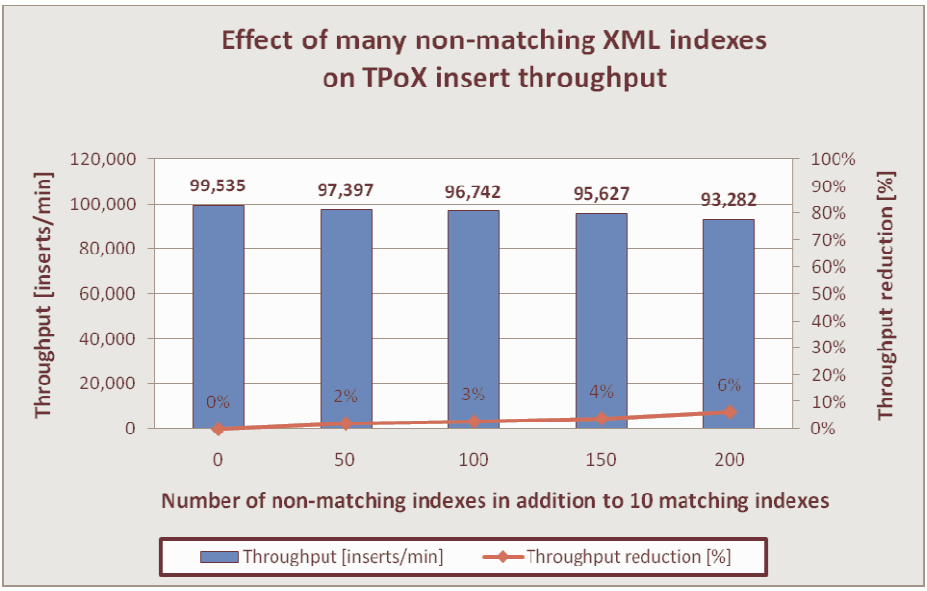



**Figure 5: Insert Performance relative to the number of Indexes (multi-level chaining approach)**

## 5 Conclusions and Outlook

In this paper, we discussed the special requirements coming from novel XML-based database application – application logging. It poses special requirements towards native XML database systems because of the mix of all kinds of different XML documents that are stored in a single XML column. In order to support search performance indexes need to be defined for the different document types. This leads to the presence of sometimes multiple hundred XML indexes, something unknown from the relational world. Because application logging is an insert-driven application the question is how the scalability is impacted by so many indexes. As we have shown in this paper, both the performance requirements as well as the design principle of simplicity and maintainability can be met with an approach that is based on what is done in the relational world, index chaining. We have extended this to multi-level chaining and enhanced how and when the index patterns are matched against the currently processed document. With these techniques DB2 scales well in this rather extreme application scenario.

The application logging does not only create challenges for the indexing in native XML database systems, but also for the storage components. The log data from dozens or hundreds of application generates a tremendous amount of XML data that needs to be processed – inserted first and after a retention period moved out of the

database system again. The size of the storage structures, the size of the generated database transaction logs, and many more factors impact the scalability. We therefore are looking forward to meeting those challenges by continuously improving XML storage and indexing technologies.

## References

[1] Balmin, A. et al.: On the Path to Efficient XML Queries, *32nd International Conference on Very Large Databases*, September 2006.

[2] Chicago Mercantile Exchange: "*The Business Case for FIXML*", 2004, http://www.cme.com/files/BusinessCaseFIXML.ppt http://www.cme.com/clearing/cm/stan/index.html

[3] Malaika, S.: Get started with Industry Formats and Services with pureXML, *IBM developerWorks*, http://www.ibm.com/developerworks/db2/library/techarticle/dm-0705malaika/

[4] Nicola, M., Kogan, I., Schiefer, B.: An XML Transaction Processing Benchmark, *ACM SIGMOD Conference* , June 2007.

[5] Nicola, M., Van der Linden, B.: Native XML Support in DB2 Universal Database, *International Conference on Very Large Data Bases (VLDB),* 2005.

[6] Pal et al.: Indexing XML Data Stored in a Relational Database, *International Conference on Very Large Data Bases (VLDB),* 2004.

[7] The Financial Information eXchange Protocol, FIXML 4.4 Schema Specification 20040109, Revision1 2006-10-06 http://www.fixprotocol.org/specifications/fix4.4fixml

[8] XML on Wall Street, *Financial XML Projects*, http://lighthouse-partners.com/xml

[9] XML Database Benchmark, *Transaction Processing over XML (TPoX)*, http://tpox.sourceforge.net/

# IT-Infrastrukturen für flexible, service-orientierte Anwendungen - Ein Rahmenwerk zur Bewertung

Stephan Buchwald[1], Thomas Bauer[1] und Rüdiger Pryss[2]
[1]Abteilung für Daten- und Prozessmanagement, Daimler AG,
{stephan.buchwald, thomas.tb.bauer}@daimler.com
[2]Institut für Datenbanken und Informationssysteme, Universität Ulm
ruediger.pryss@uni-ulm.de

**Zusammenfassung:** Service-orientierte Architekturen (SOA) sind in vielen Unternehmen ein zwar noch entstehendes, aber bereits auch sehr wichtiges Thema. Ein entscheidender Aspekt jeder SOA stellt die Standardisierung der IT-Infrastruktur des Unternehmens dar. Dadurch sparen Unternehmen Kosten, da sie die IT-Anbieter leichter austauschen können. Ferner reduziert sich durch Standardisierung der Wissensaufwand bei der Softwareentwicklung und die Funktionalität der IT für die Fachanwender wird so vereinheitlicht. Dies bedeutet, dass für jede im Unternehmen benötigte Funktionalität ein Produkt ausgewählt werden muss, das dann für IT-Applikationen als Implementierungsplattform fest vorgegeben ist (bzw. eine kleine Anzahl unterschiedlicher Produkte ggf. verschiedener Hersteller). Obwohl für eine solche Entscheidung die potentiell relevanten SOA-Komponenten bekannt sein sollten, gibt es in der Literatur keine systematische und produktunabhängige Darstellung und Bewertungsgrundlage von IT-Infrastrukturen für eine SOA. Da jedes Unternehmen unterschiedliche Anforderungen an die Flexibilität einer SOA hat, ist zudem eine Betrachtung unterschiedlicher Ausbaustufen der einzelnen SOA-Komponenten sinnvoll. Die Anforderungen der Unternehmen erstrecken sich von mehr Funktionalität für die Benutzer bis hin zu mehr Funktionalität bei der Prozesssteuerung. Darüber hinaus sollen Kosten infolge redundanter Implementierungen vermieden werden. Aus diesem Grund betrachtet dieser Beitrag Komponenten einer SOA ebenso wie eine Darstellung ihres Zusammenspiels in einer IT-Gesamtinfrastruktur. Der Fokus liegt auf Geschäftsprozessen, die geeignet durch IT-Applikationen unterstützt werden sollen.

## 1 Einleitung

Ein entscheidender Erfolgsfaktor für Unternehmen ist ihre Anpassungsfähigkeit auf Änderungen ihrer Umgebung [MRB08, Rei00, RMRD04, RMR07]. Diese Fähigkeit wird zum Wettbewerbsvorteil, wenn die jeweiligen Anpassungen schneller und kostengünstiger realisierbar sind als bei Konkurrenten. Um diesen Vorsprung zu erreichen, wird die geforderte Informationstechnologie (IT) immer mehr zum Schlüsselfaktor. Unternehmen haben erkannt, dass die geforderte Anpassungsfähigkeit mit monolithischen IT-Systemen nicht realisierbar ist. Um Anpassungen dennoch durchführen zu können, wurden im Laufe der Zeit zahlreiche Technologien und Methoden in den Unternehmen etabliert. Diese wurden eingesetzt, um die Abläufe zwischen den IT-Systemen abzubilden und damit die Integration zu ermöglichen.

Das kontinuierliche Anwenden neu entwickelter Technologien, mit dem Ziel der Integration von IT-Systemen, führte zu immer komplexer werdenden Unternehmenslandschaften. Die Komplexität ist in der Nutzung unterschiedlicher Integrationslösungen begründet, was zu intransparenten Abläufen geführt hat [RD00]. Erschwerend kommt hinzu, dass zusätzlich zur fehlenden Transparenz der Abläufe im eigenen Unternehmen auch die Beziehungen mit Partnern immer komplexer werden. Um dennoch die Transparenz dieser komplexen Abläufe zu erhöhen, werden diese losgelöst von den IT-Systemen dokumentiert, d.h. auf einer fachlichen Ebene. Dies bildet auch die Grundlage für Optimierungen. Diese Abläufe stellen die fachlichen Anforderungen dar, die von den IT-Systemen erfüllt werden müssen, um die Geschäftsfähigkeit des Unternehmens sicherzustellen.
Das Problem der Anpassungsfähigkeit resultiert aus der Notwendigkeit zur effizienten Abbildung der fachlichen Anforderungen, repräsentiert durch die Abläufe im Unternehmen, auf die IT-Systeme. Die fachliche Sicht muss sehr viel stärker betont werden als bisher. Dazu werden die Fachanwender des Unternehmens häufiger in die Anpassung der IT-Systeme einbezogen. Die bisher lang andauernden Software-Entwicklungszyklen werden durch diese Maßnahme kürzer, da Änderungen des Fachanwenders gezielter in die IT überführbar sind. Aus diesen Gründen müssen die fachliche und technische Sicht eines Unternehmens enger aufeinander abgestimmt werden.
Getrieben durch diese Erkenntnis versuchen Unternehmen ihre vorhandene Systemlandschaft service-orientiert auszurichten [Erl05]. Aus fachlicher Sicht beschreibt ein Service eine Funktionalität, die im Unternehmen angeboten oder verwendet wird. Die Einführung von Services und dazu notwendiger Technologien und Methoden führt zu einer service-orientierten Architektur (SOA). Darunter versteht man ein Architekturparadigma, welches das Modellieren von Services, das Ausführen von Services sowie das Kapseln von Funktionalität durch Services und die service-orientierte Softwareentwicklung unterstützt. Hierunter fallen vor allem die Ausrichtung der IT an fachlichen Anforderungen und die schnelle Reaktion auf (geänderte) fachliche Anforderungen. Darüber hinaus bilden Service-Prinzipien wie Kapselung, lose Kopplung, standardisierte Schnittstellen, Auffindbarkeit, Wiederverwendbarkeit und Autonomie von Services die Basis für eine SOA [Erl05].
Wesentlich für die Realisierbarkeit dieser Funktionalitäten ist eine standardisierte, service-orientierte IT-Infrastruktur. Diese beinhaltet funktionale Komponenten zur Modellierung von fachlichen Anforderungen und deren Abbildung auf IT-Systeme. Die IT-Infrastruktur muss so gestaltet sein, dass die genannten SOA-Prinzipien umgesetzt werden können. Um die fachlichen Anforderungen abzubilden, werden Geschäftsprozesse modelliert, die die Abläufe im Unternehmen beschreiben [RD00]. Änderungen an Geschäftsprozessen müssen gestützt durch die IT-Infrastruktur flexibel auf die Implementierung abgebildet werden können. Sowohl die Geschäftsseite (z.B. Änderungen von gesetzlichen Rahmenbedingungen) als auch deren fachliche Umgebung (z.B. organisatorische Umstrukturierung) lösen Änderungen aus. Außerdem können Änderungen durch die Umgebung der Implementierung (z.B. dem physischen Ort der Service-Implementierung) initiiert werden.
Um Unternehmenslandschaften und deren IT-Systeme möglichst flexibel in einer SOA betreiben zu können, ist eine IT-Infrastruktur notwendig, welche die dafür geeigneten Komponenten verwendet. Obwohl Hersteller, Gremien und Autoren zahlreiche SOA-Produkte und -Konzepte beschreiben, gibt es keine konzeptionellen und plattformunabhängigen Betrachtungen von Komponenten sowie deren Funktionalität und Zusammenspiel. Des-

halb stellt dieser Beitrag eine produktunabhängige Sicht auf die Komponenten einer IT-Infrastruktur vor, welche deren Bewertung, Analyse und Weiterentwicklung unterstützt. Das Ergebnis dieser Betrachtung ist ein Rahmenwerk, das eine Orientierungshilfe hinsichtlich der Bewertung einer Unternehmenslandschaft bietet. Auf Basis dieser Bewertung soll eine Weiterentwicklung der bestehenden Unternehmenslandschaft in Richtung Service-Orientierung ermöglicht werden.
In Kapitel 2 wird ein Anwendungsszenario vorgestellt. Dieses führt einen vereinfachten Geschäftsprozess ein, der zur Verdeutlichung der Überführung von „hart verdrahteten" IT-Systemen zu service-orientierten Architekturen dient. Bei der Beschreibung von Komponenten in Kapitel 3 wird dieses Anwendungsszenario aufgegriffen, um aufzuzeigen, wieso eine IT-Infrastruktur die dargestellten Komponenten beinhalten muss. Wir unterteilen unsere IT-Infrastruktur dazu in zwei Entwicklungsstufen, die dazu dienen, existierende IT-Systeme service-orientiert weiterzuentwickeln. Kapitel 4 betrachtet den Stand der Technik, bevor der Beitrag mit einer Zusammenfassung und einem Ausblick schließt.

## 2 Anwendungsszenario und Ziele

Wir diskutieren im Folgenden ein typisches Anwendungsszenario. Abbildung 1 zeigt einen Prozess für das Änderungsmanagement in der Fahrzeugentwicklung. Dieser ist vereinheitlicht und verfremdet dargestellt, beinhaltet aber realitätsnahe Anforderungen. Er stellt sicher, dass Änderungsvorhaben an Bauteilen vor ihrer eigentlichen Umsetzung bewertet, genehmigt und entsprechend dokumentiert werden.

Im Geschäftsprozessschritt ① wird ein Änderungsvorhaben angelegt, indem die Änderung in Form eines Änderungsantrages beschrieben wird. Dieser enthält neben der Änderungskategorie eine Beschreibung der Änderung, mögliche Maßnahmen zur Umsetzung, Angaben zu betroffenen Bauteilen sowie eine grobe Abschätzung resultierender Kosten.

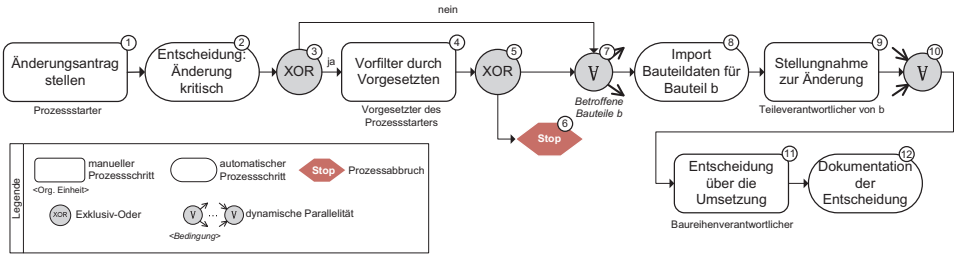



Abbildung 1: Geschäftsprozess zum Änderungsmanagement

Eine Änderung wirkt sich oft auf viele Bauteile aus. Das Durchlaufen des Änderungsprozesses kann deshalb hohe Kosten (für Stellungnahmen) verursachen. Diese sollten vermieden werden, wenn eine Umsetzung der Änderung unwahrscheinlich ist. Aus diesem Grund

bewertet im Prozessschritt ④ der Vorgesetzte des Antragstellers die Erfolgsaussichten des Änderungsvorhabens. Gegebenenfalls kann er den Änderungsprozess vorzeitig beenden ⑥. Allerdings sollen auch bei dem Vorgesetzten unnötiger Aufwand sowie unnötige Verzögerungen durch den Prozessschritt ④ vermieden werden. Deshalb wird dieser nur ausgeführt, wenn ein solcher Vorfilter sinnvoll ist. Wann dies der Fall ist, entscheidet ein automatischer Prozessschritt mittels vordefinierter Geschäftsregeln ②. So kann ein Vorfilter bei hoher Anzahl zu bewertender Bauteile, in Kombination mit hohen erwarteten Kosten für die Umsetzung der Änderung, sinnvoll sein.
Für alle vom Änderungsvorhaben betroffene Bauteile werden entsprechende Bauteildaten (z.B. Teilenummer, Beschreibung, Material, etc.) automatisch aus dem jeweiligen Produktdaten-Management-System (PDM) importiert ⑧. Anschließend geben die Bauteilverantwortlichen eine Stellungnahme zur Änderung ab ⑨. Hierbei werden insbesondere die technische Realisierbarkeit der Änderung sowie die tatsächlich resultierenden Kosten am Bauteil bewertet.
Der Baureihenverantwortliche entscheidet im Prozessschritt ⑪ über die Umsetzung des Änderungsvorhabens. Diese Entscheidung wird automatisiert dokumentiert ⑫. Es werden sowohl Informationen an das PDM zurückübermittelt, als auch an alle IT-Systeme übertragen, die später an der Umsetzung des Änderungsvorhabens beteiligt sind.
Das zuvor beschriebene Anwendungsszenario wurde bereits einer fachlichen Optimierung unterzogen, d.h. das Anwendungsszenario aus Abbildung 1 hatte bei der ersten Implementierung eine andere Struktur. Ursprünglich waren die Prozessschritt ③ bis ⑥ nicht vorhanden und sind erst durch eine fachliche Optimierung hinzugekommen. Aufgrund der nachträglichen Änderung am Geschäftsprozess, musste dieser erneut auf die IT-Systeme abgebildet werden.
Ziel einer SOA ist es zum einen, aus solch fachlich beschriebenen Geschäftsprozessen schnell und kostengünstig eine IT-Implementierung abzuleiten. Zum anderen sollten Änderungen, wie beispielsweise das Einfügen der Prozessschritte ③ bis ⑥, einfach realisiert werden können, um eine erneuten Implementierung des gesamten Geschäftsprozesses zu vermeiden. Deshalb sollte die Automatisierung durchgängig entlang des gesamten Prozesslebenszyklus gestaltet werden, um dadurch eine möglichst vollständige Informationsweitergabe von der fachlichen Geschäftsprozessbeschreibung bis zur konkreten Implementierung zu realisieren. Insbesondere die Änderbarkeit von Geschäftsprozessen und die zugehörige Synchronisation mit der konkreten IT-Implementierung sind essentiell, sowohl aus Sicht des Geschäftsprozessmodells in Richtung der IT-Implementierung, als auch in entgegengesetzter Richtung. Dadurch wird es möglich, Geschäftsprozesse in einer SOA zu etablieren und auf Änderungen zu reagieren. Ein Faktor zur Verringerung der Kosten für die Implementierung von Geschäftsprozessen ist die Wiederverwendung existierender Funktionalität. Beispielsweise sollte im Geschäftsprozessschritt ⑧ ein Service verwendet werden, der bereits existiert und die notwendigen Informationen über ein Bauteil bereitstellt.
Die zuvor beschriebenen Ziele lassen sich nur erreichen, wenn dafür geeignete Rahmenbedingungen geschaffen werden. Dazu gehört unter anderem, dass Informationen über fachliche Anforderungen und Geschäftsprozesse sowie Informationen, die für die konkrete Implementierung notwendig sind, zentral zugreifbar dokumentiert sein müssen. Dadurch können beispielsweise geeignete Services für die Geschäftsprozessschritte ⑧ und

⑫ bereits in frühen Phasen des Prozesslebenszyklus gefunden und verwendet werden. Organisatorische Faktoren wie Enterprise Architecture Management [MBL07] oder zentral gesteuerte Governance-Prozesse[1] und -Gremien nutzen diese Information, indem sie beispielsweise die zentral abgelegten Informationen als Grundlage für Entscheidungen verwenden. Neben der reinen Informationsspeicherung müssen insbesondere Abhängigkeiten zwischen Geschäftsprozessen, Services, IT-Systemen und Bearbeitern verwaltet werden. Auf Basis dieser Informationen setzen Analysemethoden an, um auftretende Fehler frühzeitg zu erkennen und entsprechend darauf reagieren zu können. Beispielsweise kann analysiert werden, welche Services von welchen Geschäftsprozessen verwendet werden. Diese Information ist wichtig, um bei Änderungen an Services, die davon betroffenen Geschäftsprozesse identifizieren und ggf. anpassen zu können.

## 3 Ausbaustufen einer IT-Infrastruktur

Das Zusammenspiel aller IT-Systeme verbessert die Geschäftsfähigkeit eines Unternehmens. Hierzu müssen die IT-Systeme miteinander interagieren. Um die grundlegende Funktionsfähigkeit der Unternehmenslandschaft sicherzustellen, ist die Integration der IT-Systeme notwendig. Diese Fähigkeit wird technisch durch IT-Infrastrukturen unterstützt. Darunter versteht man die Datenverarbeitung und die dafür benötigten Hard- und Softwarekomponenten sowie unterstützende Technologien zur Kommunikation zwischen diesen Komponenten. Da die Komplexität der IT-Systeme oftmals groß ist, ist schwer bewertbar, ob die im Unternehmen vorhandene IT-Infrastruktur in Bezug auf die Anpassungsfähigkeit adäquat ist. Um diese Bewertung vornehmen zu können, schlagen wir ein Rahmenwerk vor, welches Praktikern eine detaillierte Betrachtung der Komponenten und deren Zusammenspiel im Vergleich ermöglicht. Dieses Rahmenwerk beginnt mit der Beschreibung klassischer Integration auf Basis von Punkt-zu-Punkt-Verbindungen und endet mit der prozessorientierten Integration von Services. Bewusst ausgeklammert haben wir in dieser Betrachtung Aspekte wie Governance und Security. Eine solche Betrachtung würde diesen Beitrag sprengen, zumal unser Fokus auf der geeigneten Unterstützung von Geschäftsprozessen liegt.
Zur besseren Strukturierung unterteilen wir das Rahmenwerk in zwei Ausbaustufen, welche den oben genannten Übergang abbilden. Davor wird die Entwicklung von IT-Infrastrukturen vor Einführung von Services beschrieben, d.h. IT-Infrastrukturen wie sie heute in der Praxis meist zu finden sind. Aus service-orientierter Sichtweise sind diese meist unzureichend und weisen bei der Einführung von Services und einer SOA gewisse Probleme auf. Diese werden analysiert und motivieren zugleich die Ausgangsbasis für die beiden Ausbaustufen. Unternehmen erhalten durch diese die Möglichkeit zu evaluieren, an welcher Stelle sie sich in Bezug der Service-Orientierung befinden und was für eine Weiterentwicklung notwendig ist.

[1] Governance-Prozesse werden auch durch die IT-Umgebung gesteuert und unterstützt. Eine Behandlung der Governance-Prozesse geben wir indes nicht, da dies außerhalb unseres Beitrages liegt, welcher sich auf die Kern-Prozesse eines Unternehmens fokussiert.

### 3.1 IT-Infrastrukturen vor einer SOA-Einführung

Durch Eigenentwicklungen oder Zukauf von IT-Systemen hat sich in Unternehmen eine sehr heterogene IT-Infrastruktur entwickelt. Die Wartung und Pflege der immer komplexer werdenden IT-Infrastrukturen hat sich als schwer handhabbar herausgestellt [Pry05]. Hauptsächlich wird dies durch eine „harte Verdrahtung“ der IT-Systeme verursacht. Diese Verdrahtung ist meist als Punkt-zu-Punkt-Verbindung zwischen den einzelnen IT-Systemen realisiert. Dies führt bei kontinuierlichem Hinzufügen neuer IT-Systeme zu vielen Abhängigkeiten, deren Zusammenhänge nur schwer nachvollziehbar sind. Erschwerend kommt hinzu, dass die IT-Infrastrukturen mit Fortschritt der Technik gewachsen sind, d.h. unterschiedliche Standards und Technologien für die Integration verwendet werden. Oftmals geschieht der Datenaustausch manuell. Ferner kann der gemeinsame Zugriff von IT-Systemen auf die Daten über den Austausch von Dateien oder über eine gemeinsame Datenbank erfolgen. Hierbei stellt sich das Problem, dass die Integration ausschließlich über die Datenbasis erfolgt und der eigentliche Ablauf oft nicht nachvollziehbar ist. Eine Analyse der Abläufe ist somit aufgrund der komplexen Beziehungen zwischen den IT-Systemen nicht möglich.
Diese Betrachtung einer IT-Infrastruktur behandelt bisher nur innerbetriebliche Aspekte. Die Probleme verstärken sich im unternehmensübergreifenden Fall, da hier nicht nur die unternehmenseigenen IT-Systeme betrachtet werden müssen, sondern auch die des Partnerunternehmens. Um dieses Problem zu vermeiden, wird eine abstraktere Sicht auf die Integration benötigt. Die erste abstrakte Verfeinerung geben wir in der Ausbaustufe 1, welche eine service-orientierte Sicht einführt.

### 3.2 Ausbaustufe 1: Minimale service-orientierte IT-Infrastruktur

Ziel dieser Ausbaustufe ist es, von hart verdrahteten, heterogenen IT-Infrastrukturen und ihren Problemen wegzukommen und in Richtung einer service-orientierten Architektur zu gelangen. Deshalb werden Services zur Kapselung vorhandener IT-Systemfunktionalität eingeführt. Dadurch wird es möglich, standardisierte Technologien für die Kommunikation zwischen Services einzusetzen, wodurch auch Punkt-zu-Punkt-Verbindungen ersetzt werden. Neben einer solchen Kapselung ist die Wiederverwendung von Services ein wichtiger Schritt in Richtung einer service-orientierten Architektur. Damit IT-Systeme bereits existierende Services wiederverwenden können, müssen diese zentralisiert abgelegt und auffindbar sein, d.h. eine zentrale Speicherung von Informationen ist notwendig. Ein weiteres Ziel ist die Betrachtung von Geschäftsprozessen, wodurch Abläufe zwischen IT-Systemen transparent und analysierbar werden.
Um dies zu unterstützen, sind diverse Komponenten in der IT-Infrastruktur notwendig, z.B. eine Komponente zur Modellierung von Geschäftsprozessen und eine weitere zur Ausführung von Services. Abbildung 2 zeigt diese Komponenten und stellt zugleich deren Zusammenspiel mittels Informationsfluss-Kanten dar. Im Folgenden werden die Komponenten detailliert beschrieben.

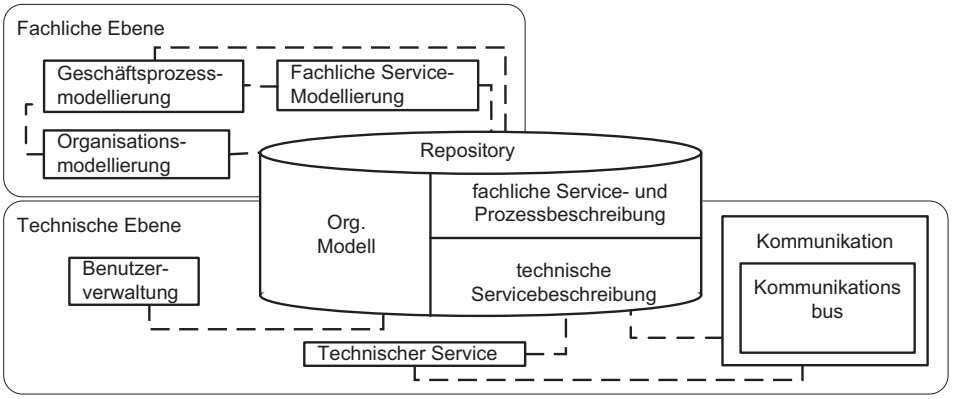



Abbildung 2: Ausbaustufe 1 - Minimale serviceorientierte IT-Infrastruktur

### 3.2.1 Fachliche Service-Modellierung

Fachliche Services sind Dienste, welche in einem Unternehmen angeboten oder konsumiert werden. Bei der Modellierung fachlicher Services geht es ausschließlich um eine fachliche Dokumentation der Dienste. Die Wiederverwendung solcher Dokumentationen ist ein weiteres wichtiges Ziel. Beispielsweise kann der fachliche Service zum Importieren von Bauteildaten (Geschäftsprozessschritt ⑧ in Abbildung 1) oder für die Dokumentation über die Entscheidung einer Umsetzung (Geschäftsprozessschritt ⑫) in verschiedenen Geschäftsprozessen (wieder) verwendet werden.
Der Nutzen einer fachlichen Modellierung besteht aus der Unterstützung beim Auffinden von fachlichen Services, der Komposition und dem Austausch von Services [Ste08]. Um diesen Nutzen zu realisieren, werden auf fachlicher Ebene Anforderungen definiert, die der fachliche Service unterstützen muss. Diese betreffen die Vergabe fachlichen Service-Bezeichner und der Signatur des fachlichen Service sowie der Unterstützung von Service-Level-Agreements (SLAs) und dem Aufstellen von Quality-of-Service Kriterien.
Beispielsweise muss für den fachlichen Service, der die Bauteildaten bereitstellt, definiert sein, welche Daten dieser benötigt, damit er im Prozessschritt ⑧ verwendbar ist. Grundsätzliches Ziel der fachlichen Service-Modellierung ist zudem, die Geschäftsfunktionen gegenüber Partnerunternehmen und auch innerhalb des Unternehmens gekapselt anzubieten.

### 3.2.2 Geschäftsprozessmodellierung

Geschäftsprozesse dienen der Dokumentation von fachlichen Anforderungen und realisieren damit die Geschäftsfähigkeit eines Unternehmens. Die einzelnen Fachbereiche des Unternehmens definieren Geschäftsprozesse, um fachliche Anforderungen aufzunehmen. Auf diese Art und Weise kann z.B. die in Kapitel 2.1 beschriebene Optimierung (Einführung der Geschäftsprozesschritte ② bis ⑥) beschrieben werden.

Daher ist bereits in dieser Ausbaustufe die Möglichkeit zur Geschäftsprozessmodellierung vorhanden. Außerdem ist eine Schnittstelle zur zentralen Speicherung in einem Repository (vgl. Abschnitt 3.2.4) erforderlich, damit bereits existierende fachliche Services zur Geschäftsprozessmodellierung verwendet werden können. Neben diesem rein lesenden Zugriff auf das Repository, müssen Informationen aus Geschäftsprozessen für strategische Vorgaben abgelegt werden können, z.B. Informationen über den Prozessverantwortlichen.

### 3.2.3 Modellierung des Organisationsmodells

Jedes Unternehmen besteht aus einer Vielzahl von Mitarbeitern und Fachabteilungen mit meist hierarchischer Organisationsstruktur. Rollen und Personen müssen sowohl bei der Modellierung als auch der Ausführung der Geschäftsprozesse einander eindeutig zugeordnet werden können. Deshalb werden Rollen, Abteilungen, etc. im Organisationsmodell des Unternehmens definiert [RMR07].
Die Organisationstruktur, inklusive der entsprechenden Rollen, sollte zentral vorgegeben und in allen IT-Systemen verwendet werden. Dadurch reduziert sich der Pflegeaufwand für das Organisationsmodell. Die zentral gespeicherten Informationen werden bei der Geschäftsprozessmodellierung und der fachlichen Service-Modellierung verwendet, um Zuständigkeiten und Berechtigungen für die Ausführung von Geschäftsprozessschritten oder den Aufruf von Services zu definieren. Heute existiert (noch) keine standardisierte Methode für die Definition von Organisationsmodellen, dennoch sollte eine Komponente in der IT-Infrastruktur existieren, um Organisationmodelle zu dokumentieren. Realisiert sind die Organisationsmodelle meist in bereits bestehenden Anwendungen wie SAP, Oracle oder Siebel. Eine weitere Möglichkeit ist die Speicherung in einem *Lightweight Directory Access Protocol* fähigen Directory (LDAP Directory) [YHK95].

### 3.2.4 Repository

In einer SOA werden ein zentrales Register zur Speicherung von Informationen sowie eine Dokumentation der Abhängigkeiten benötigt. [Jos07] unterscheidet zwischen einer Registry für Metadaten und einem Repository für Software-Artefakte. Hersteller von IT-Infrastrukturen folgen üblicherweise dieser Aufteilung [IBM07, HR08]. Wir unterscheiden in unserer IT-Infrastruktur zwischen dem fachlichen Repository für die Geschäftsprozess-Beschreibung, die Service-Beschreibung und Metadaten sowie dem technischen Repository für die zur Ausführungszeit von Services benötigten Informationen. Letzteres entspricht somit weitgehend der Registry. Ferner verstehen wir unter Repository die Kombination aus fachlichem und technischem Anteil.
Eine wichtige Frage in diesem Zusammenhang ist, mit welchen Informationen ein solches Repository befüllt werden soll. Dies wird im Folgenden diskutiert. Außerdem muss eine Schnittstelle existieren, um Repository-Daten zu schreiben und zu lesen.

- **Fachliche Sicht:** Daten des fachlichen Repositories entstehen bei der Modellierung der Geschäftsprozesse, des Organisationsmodells und der fachlichen Services. Es werden Informationen wie Prozessversionen, Prozessbeschreibung, Gültigkeitszeitraum des Prozesses und die konkrete Bearbeiterrolle aus dem Organisationsmodell

zu den einzelnen Schritten der Geschäftsprozesse gespeichert. Darüber hinaus wird die Information über die Verwendung fachlicher Services in Geschäftsprozessen im Repository abgelegt. Diese Informationen bestehen unter anderem aus Servicequalität (z.B. Reaktionszeit), Funktionalität (welchen Zweck erfüllt der Service) und Rahmenbedingungen (unter welchen Umständen erbringt der Service seinen Dienst) des Services als auch die von ihm erzeugten Geschäftsobjekte.

- **Technische Sicht:** Das technische Repository speichert Informationen über technische Services, die zur Ausführung notwendig sind (vgl. Abschnitt 3.2.6). Diese Informationen können z.B. den aktuellen Status (Service ist produktiv geschalten oder in Testphase) beschreiben. Es handelt sich um Informationen, welche die konkreten Schnittstellen (z.B. WSDL), die zugehörigen Schemata (XSD) sowie Laufzeitinformationen betreffen [ACKM04].

- **Organisatorische Sicht:** Der organisatorische Teil des Repositories dient der zentralen Speicherung (meist LDAP-fähiges Verzeichnis [YHK95]), Pflege und Dokumentation der Organisationsmodelle, um die Auflösung von Rollen, Personen und Verantwortlichkeiten zur Laufzeit zu ermöglichen.

### 3.2.5 Benutzerverwaltung

Der Administrator (oder eine autorisierte Person) legt sowohl die hierarchische Struktur als auch die Benutzer und Rollen des Unternehmens an, die dann im Repository gespeichert werden. Eine Berechnung konkreter Mitarbeiter zu einer vorgegebenen Rolle erfolgt zur Ausführungszeit auf Basis des Repositories [RD00]. Die bei der Ausführung der Geschäftsprozessimplementierung einem Prozessschritt zugeordnete Person wird bei der Ausführung berechnet (so können den Geschäftsprozessschritt ⑤ aus Abbildung 1 nur diejenigen Personen bearbeiten, die der Rolle Teileverantwortlicher zugeordnet sind).

### 3.2.6 Technische Services

Technische Services realisieren die Schnittstelle zu den IT-Systemen der Unternehmenslandschaft. Teile der Funktionalität (sofern sie auch für andere Anwendungen oder Domänen relevant ist) verwendeter IT-Systeme werden durch die technischen Services abstrahiert. Geschäftsprozessschritte, die automatisiert ausgeführt werden, z.B. der Import von Daten aus einem PDM (vgl. Geschäftsprozessschritt ④) benutzen auf diese Weise die IT-Systeme der Unternehmenslandschaft. Durch die Abstraktion werden dem Geschäftsprozess die Daten der verwendeten IT-Systeme zur Verfügung gestellt. So wird über einen technischen Service *getBauteilDaten()* dem Geschäftsprozessschritt ⑧ aus Abbildung 1 die benötigte Information zur Verfügung gestellt. Technische Service haben standardisierte Interfaces (z.B. WSDL), welche im Repository abgelegt werden.
Da nicht alle technischen Services unmittelbar auf die IT-Systeme zugreifen, werden diese weiter unterteilt [ELK+06]. Diese Gliederung wird für Ausbaustufe 2 beibehalten und stellt sich wie folgt dar (vgl. Abbildung 3):

- **Elementare Services:** Unternehmenslandschaften bestehen aus einer Vielzahl von IT-Systemen. Dazu zählen Datenbanken, Legacy-Anwendungen [Pry05] und individuelle Entwicklungen. Um IT-Systeme in service-orientierten IT-Infrastrukturen nutzen zu können, müssen sie abstrahiert werden. Dazu wird entweder die benötigte Funktionalität mittels Adapter-Technik als Service publiziert oder eine existierende Schnittstelle (Programmierschnittstelle des IT-Systems) durch Transformation einem Service-Standard angepasst [ACKM04].
- **Zusammengesetzte Services:** Damit technische Services die vom Geschäftsprozess geforderte Funktionalität zur Verfügung stellen können, müssen Daten oder Funktionen aus mehreren IT-Systemen durch elementare Services extrahiert werden. In diesen Fällen werden elementare Services komponiert. Sie bilden in Kombination die geforderte Funktionalität ab. Eine solche Komposition kann in Ausbaustufe 1 der IT-Infrastruktur z.B. durch eine J2EE-Session-Bean realisiert werden. In der nächsten Ausbaustufe kann man hierzu einen Orchestrierungsstandard [RS04] wie WS-BPEL [AAA+07] nutzen.
- **Öffentliche Services:** Dies sind entweder elementare Services oder zusammengesetzte Services. Wir gehen im Folgenden von öffentlichen Services als Grundabstraktion der IT-Systeme aus. Somit sind öffentliche Services die technischen Repräsentionen, welche in den Geschäftsprozessen verwendet werden.

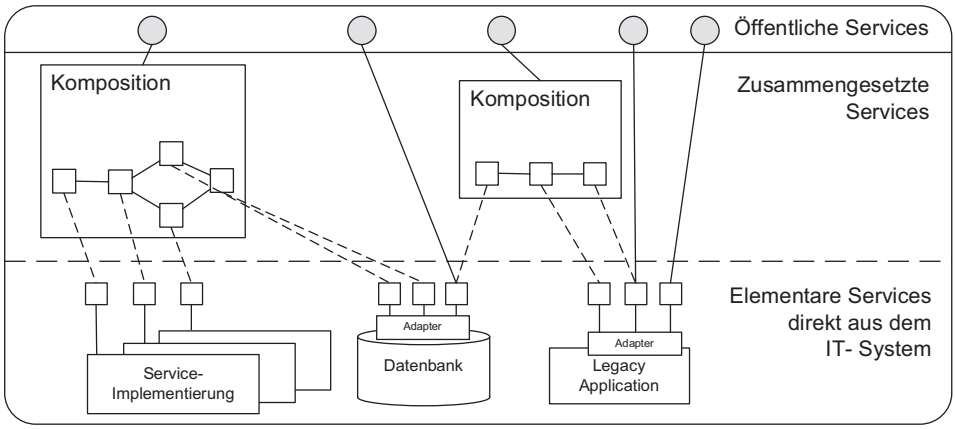



Abbildung 3: Kategorisierung technischer Services

### 3.2.7 Kommunikationsbus

Für Implementierungen von Geschäftsprozessen werden öffentliche Services benötigt. Diese müssen technisch aufrufbar sein. Deshalb wird ein Kommmunikationsbus eingeführt, der diesen Aufruf realisiert. Dazu erhält dieser die Information über den öffentlichen Service (z.B. XSD und WSDL) aus dem Repository. Mit dieser Information ruft er

den Service durch ein entsprechendes Routing des Funktionsaufrufs (inklusive der enthaltenen Daten). Prinzipiell könnte in Ausbaustufe 1 auch auf das Repository zur Speicherung der Informationen verzichtet werden. Dann müsste jedoch die Information über den öffentlichen Service-Aufruf und dessen Routing fest im Kommunikationsbus implementiert sein. In Ausbaustufe 1 sehen wir den Kommunikationsbus nicht zur Transformation der Daten. Da meist aber auch nicht von einem einheitlichen Datenformat ausgegangen werden kann, müssen die Daten-empfangenden Anwendungen die Transformation selbst durchführen. Gewachsene IT-Landschaften beinhalten oftmals einen Kommunikationsbus dieser Art (ohne explizite Datentransformationsfunktionalität). Wie gezeigt, ist diese notwendig und sollte zentral im Kommunikationsbus erfolgen. Diese Erweiterung beschreiben wir in Ausbaustufe 2.

### 3.3 Ausbaustufe 2: Typische service-orientierte IT-Infrastruktur

In diesem Abschnitt beschreiben wir die aus unserer Sicht notwendige IT-Infrastruktur, um Geschäftsprozesse technisch umzusetzen (vgl. Kapitel 2) und flexibel auf Änderungen zu reagieren. Es soll an dieser Stelle erwähnt werden, dass in vielen praktischen Fällen noch nicht einmal die Ausbaustufe 1 umgesetzt wurde.
In Ausbaustufe 1 ist es möglich, öffentliche Services aufzurufen, jedoch mit der Einschränkung, dass die Hintergrundausführung mehrerer Services hart verdrahtet erfolgt. Um eine lose Kopplung zwischen den öffentlichen Services zu erreichen, und auch deren Aufruf möglichst dynamisch zu realisieren, ist erweiterte Funktionalität notwendig.
Ausgehend von Ausbaustufe 1 beschreibt Ausbaustufe 2 weitere Komponenten, welche die geforderte Funktionalität bereitstellen. Die einzelnen Komponenten werden anschließend in kombinierbaren Unterstufen detailliert. Die Unterstufen dienen dem Zweck, Komponenten und deren Abhängigkeiten untereinander zu beschreiben. Wenn beispielsweise öffentliche Services erst zur Ausführungszeit der Implementierung bekannt sind, reicht die Funktionalität des Kommunikationsbusses aus Ausbaustufe 1 nicht aus. Dies bedeutet, es muss Funktionalität geschaffen werden, welche das Binden von öffentlichen Services zur Ausführungszeit ermöglicht. Die Geschäftsprozessschritte ⑧ und ⑫ aus Abbildung 1 sind hierfür gute Beispiele: Die Lokation (Endpunkte) der Services muss nicht fest sein. Erst zur Laufzeit ist klar, welche Bauteildaten zu importieren sind. Eventuell sind für die Anfrage durch PDM-Systeme notwendig, die in verschiedenen Geschäftsbereichen angesiedelt sind. Unter Umständen sind sogar die Schnittstellen der verschiedenen PDM-Systeme unterschiedlich. Diese Funktionalität kann jedoch nicht alleine durch die Erweiterung des Kommunikationsbusses erreicht werden. Deshalb werden zusätzliche Komponenten notwendig und bestehende erweitert. Die resultierenden Abhängigkeiten werden durch eine detaillierte Beschreibung der Komponenten transparent gemacht. Erst dadurch läßt sich bewerten, welche Komponenten für eine Unternehmenslandschaft in welcher Detaillierung sinnvoll sind.

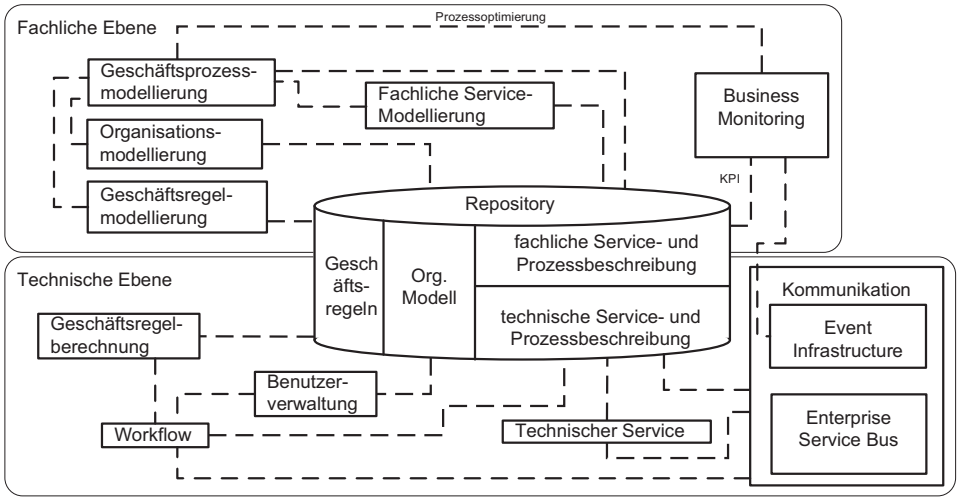



Abbildung 4: Ausbaustufe 2 der SOA IT-Infrastruktur

### 3.3.1 Geschäftsregeln

In Geschäftsprozessen sind Geschäftsregeln häufig mit Verzweigungsknoten verknüpft. Hierzu werden bei Verzweigungsentscheidungen in Geschäftsprozessen Variablen mittels Geschäftsregeln berechnet. Die Flexibilität des Geschäftsprozesses wird hierdurch erhöht, da auch Fachanwender diese Regeln zur Ausführungszeit anpassen können. In unserem Anwendungsszenario aus Kapitel 2 geschieht die Auswertung bei Geschäftsprozessschritt ② mittels einer Geschäftsregel. Dazu wertet die Geschäftsregel aus, inwieweit sich der Aufwand für den Geschäftsprozessschritt ④ rechnet.
Außerdem werden Geschäftsregeln für die Zuordnung von Ressourcen und Bearbeitern verwendet. Geschäftsregeln können unternehmensweit, für einen bestimmten Geschäftsprozess oder genau einen Prozessschritt definiert werden.
Die Geschäftsregeldefinition findet auf fachlicher Ebene statt. Die Berechnung dieser Regeln erfolgt je nach Art der Detaillierung (auf fachlicher Ebene) zur Ausführungszeit der Implementierung (technische Ebene). Die in Abbildung 2 dargestellte Komponente zur Geschäftsregelberechnung dient der beschriebenen Berechnung zur Ausführungszeit.

**Fachliche Ebene:** Auf fachlicher Ebene werden Geschäftsregeln von Fachanwendern definiert. Der Vorteil liegt in der Transparenz für den Geschäftsprozess. Die definierten Geschäftsregeln sind sofort als solche erkennbar, können im Geschäftsprozess verwendet werden und sind flexibel änderbar. Die Mächtigkeit zur Modellierung von Geschäftsregeln unterscheidet sich wie folgt und bedingt dadurch eine entsprechende Berechnungsmethode zur Ausführungszeit:

- Eine rudimentäre Definition von Geschäftsregeln geschieht meist durch eine textuelle Beschreibung.

- Idealerweise existiert zur Definition der Geschäftsregeln eine grafische (z.B. Regeleditor) oder anders geartete Tool-Unterstützung.
- Dieses Tool sollte die Geschäftsregeln geeignet im Repository ablegen.
- Als Erweiterung dieser Funktionalität können syntaktische und strukturelle Prüfungen der Geschäftsregeln automatisch vorgenommen werden.

**Technische Ebene:** Die fachlich modellierten Geschäftsregeln müssen zur Ausführungszeit genutzt werden können. Dazu erfolgt die Speicherung der Geschäftsregeln im Repository. Die Berechnung der Geschäftsregeln durch die Geschäftsregelberechnung kann ggf. als öffentlicher Service realisiert werden.

- Die Geschäftsregelberechnung greift zur Ausführungszeit (zur Berechnung der Geschäftsregeln) auf die im Repository hinterlegten Werte zu. Das heißt, es erfolgt eine Abfrage zur Ausführungszeit bei der Komponente zur Geschäftsregelberechnung. Diese liefert dann lediglich das Berechnungsergebnis anhand statisch vordefinierter Regeln zurück. Sollte auf fachlicher Ebene eine reine textuelle Definition der Geschäftsregeln erfolgt sein, muss diese nun manuell durch eine entsprechende Transformation in Regeln in das Repository eingepflegt werden.
- Ausgehend von der statischen Berechnung der Geschäftsregeln zur Ausführungszeit, ergänzt diese Unterstufe die Berechnung der Geschäftsregeln um einen dynamischen Aspekt. Dazu kann die Geschäftsregelberechnung zur Ausführungszeit dynamische Schwellwertänderungen an Regelbedingungen berücksichtigen. Das ermöglicht Fachanwendern selbst zur Ausführungszeit eine Einflußnahme auf die Geschäftsprozesse vorzunehmen.
- Im Vergleich zu den vorherigen Unterstufen können zusätzlich Geschäftsregeln zur Ausführungszeit ausgetauscht werden. Der Austausch ist oft mittels vordefinierter Geschäftsregel-Fragmente möglich. Dies bedeutet, dass ein Satz solcher Fragmente durch Fachanwender definiert wird, die zur Ausführungszeit kombiniert und verwendet werden. Dadurch kann z.B. im Prozessschritt ② aus Abbildung 1 zusätzlich berücksichtigt werden, ob die beantragte Änderung das Gewicht des Fahrzeugs positiv oder negativ beeinflusst und dadurch einen Vorfilter notwendig macht.

### 3.3.2 Workflow

In Ausbaustufe 1 wurde die Geschäftsprozesslogik fest in Programm-Code implementiert. Die in dieser Ausbaustufe verwendete Implementierung von Geschäftsprozessen wird hingegen durch Workflows realisiert. Nach [JBS97] ist ein Workflow die Automatisierung eines Geschäftsprozesses. Dabei werden alle zugehörigen Informationen über Geschäftsprozessschritte, Kontroll- und Datenflüsse sowie Ressourcenspezifikationen oder zeitliche Abhängigkeiten verwendet. Durch den Einsatz von Workflows lösen wir die Ablauflogik aus den IT-Systemen heraus [RD00]. Aufgrund dieser Zentralisierung und Entkopplung der Logik wird die Unternehmenslandschaft flexibler, da nicht nur auf fachlicher

Ebene durch den Geschäftsprozess, sondern auch auf technischer Ebene die Transparenz erhöht wird. Im Folgenden unterscheiden wir zwischen der Modellierung eines Workflow-Schemas und der Ausführung des Schemas durch eine Workflow-Engine.

**Workflow-Modellierung:** Bei der Workflow-Modellierung werden die fachlich modellierten Geschäftsprozesse technisch abgebildet. Dazu werden meist grafische Editoren zur Erstellung der Workflow-Schemas eingesetzt. Die Workflow-Schemas beschreiben dabei die Logik, wie öffentliche Services orchestriert [RS04] werden. Dadurch wird die Logik, welche bisher in den einzelnen IT-Systemen verteilt lag, an zentraler Stelle änderbar.

- Die erste Unterstufe beschreibt die Modellierung eines Workflow-Schemas. Dazu sind die Modellierung des Kontrollflusses, des Datenflusses und der Ressourcenzuteilung einzelner Geschäftsprozessschritte notwendig. Das resultierende Workflow-Schema kann später durch Ergänzung von Laufzeitinformationen von der Workflow-Engine ausgeführt werden.
- Der nächste Schritte ist die automatische Transformation eines Geschäftsprozesses in ein Workflow-Schema. In einem zweiten, manuellen Schritt muss dieses Schema um Informationen ergänzt werden, die zu dessen Ausführung notwendig sind. Dabei handelt es sich unter anderem um Informationen, die zum Aufruf der öffentlichen Services notwendig sind.

**Workflow-Ausführung:** Die Workflow-Engine führt das Workflow-Schema aus. Dabei steuert sie die Aufrufe öffentlicher Services (vgl. Geschäftsprozessschritte ②, ⑧ und ⑫ aus Abbildung 1) sowie das Auflösen von Ressourcen und die Zuordnung von konkreten Bearbeitern für manuell ausgeführte Geschäftsprozessschritte (vgl. ①, ④, ⑨, ⑪). Sowohl für das Aufrufen von öffentlichen Services als auch für die Interaktion zwischen den öffentlichen Services und notwendige Datentransformationen werden weitere Komponenten benötigt. Dies sind insbesondere Komponenten für die Kommunikation (Erweiterung des Kommunikationsbus) und die zentrale Informationsspeicherung (Repository).

- Die Workflow-Engine muss in der Lage sein, Workflow-Schemata auszuführen. Dazu gehören auch die eingangs erwähnten Aufgaben, etwa das Rufen von öffentlichen Services.
- In der Praxis muss oftmals auf eine Ausnahmesituation reagiert werden [WRRM08]. Beispielsweise könnte der Geschäftsprozessschritt ⑪ aus Abbildung 1 in einem Sonderfall entfallen, da die Umsetzung bereits durch einen Vorstandsbeschluss genehmigt wurde. Kontrollierte Ad-hoc Abweichungen ermöglichen solche Ausnahmesituationen bei der Ausführung von Workflow-Schemas [Rei00] zu behandeln. In Fällen einer Ad-hoc Abweichung muss die Workflow-Engine in der Lage sein, die vom Workflow-Schema abgeleiteten Workflow-Instanzen durch entsprechende Eskalationsmechanismen und Kompensationen zu unterstützen. Deshalb muss zur Vermeidung von Laufzeitfehlern ein manueller Eingriff möglich sein.
- Über den Aspekt einer Ad-hoc-Abweichung hinaus kann es notwendig sein, dass komplette Workflow-Schema zu ändern (und nicht nur eine einzelne laufende

Workflow-Instanz). In einem solchen Fall muss die Workflow-Engine die Migration laufender Workflow-Instanzen auf ein geändertes Schema unterstützen [Rin04, RMRD04]. Ferner unterscheidet die Workflow-Engine, ob laufende Workflow-Instanzen migriert werden oder auf dem alten Workflow-Schema weiterlaufen sollen. Auch für noch nicht gestartete Workflow-Instanzen kann die Workflow-Engine unterscheiden, ob das alte oder das neue Workflow-Schema verwendet werden soll.

### 3.3.3 Enterprise Service Bus (ESB)

Der ESB ist die Erweiterung des Kommunikationsbusses aus Ausbaustufe 1. Durch den zusätzlichen Einsatz von Geschäftsregeln und Workflow-Engine in Ausbaustufe 2, zur Erhöhung der Flexibilität, sind die Anforderungen an den ESB höher als an einen Kommunikationsbus. Die eindeutige Zuordnung der von Herstellern angebotenen Tools zur ESB-Komponente gestaltet sich oft schwierig. Am Beispiel des ESB wird daher gut die Notwendigkeit ersichtlich, allgemeine Kriterien zur Evaluierung der eigenen IT-Infrastruktur aufzustellen.

- Beim Kommunikationsbus der Ausbaustufe 1 wurden öffentliche Services statisch gebunden. Beim ESB erfolgt die Bindung von Services dynamisch: Endpunkte für öffentliche Services werden vom ESB zur Ausführungszeit mittels Informationen aus dem Repository ermittelt und gebunden. Der ESB bekommt lediglich eine Service-Spezifikation vom Service-Consumer (z.B. einer Workflow-Engine) übergeben und ruft mittels dieser den passenden öffentlichen Service. Durch das Zusammenspiel von ESB und Repository ist es möglich, öffentliche Services physisch zu verschieben ohne dabei laufende Workflow-Instanzen zu beeinträchtigen. Die eigentliche Information über die Lokation des öffentlichen Services ist im Repository hinterlegt und wird vom ESB zur Ausführungszeit verwendet.

- Über den Aspekt des dynamischen Bindens hinaus, ist der ESB in der Lage, erweiterte Suchanfragen an das Repository zu stellen. Diese Suchanfragen können über Service-Eigenschaften und Service-Merkmale - wie Service Level Agreements (SLA) - verfügen. Ein SLA könnte zum Beispiel lauten, dass der Service-Aufruf zwingend verschlüsselt ablaufen muss. Findet der ESB zu einer Servicespezifikation mehr als einen passenden Service, existieren vorgebene Strategien (z.B. der erste passende) zur Auflösung dieses Szenarios.

- Eine weitere Aufgabe des ESB ist die Transformation von Daten zwischen Service-Consumer und -Provider. Dazu wird im ESB meist die notwendige Transformation definiert.

- Um den im vorangegangenen Abschnitt beschriebenen Aufruf zwischen Service-Consumer und -Provider zu realisieren, werden im ESB statisch Abläufe definiert. Diese realisieren durch Komposition elementarer Services, und der dazu notwendigen Transformation der Daten, öffentliche Services (vgl. Abschnitt 3.2.6). Durch den einfachen und zentral konfigurierbaren Austausch dieser Abläufe wird die Flexibilität der IT-Systeme erhöht.

### 3.3.4 Event Infrastructure

In einer IT-Infrastruktur ist es wichtig, Ereignisse definieren zu können. Durch den Einsatz von Ereignissen können Anwendungen untereinander Daten austauschen, die außer den reinen Nutzdaten wichtige Statusinformationen liefern. Ferner können Ereignisse zum Überwachen von Anwendungen genutzt werden. Technisch wird dies durch einen Nachrichtenaustausch (standardisiertes Format) über den ESB realisiert. Mittels eines Publish und Subscribe-Mechanismus [Pry05] können sich Empfänger für relevante Ereignisse registrieren, etwa Start- und Endezeitpunkte (z.B. wann wurde Workflow-Instanz x auf Workflow-Engine z gestartet und beendet) sowie auch jede Art von Anwendungsdaten (z.B. Kundennamen) [Buc05]. So könnte ein Ereignis zum Geschäftsprozessschritt ⑨ aus Abbildung 1 definiert werden, welches anzeigt, ob der zuständige Teileverantwortliche den Freigabestatus schon erteilt hat. Dauert diese Freigabe zu lange, kann entsprechend darauf reagiert werden.
Durch das Überwachen (Business Monitoring) von Ereignissen ist es möglich, Engpässe bei der Bearbeitung von Aufgaben (z.B. sehr lange Arbeitslisten von Benutzern oder größere Verzögerungen bei einzelnen Aufgaben) zu identifizieren, was die Grundlage für Maßnahmen zur Prozessoptimierung ist.

### 3.3.5 Business Monitoring

**Business Activity Monitoring:** Ein wichtiger Aspekt, um die Qualität einer Unternehmenslandschaft messbar zu machen, ist das Überwachen (Monitoring) der Applikationen und Geschäftsprozesse. Überwachung meint hier nicht, dass technische Monitoring der IT-Infrastruktur auf Verfügbarkeit von Komponenten. Es geht vielmehr um eine geeignete Überwachung der Applikationen zur Sicherstellung der Geschäftsziele, was auch als Business Activity Monitoring (BAM) bezeichnet wird. Informationslieferant für BAM ist der ESB, der für das Monitoring relevante Ereignisse an die Monitoring-Komponente übermittelt. Die auf fachlicher Ebene definierten Key-Performance-Indicators (KPI) werden dann auf Basis dieser Ereignisse berechnet [AAC+08].

- In der ersten Unterstufe des BAM müssen die KPI - und dazu notwendige zu überwachenden Ereignisse - manuell in der Monitoring-Komponente definiert werden [BWRJ08]. Ein solcher KPI könnte zum Beispiel die maximale Durchlaufzeit eines Geschäftsprozesses sein, die von der Fachabteilung vorgegeben wird. Anschließend werden im ESB Ereignisse definiert, welche die Zeitpunkte der einzelnen Aktionen enthalten, und welche dann zur Ausführungszeit an die BAM-Komponente übermittelt werden. Auf dieser Basis berechnet das BAM die benötigte KPI. Beispielsweise soll so überprüft werden, wie sich der Vorfilter aus Abbildung 1 auf die Durchlaufzeit der Geschäftsprozessschritte ② bis ⑪ ausgewirkt hat.
- Als Erweiterung unterstützt das System Benutzer bei der technischen Definition von KPIs. Dazu leitet es, ausgehend von der Geschäftsprozessmodellierung und den fachlich modellierten KPIs, die für den ESB notwendigen Events ebenso ab, wie den in der BAM-Komponente notwendigen Algorithmus zur Berechnung der KPIs.

**Prozessvisualisierung:** Im Vergleich zu unserem stark vereinfachten Geschäftsprozess aus Abbildung 1 sind reale Prozesse oftmals sehr komplex. Da IT-unterstützte Geschäftsprozessabläufe dem Ziel Flexibilität bei Prozessänderungen und Effizienzsteigerung dienen sollen, müssen diese gut analysierbar sein. Eine adäquate Visualisierung von ausgeführten Workflow-Instanzen kann die Geschäftsprozessanalyse unterstützen. Hierbei benötigt nicht jeder Nutzer oder Analyst den kompletten Prozess, um seine Analyse durchzuführen, sondern vielmehr personalisierte Sichten (Views) auf den Geschäftsprozess [Bob08, BRB07]. Durch den Einsatz solcher Views kann zum Beispiel bei zu hohen Durchlaufzeiten erkannt werden, bei welchen Geschäftsprozessschritten Probleme auftreten, ohne dass eine Vielzahl irrelevanter Funktionen den Analysten verwirrt.

### 3.3.6 Repository

Das Repository der Ausbaustufe 2 speichert zusätzlich zur Stufe 1 noch Daten, die für den ESB, die Geschäftsregelberechnung, den Workflow und das Monitoring notwendig sind. So werden aufzurufende Service-Endpunkte für die Geschäftsprozessschritte ⑧ und ⑫ aus Abbildung 1 oder Schwellwerte der Geschäftsregel für ② aus Abbildung 1 verwaltet. Außerdem kann ein Repository noch weitergehende Funktionalität realisieren:

- Ein wichtiger zusätzlicher Aspekt ist die Speicherung und das Management von Versionen, die Dokumentation von Abhängigkeiten zwischen öffentlichen Services, Geschäftsprozessen und Daten, sowie zwischen fachlichen und technischen Objekten. Aufbauend darauf lassen sich Analysen durchführen, die Problemsituationen frühzeitig erkennbar machen und eine Reaktion darauf ermöglichen (bspw. ein im Geschäftsprozess verwendeter öffentlicher Service ist nicht mehr verfügbar).
- Neben der Dokumentation von Abhängigkeiten werden Analyseverfahren für Problemsituationen bereitgestellt (z.B. Suchen verletzter Abhängigkeiten). Außerdem werden Notifikationsmechanismen angeboten, welche die von Änderungen betroffenen Objekte (bspw. Geschäftsprozess- oder Service-Verantwortliche) informieren.

## 4 Stand der Technik

SOA ist ein Thema, mit dem sich viele Softwarehersteller, Gremien und Autoren beschäftigen. Seit der Definition von SOA durch Gartner [Nat03] hat sich in dieser Domäne sehr viel getan. Keine dieser Arbeiten bietet eine generelle Betrachtung der IT-Infrastruktur wie in Kapitel 3 erfolgt. Dies führt dazu, dass für Unternehmen die Bewertung der eigenen IT-Infrastruktur im Hinblick auf die Unterstützung von Änderungen schwer möglich ist.
Hersteller von SOA-Produkten (wie z.B. IBM) haben viele Komponenten in ihrem Portfolio, betrachten die IT-Infrastuktur aber nicht auf einer konzeptionellen Ebene. Ferner wird auch die Kombination der Komponenten nicht konzeptionell betrachtet. Als Ergebnis wird die Gesamtfunktionalität, die wir in unseren Ausbaustufen beschrieben haben, nicht erreicht. So hat IBM zum Beispiel mächtige ESBs [FN08] und Prozess-Engines. In der

Praxis wird jedoch zur Modellierung von Geschäftsprozessen meist ARIS [Sch00] bevorzugt. Dies erfordert es dann, die in ARIS modellierten Geschäftsprozesse auf dem IBM SOA-Portfolio technisch abzubilden. Dies ist nur ein Beispiel für die Kombination von Komponenten verschiedener Hersteller aufgrund unterschiedlicher Funktionalität. Unsere IT-Infrastruktur bietet die Möglichkeit, zu bewerten, welche Funktionalität eine Komponente aufweisen muss, um der Forderung, flexibel auf Änderungen reagieren zu können, gerecht zu werden.
Gremien wie OASIS [MLM+06] beschäftigen sich mit einer abstrakten Sicht auf SOA. Eine funktionale Beschreibung von Komponenten, etwa ein ESB, erfolgt nicht. Auch eine detaillierte Beschreibung, wie und in welcher Form Informationen im Repository abgelegt werden, existiert nicht. Ein Beispiel dazu ist die Frage, wie das Repository Service-Versionen unterstützt und wie die Struktur im Repository hierfür genau aussehen muss.
Fachbücher und Forschungsarbeiten zum Thema SOA existieren in großer Zahl. Diese Arbeiten führen oft abstrakte Sichten auf eine SOA ein. Die Auswertung der von uns betrachteten Arbeiten hat dazu geführt, dass wir eine konzeptionelle Sicht auf die Funktionalitäten benötigen. Eine solche Sicht muss sowohl die einzelnen Komponenten betrachten als auch deren Interaktion.
Es existieren weitere Forschungskonzepte, wie zum Beispiel technische Services, die sich in einer ändernden Umgebung selber adaptieren [CKP06]. Um ein solches Konzept zu verwirklichen, müssen Teilkomponenten der IT-Infrastruktur, wie das Repository, erweitert werden. Diese Konzepte beschäftigen sich speziell mit der Verbesserung von Teilaspekten einer service-orientierten IT-Infrastruktur. Wir sehen dies daher als Stellhebel, einzelne Komponenten effizienter zu gestalten, die konzeptionelle Gesamtsicht wird hierdurch jedoch nicht verändert.

## 5 Zusammenfassung

Der Weg zu einer service-orientierten IT-Infrastruktur, um flexible Geschäftsprozessanwendungen zu unterstützen, ist aus vielen Gründen schwierig. Wir haben die Gründe hierfür diskutiert. Um den Erfolg einer Einführung dennoch zu gewährleisten, ist es zwingend notwendig, die eigene IT-Infrastruktur bewerten zu können. Für diese Bewertung wird eine Referenz benötigt, die einen Rahmen von Komponenten zur Realisierung von SOA-Funktionalität vorgibt.
Wir haben ein Rahmenwerk vorgestellt, welches in unterschiedlichen Ausbaustufen eine Referenz für eine service-orientierte IT-Infrastrukur definiert. Unternehmen sollen in der Lage sein, Geschäftsprozesse effizient umzusetzen. Deshalb beinhaltet das Rahmenwerk zahlreiche relevante Komponenten, z.B. zur Verwaltung von Services, Workflows und Geschäftsregeln. Hierdurch werden Unternehmen in die Lage versetzt, fehlende Komponenten oder Funktionalität zu identifizieren. Ferner wird durch die detaillierte Betrachtung der Komponenten in Unterstufen sichergestellt, dass auch die Abhängigkeiten zwischen den Komponenten bewertbar sind.
Wie in Kapitel 4 beschrieben, beschäftigen sich viele Gruppen mit dem Thema SOA. Hersteller bieten mächtige Tools an, die seit der Einführung von SOA einen ausgereiften Stand

erreicht haben. So deckt das Portfolio von IBM zum Beispiel einen großen Funktionsumfang ab. Was aktuell noch fehlt, ist eine konzeptionelle Sicht, die das Zusammenspiel der einzelnen Komponenten in einer IT-Infrastruktur betrachtet, um die eigene Unternehmenslandschaft geeignet migrieren zu können. Da auch die Mächtigkeit einzelner Komponenten unterschiedlicher Hersteller aktuell differiert, ist deren Kombination eine wichtige Anforderung. Dies ist aber aufgrund fehlender Richtlinien und Standards schwierig. Unser Rahmenwerk soll auch hier den Herstellern ermöglichen, die eigene Tool-Palette evaluieren zu können, um deren Funktionalität oder Integrationsfähigkeit zu erhöhen.
Ferner sehen wir eine Betrachtung weitergehender Konzepte, wie in Kapitel 4 beschrieben, noch in einem frühen Stadium. Auch hier soll unser Rahmenwerk als Basis dienen, vom dem ausgehend solche Konzepte bewertet werden können, um entsprechende Komponenten in Zukunft einsetzen und mit anderen Komponenten integrieren zu können.

## Literatur


[AAA+07] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C.K. Liu, V. Mehta, S. Thatte, P. Yendluri und A. Yiu. Web Services Business Process Execution Language Version 2.0 Standard. WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[AAC+08] S. Amnajmongkol, J. Angani, Y. Che, T. Fox, A. Lim und M. Keen. *Business Activity Monitoring with WebSphere Business Monitor V6.1, IBM Redbooks*, 2008.

[ACKM04] G. Alonso, F. Casati, H. Kuno und V. Machiraju. *Web Services Concepts, Architectures and Applications*. Springer-Verlag, Berlin, 2004.

[Bob08] R. Bobrik. *Konfigurierbare Visualisierung komplexer Prozessmodelle*. Dissertation, Universität Ulm, 2008.

[BRB07] R. Bobrik, M. Reichert und T. Bauer. View-based process visualization. In:. *Proc. BPM'07*, LNCS 4714:S. 88–95, 2007.

[Buc05] S. Buchwald. Konzeption und Realisierung einer Infrastruktur zur prozessorientierten Integration von Applikationen in Workflows. Diplomarbeit, Fachhochschule Ulm, 2005.

[BWRJ08] L. Bodenstaff, A. Wombacher, M. Reichert und M.C. Jaeger. Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In:. *Proc. SCC'08*, Seiten S. 21–29, 2008.

[CKP06] F. Cubera, B.J. Krämer und M.P. Papazoglou, Hrsg. *Service-Oriented Computing: A Research Roadmap, Number 05462*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[ELK+06] C. Emig, K. Langer, K. Krutz, S. Link, C. Momm und S. Abeck. The SOA's Layers, C+M Research Report, University of Karlsruhe. 2006.

[Erl05] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design, Book*. Prentice Hall Verlag, 2005.

[FN08] T. Freund und P. Niblett. ESB Interoperability Standards, Whitepaper. IBM Corporation, 2008.

[HR08] M. Herzog und A. Richter. SOA benötigt Governance - Eine virtuelle Diskussion zwischen IBM und SAP über das optimale Repository. Bericht, IBM SAP Competence Center Walldorf, IBM WebSphere Technical Sales BPM, 2008.

[IBM07] IBM. *Websphere Service Registry and Repository Handbook, IBM Redbook*, 2007.

[JBS97] S. Jablonski, N. Böhm und W. Schulze. *Workflow Management: Entwicklung von Anwendungen; Facetten einer neuen Technologie*. dpunkt-Verlag, 1997.

[Jos07] N.M. Josuttis. *SOA in Practice; The Art of Distributed System Design*. O'Reilly, 2007.

[MBL07] D. Maurer, P. Büch und M. Linke. Vom Geschäftsprozess zur Enterprise Architecture - ARIS Solution for Enterprise Architecture Management. Bericht, 2007.

[MLM+06] C.M. MacKenzie, K. Laskey, F. McCabe, P.F. Brown und R. Metz. *Reference Model for Service Oriented Architecture. OASIS Committee Specification 1*, August 2006.

[MRB08] B. Mutschler, M. Reichert und J. Bumiller. Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. *IEEE Computer Society Press, IEEE Transactions on Systems*, 38, 2008.

[Nat03] Y.V. Natis. Service-Oriented Architecture Scenario, Technical Report, Gartner Research ID Number: AV-19-6751. 2003.

[Pry05] R. Pryss. Enterprise Application Integration, Diplomarbeit. University of Ulm, 2005.

[RD00] M. Reichert und P. Dadam. Geschäftsprozessmodellierung und Workflow-Management Konzepte, Systeme und deren Anwendung. *GITO-Verlag, Industrie Management*, Vol. 3:S. 23–27, 2000.

[Rei00] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. Dissertation, Universität Ulm, Mai 2000.

[Rin04] S. Rinderle. *Schema Evolution in Process Management Systems*. Dissertation, Universität Ulm, 2004.

[RMR07] S. Rinderle-Ma und M. Reichert. A Formal Framework for Adaptive Access Control Models. *Journal of Data Semantics, Vol. IX*, LNCS 4601:S. 82–112, 2007.

[RMRD04] S. Rinderle-Ma, M. Reichert und P. Dadam. Flexible Support of Team Processes by Adaptive Workflow Systems. *Distributed and Parallel Databases*, 16(1):S. 91–116, 2004.

[RS04] M. Reichert und D. Stoll. Komposition, Choreographie und Orchestrierung von Web Services - Ein Überblick. *EMISA Forum*, 24(2):S. 21–32, 2004.

[Sch00] A. W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 2000.

[Ste08] S. Stein. Fachliche Servicebeschreibung am Beispiel eines virtuellen Autokonzerns, Expert Paper. Bericht, 2008.

[WRRM08] B. Weber, M. Reichert und S. Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. *Data and Knowledge Engineering*, 66(3):S. 438–466, 2008.

[YHK95] W. Yeong, T. Howes und S. Kille. Lightweight Directory Access Protocol. RFC 1777, Internet Engineering Task Force (IETF), Specification, 1995.

# Anforderungen an Datenbanksysteme für Multi-Tenancy- und Software-as-a-Service-Applikationen

Stefan Aulbach[‡] Dean Jacobs[$] Jürgen Primsch[$] Alfons Kemper[‡]

[‡]Technische Universität München, Garching
{stefan.aulbach, alfons.kemper}@in.tum.de

[$]SAP AG, Walldorf
{dean.jacobs, j.primsch}@sap.com

**Abstract:** Für Multi-Tenancy-Applikationen und Software as a Service (SaaS) stellen sich konventionelle Datenbanksysteme oftmals als ungeeignet heraus. In einem solchen Umfeld müssen Anforderungen, wie Erweiterbarkeit und Verfügbarkeit, trotz geringer Kosten für den Service Provider erfüllt werden. Dieser Beitrag spezifiziert Anforderungen an ein Datenbanksystem, das eigens für den Einsatz in Multi-Tenancy-Applikationen gedacht ist, und evaluiert, in welchem Umfang eine Umsetzung mittels aktuell verfügbarer Datenbanktechnik möglich ist. Weiterhin wird ein Modell eines Datenbanksystems präsentiert, das speziell für Multi-Tenancy und SaaS optimiert ist und konsequent auf Mandantenvirtualisierung, Grid-Computing-Infrastrukturen und Hauptspeicherlokalität setzt.

## 1 Einführung

Das Vertriebsmodell *Software as a Service* (SaaS) wird immer beliebter. Anstatt Applikationen im eigenen Haus zu betreiben und hierfür Hard- und Software sowie Personal bereithalten zu müssen, lagern viele Unternehmen die Bereitstellung an Service Provider aus. Für die Benutzer reichen dann ein Internetzugang und ein Browser aus, allerdings erwartet der Kunde auch, dass sich die Applikation so verhält, als ob sie im eigenen Haus installiert wäre. Der Service Provider hingegen will die *Total Cost of Ownership* (TCO) reduzieren, indem er seine Infrastruktur vielen Mandanten zur Verfügung stellt.

Die folgende Arbeit diskutiert genau diese Ansprüche und Anforderungen, die Kunden, Service Provider und Anwendungsentwickler an das zu Grunde liegende Datenbanksystem (DBMS) stellen. Setzt man für *Multi-Tenancy*, also für die Fähigkeit, mehrere Mandanten auf einem gemeinsamen System zu konzentrieren, marktübliche DBMS ein, so stellt man schnell fest, dass die dazu beworbenen Fähigkeiten nicht direkt im DBMS, sondern nur als Modul implementiert sind. Zwar werden die Anforderungen aus Sicht der Kunden mehr oder weniger erfüllt, jedoch existieren weiterhin grundlegende Probleme wie Isolation und Ressourcenverwaltung, die sich erst beim Einsatz der Anwendung in einem Data Center bemerkbar machen. Anwendungsentwickler stehen bei konventionellen DBMS ebenfalls vor Problemen: Soll zum Beispiel die Applikation erweiterbar und anpassbar sein, so muss dies innerhalb der Applikation implementiert werden, falls sich Mandanten eine gemeinsame Datenbankinstanz teilen. Dabei wird eine zusätzlich zu wartende Zwischenschicht notwendig, in der die eigentliche Datenverwaltung implementiert wird, und das DBMS

wird zu einem reinen Speichermedium degradiert, das die Semantik und die Beziehungen der Daten nicht mehr zur Optimierung und zur Konsistenthaltung heranziehen kann. Andere Implementierungen, bei denen Mandanten ihr eigenes Schema oder gar ihre eigene Instanz erhalten, scheiden aufgrund ihres hohen Speicherverbrauchs aus [JA07, AGJ+08].

Als Antwort auf die mangelnde Unterstützung von Multi-Tenancy in aktuellen DBMS stellen wir in dieser Arbeit ein Modell eines DBMS vor, das speziell an die Anforderungen von Multi-Tenancy-Applikation und SaaS angepasst ist. Einsatzgebiete sind hierbei OLTP- und OLAP-Applikationen mittlerer Komplexität, wie sie in typischen Business-Anwendungen vorkommen. Ein solcher Workload ist damit eher I/O- als CPU-lastig [LMP+08]. Wir haben uns deshalb an einem Zitat von Jim Gray orientiert, das optimale Hauptspeicherlokalität postuliert: *„Tape is Dead, Disk is Tape, Flash is Disk, RAM Locality is King“*

Der Beitrag ist wie folgt aufgebaut: Zunächst werden in Abschnitt 2 die Anforderungen an ein Multi-Tenancy-DBMS genauer erläutert, ehe in Abschnitt 3 beschrieben wird, in welchem Umfang sich aktuell am Markt erhältliche DBMS für die Umsetzung dieser Anforderungen eignen. Abschnitt 4 widmet sich der Behebung der übrigen Unzulänglichkeiten mittels einer neuen DBMS-Generation, bevor Abschnitt 5 schließlich die Ergebnisse zusammenfasst und einen Ausblick auf zukünftige Arbeiten gibt.

## 2 Anforderungen an Datenbanksysteme für Multi-Tenancy

Der Einsatz konventioneller DBMS ist auf den Betrieb innerhalb eines Unternehmens ausgerichtet, wo sich gewöhnlich nur die Daten und Anwendungen eines einzelnen Unternehmens oder einer Unternehmensgruppe finden. Bei Anwendungen, die als SaaS vermarktet werden, kommen zwangsläufig Technologien zum Einsatz, mit deren Hilfe auch die Datenbanken im Hintergrund für mehrere parallel agierende Mandanten verfügbar gemacht werden. Aus dem Vermarktungsmodell kombiniert mit Mehrmandantenfähigkeit ergeben sich spezielle Anforderungen, denen sich dieser Abschnitt widmet. Frühere Arbeiten [JA07, AGJ+08] zeigen, dass sich diejenigen Ansätze, bei denen sich mehrere Mandanten eine Instanz teilen, bezüglich des Ressourcenbedarfs besser positionieren als Ansätze, bei denen Mandanten eine eigene Datenbankinstanz erhalten.

Bei der Anforderungsdefinition lassen sich drei Rollen erkennen. Der *Service Provider*, der die Anwendung und die dafür notwendige Infrastruktur bereitstellt, legt besonderen Wert auf niedrige laufende Kosten bei hoher Verfügbarkeit. Der *Kunde* möchte im Wesentlichen eine hohe Verfügbarkeit bei hoher Datensicherheit, während sich der *Applikationsentwickler* einfache, aber mächtige Funktionsmodelle wünscht. Der Einfachheit halber verzichten wir bei der Aufstellung dieser Anforderungen auf die Trennung dieser drei Rollen.

**Mandantenkonsolidierung** ermöglicht dem Service Provider, mehrere Mandanten in einer Datenbankinstanz zu aggregieren. Bei der Konsolidierung darf weder die Anzahl, noch die Größe oder die Verteilung der Mandanten als limitierender Faktor auftreten. Das heißt insbesondere, dass es Mechanismen geben muss, einen Mandanten über mehrere Rechner zu verteilen, sollte dieser bis an die Grenzen eines einzelnen Rechners wachsen.

**Erweiterbarkeit der Kernapplikation** ermöglicht, die vom Service Provider angebote-

ne Applikation an unterschiedliche Geschäftsprozesse oder Spezialisierungen eines jeden Mandanten anzupassen. Der Mandant kann sowohl zu bestehenden Kern-Entitäten weitere Attribute, als auch gänzlich neue Entitäten definieren. Solche Erweiterungen müssen dann in der gesamten Anwendung berücksichtigt werden. Diese Anforderung ergibt sich aus der strengen Trennung zwischen Applikation und DBMS: die Applikation sorgt für die Datenpräsentation; das DBMS für die Datenhaltung und ist somit Eigentümer des Schemas.

**Isolation der Mandanten** ist erforderlich, um Zugriff nur auf die eigenen Daten des Mandanten zu gewähren. Aus diesem Grund muss das DBMS Mechanismen bereitstellen, um einen mandantenübergreifenden Zugriff von vornherein auszuschließen. Angelehnt an Zugriffsschutzmechanismen aus marktüblichen DBMS wird hier ein *Tenant Level Based Access Control* gefordert. Ein zentrales Identitätsmanagement verwaltet die Zugriffsrechte und verteilt diese in der Datenbank und den Applikationsservern.

Derzeit ist es üblich, dass Applikationsserver mit höchstmöglicher Autorisierung (also als Superuser) auf das DBMS zugreifen. Die eigentliche Autorisierung der Klienten erfolgt im Applikationsserver, so dass das Autorisierungssystem des DBMS gar nicht genutzt wird. Diese Vorgehensweise der Überprivilegierung des Applikationsservers birgt jedoch Gefahren, da ein erfolgreich kompromittierter Applikationsserver vollen Zugriff auf die sensiblen Daten aller Klienten in der Datenbank ermöglicht. Deshalb sollte man anwendungsspezifische und möglichst klientenspezifische so genannte *Autorisierungskorridore* [WEEK04] generieren, die den Anwendungen nur die benötigten Daten zugreifbar machen. Diese minimal autorisierten Datenbankkonten sollten automatisch aus den Anwendungen generiert werden.

**Ressourcenverwaltung und -überwachung** sind Voraussetzung für die bessere Auslastung von kritischen Ressourcen in einem DBMS, wie Hauptspeicher, CPU und verfügbare I/O-Bandbreite. In einem gewöhnlichen DBMS müssen diese Ressourcen zwischen den parallel abzuarbeitenden Anfragen verteilt werden; bei den Multi-Tenancy-DBMS findet zusätzlich noch eine Aufteilung auf Mandaten statt.

Ein Service Provider möchte seine physischen Ressourcen vollständig ausgenutzt wissen, weswegen das DBMS eine Methode zur Erfassung der aktuellen Lastsituation bereitstellen muss. Diese Erfassung muss so feingranular sein, dass sie sowohl die Lastsituation des Rechners, als auch den Ressourcenbedarf von mandantenspezifischen Tasks (z.B. Datenbankanfragen oder Operatoren im Auswertungsplan) erfassen kann und an eine zentrale Überwachungsinstanz oder gar einen Adaptiven Controller [GKS+08] melden kann.

Ein Adaptiver Controller steuert die Einhaltung von *Service Level Agreements (SLA)*, indem er Datenbankanfragen gemäß der aktuellen SLA-Einhaltung der einzelnen Mandanten priorisiert. Ein derartiges SLA-Management ermöglicht eine optimale Ressourcenausnutzung durch Überbuchung der Systeme bei gleichzeitiger Bereitstellung von *Quality of Service (QoS)*. Eine solche Überbuchung ist sinnvoll, da nicht immer alle Mandanten gleichzeitig online sind, beispielsweise wegen verschiedener Zeitzonen. Zudem müssen Amok laufende Anfragen vom DBMS erkannt und gestoppt werden können. Eine proaktive Ressourcenbeschränkung erweist sich möglicherweise als besseres Verfahren, da das Abbrechen von Anfragen sich auch negativ auf die Ressourcensituation auswirken kann.

**Lastverteilung und Migration von Mandanten** ist eng mit dem Ressourcenmanagement

verbunden. Die *Lastverteilung* dient der Abfederung von Lastspitzen und der Gewährleistung einer SLA-konformen Dienstverfügbarkeit. Solche Lastspitzen können sowohl periodisch (z.B. bei Buchungsabschlüssen), als auch sporadisch auftreten, etwa wegen einer Fehleinschätzung der Überbuchungssituation.

Beim *initialen Staging* eines Mandaten wird ein Rechner ausgewählt, wobei dieser entweder statisch beim Vertragsabschluss festgelegt, oder dynamisch bei der Aktivierung des Mandanten anhand der aktuellen Ressourcenauslastung ausgewählt wird. Eine solche *Aktivierung*, zusammen mit der *Passivierung*, ermöglicht es, dass mehrere Schichten auf dem System gefahren werden können. Nur solange ein Mandant aktiv ist, belegt er Ressourcen auf dem System, während in den Passivzeiten keinerlei Ressourcen belegt werden. Insbesondere sind die Metadaten der passiven Mandanten nicht geladen. Eine *statische Lastverteilung* allokiert die Mandanten aufgrund Lastvorhersagen, greift aber bei einer Änderung des Ressourcenbedarfs zur Laufzeit eines Mandanten nicht ein und passt nur die Lastvorhersagen entsprechend an. Erst bei der nächsten Aktivierung wird der Mandant auf einen besser geeigneten Rechner platziert. Im Gegensatz dazu kann eine *dynamische Lastverteilung* Mandanten in Überlast auch zur Laufzeit auf einen anderen Rechner migrieren.

Wächst der Bedarf innerhalb der Vertragslaufzeit, so reichen dem Mandanten oftmals die initial geplanten Kapazitäten nicht mehr aus. In einem solchen Fall muss dieser Mandant auf einen anderen Rechner *migriert* werden: entweder wieder auf ein von mehreren Mandanten gemeinsam genutztes System oder auf einen dedizierten Rechner.

**Schemaänderungen zur Laufzeit** treten häufig bei Updates einer Applikation auf. Da ein Multi-Tenancy-DBMS wegen der besseren Ausnutzung rund um die Uhr aktiv sein muss, müssen Schemaänderungen zur Laufzeit des Systems durchgeführt werden. Aber auch parallel agierende Mandanten machen eine Schemaänderung zur Laufzeit sehr wahrscheinlich, wenn ein Mandant die Anwendung nach seinen Bedürfnissen anpasst.

DDL-Anweisungen ziehen oft deutliche Leistungseinbußen mit sich, wenn sie auf einem aktiven System ausgeführt werden. Während das Anlegen von Tabellen nur eine Sperrung des Data Dictionaries erfordert, führen manche Änderungen an der Tabellenstruktur zu einer Änderung der Recordstruktur, also wie die Tupel auf dem Hintergrundspeicher liegen. Dementsprechend wird bei Strukturänderungen einer Tabelle oft eine temporäre Tabelle angelegt, die Tupel kopiert und anschließend die Tabelle umbenannt, womit ein solcher Eingriff die I/O-Leistung des DBMS beeinflusst. Daraus ergibt sich die Anforderung, jegliche Schemaänderungen ohne belastende Reorganisation durchführen zu können.

**Geringer Administrationsaufwand** sorgt bei einem Service Provider dafür, seine Leistung mit möglichst geringen laufenden Kosten, wie Hardwarekosten, Lizenzkosten, Infrastrukturkosten und Personalkosten, zu erbringen. Aus diesem Grund scheiden komplexe Cluster-Systeme als Infrastruktur aus, da nicht nur teure Spezialhardware notwendig wäre, sondern auch speziell geschulte Administratoren eingesetzt werden müssten, um den ungleich höheren Administrationsaufwand von Cluster-Systemen verglichen mit Shared-Nothing-Systemen gleicher Leistungsfähigkeit zu bewältigen.

**Verfügbarkeit** spielt eine so zentrale Rolle bei SaaS, dass SLAs oft integraler Bestandteil von Service-Verträgen sind. Damit ist der Service Provider fest an diese SLAs gebunden und muss bei einer Verletzung mit Vertragsstrafen rechnen. Ein Multi-Tenancy-DBMS

muss deshalb sicherstellen, dass die Applikation für den Mandanten gemäß der vereinbarten SLAs verfügbar ist, was bedeutet, dass die SLAs auch dann gewährleistet sein müssen, wenn Teile des Systems ausgefallen sind. Hierbei muss der Service Provider einen Kompromiss zwischen Kosten der Redundanz und den Vertragsstrafen eingehen, was sich im Grad der Verfügbarkeit widerspiegelt: Eine Non-Stop-Verfügbarkeit, bei der im Fehlerfall garantiert keine Transaktion zurückgerollt wird, erfordert viel zu hohe Investitionen, verglichen mit einer Hochverfügbarkeit, bei der Transaktionen zurückgerollt werden dürfen.

# 3 Verwendung aktueller Datenbanksysteme

Unsere früheren Arbeiten [AGJ+08] zeigen, dass sich die größten Probleme bei der Verwendung marktüblicher DBMS für Multi-Tenancy und SaaS aus der Verwaltung des Data Dictionaries und der Bufferpools ergeben. Bufferpools werden von den DBMS auf Tabellenbasis belegt, was bedeutet, dass bei einer sehr großen Anzahl von Tabellen die Bufferpool-Seiten nur eine sehr geringe Auslastung besitzen. Auf diese und weitere Probleme wird in diesem Abschnitt näher eingegangen.

## 3.1 Nutzung bereits vorhandener Techniken

Mit vorhandenen DBMS lassen sich auch ohne besondere Erweiterungen Multi-Tenancy-Applikationen implementieren, wenn auch nur mit Einschränkungen [JA07]: Eine Variante besteht darin, jedem Mandanten eine eigene Instanz auf einem Rechner zu geben, allerdings verbunden mit dem großen Nachteil, dass dabei mit jedem Mandanten ein sehr großer Overhead einhergeht, der aus dem initialen Ressourcenbedarf einer Datenbankinstanz resultiert. Jedem Mandanten ein eigenes Schema zu geben, führt innerhalb des DBMS zu einer sehr großen Anzahl von Tabellen, was sich in einem sehr hohen Hauptspeicherbedarf aufgrund wenig effizienter Metadatenverwaltung und in nur partiell gefüllten Bufferpool-Seiten niederschlägt. Eine dritte Variante besteht darin, alle Mandanten in einem gemeinsamen Schema zu konsolidieren und dann Sichten zu verwenden, um die Isolation der Daten zu ermöglichen. Bei dieser Variante müssen dann Maßnahmen ergriffen werden, um die Erweiterbarkeit zu gewährleisten.

Eine solche Maßnahme stellen *Chunk Tables* [AGJ+08] dar. Die Tabellen der Kernapplikation werden auf eigene physische Tabellen abgebildet, mandantenspezifischen Erweiterungen hingegen auf eine *generische Struktur*. Um eine Trennung der Mandanten zu erreichen, wird für jedes Tupel die Mandaten-ID gespeichert. Der große Vorteil dieses Verfahrens ist, dass sich die Anzahl der Tabellen reduziert und somit der Bufferpool besser ausgelastet ist. Nachteilig wirkt sich allerdings aus, dass ein einzelnes Tupel mittels Joins wieder zusammengesetzt werden muss. Auch andere Lösungen, wie pureXML für IBM DB2 [DB2], verfolgen diesen Ansatz.

Allen diesen Lösungen ist gemeinsam, dass die Mehrmandantenfähigkeit von außen aufgesetzt worden ist. Eine zusätzliche Zwischenschicht enthält nicht nur das Wissen über die verwendete Technologie, um die benutzerdefinierten Attribute auf die physischen Tabelle

abzubilden, sondern auch das Wissen über die Datenverteilung. Der Nachteil dieser Zwischenschicht ist ein höherer Wartungsaufwand und die Tatsache, dass die Datenbank zu einem großen Datentopf degradiert wird, der über die Verknüpfung der Daten nichts weiß, und diese somit nicht mehr zur Optimierung verwenden kann.

Neu aufkommende Datenbanktechnologien, wie Googles *BigTable* [CDG+08], Amazons *Dynamo* [DHJ+07] oder Yahoo!'s *PNUTS* [CRS+08], spielen hierbei auch eine wichtige Rolle, werden in dieser Arbeit jedoch in einem anderen Zusammenhang in einem folgenden Abschnitt näher diskutiert.

### 3.2 Geringfügige Modifikationen im Datenbankkernel

Während die geringe Bufferpool-Auslastung nur schwer in den Griff zu bekommen ist, kann man die Probleme mit der Metadatenverwaltung vergleichsweise einfach lösen. Jede einzelne Tabelle belegt einen gewissen Hauptspeicherbereich, in dem zum einen die Struktur der Tabelle vorgehalten wird, zum anderen die Abbildung von der logischen Sichtweise auf die physische Struktur (File Pointer, B-Tree, etc.) stattfindet. Gibt man jedem Mandanten ein eigenes Schema, so nimmt mit zunehmender Anzahl von Mandanten – und somit auch mit zunehmender Anzahl von Tabellen – der davon beanspruchte Arbeitsspeicherbereich zu. Eine Reduktion der Tabellenanzahl mittels eingangs erwähnter Verfahren führt allerdings zu keiner endgültigen Lösung, da mittels generischer Strukturen die Metadatenverwaltung vom DBMS nur in die Zwischenschicht ausgelagert wird.

Eine endgültige Lösung der Metadatenproblematik bietet letztlich nur eine geänderte Implementierung mit dem Ziel vererbbarer Metadaten, so dass möglichst viele Schemainformationen gemeinsam genutzt werden und somit nur einmal im System vorhanden sind. Das Schema der Kerntabellen kann als abstrakte Klasse aufgefasst werden; jeder Mandant bildet dann sein eigenes Schema durch Ableitung.

Um eine weitere Reduktion des Hauptspeicherbedarfs der Metadaten zu erreichen, sollte das Data Dictionary denselben Paging-Mechanismen unterliegen wie die gesamte Datenbank. Zudem sollte die Abbildung auf die File Pointer nur solange im Arbeitsspeicher gehalten werden, wie die Tabelle wirklich genutzt wird.

Weiterhin kann bei bestehenden DBMS auch die Informationsdarstellung auf dem Hintergrundspeicher so angepasst werden, dass Schemaänderungen keine Reorganisation erforderlich machen und somit keine I/O-Belastung auftritt. Dies geschieht vor allem durch Verfahren wie *Update in Place*, *Sort by Insertion* und *Delete in Place*. Eine mögliche Implementierung findet sich in [GT07].

Eine andere Modifikation besteht darin, die Daten und Indexe der Mandanten untereinander physisch zu isolieren, indem jeder Mandant seinen eigenen Container erhält. Dabei werden die Daten so verteilt, dass jeder Mandant in sich abgeschlossen ist, um sicherzustellen, dass eine Migration durch Verwendung von einfachen Dateioperationen anstelle von teurem Entladen und Laden möglich ist. Zudem kann der Container bei inaktiven Mandanten ohne Belastung anderer Mandanten gesichert werden. Für Indexe kann ein Verfahren eingesetzt werden, dass sich an partitionierte B-Bäume [Gra03] anlehnt.

### 3.3 Verbleibende Schwierigkeiten und inhärente Schwächen

Zu den verbleibenden Problemen zählen nahezu alle administrativen Aufgaben, die in konventionellen DBMS auf Instanzebene, in Multi-Tenancy-Systemen jedoch auf Mandantenebene implementiert sind. Solche administrativen Tätigkeiten sind beispielsweise Wiederherstellung der Anwendung nach Fehlbedienung (Point-in-Time-Recovery, PITR), aber auch Migration und (Hoch-) Verfügbarkeit, insbesondere auch Aktualisierungen der Applikation. Wird eine solche durchgeführt, so ändert sich oft auch das Schema der Tabellen. Dabei muss sichergestellt werden, dass Mandanten, die ihr privates Schema erweitert haben, auch mit der neuen Version der Kerntabellen kompatibel bleiben. In diesem Zusammenhang muss das Data Dictionary einer Versionierung unterworfen werden, so dass auch die Möglichkeit besteht, nicht alle Mandanten auf einmal auf die neue Version zu aktualisieren. Schemakompatibilität spielt zudem eine große Rolle in einem Multi-Tenancy Data Center mit unterschiedlichen Applikationsdomänen (z.B. SCM und CRM), die zwecks besserer Auslastung auf demselben operationalen System beheimatet sind.

Stonebraker et al. [SMA+07] fordern das Ende von den General Purpose-DBMS, indem sie spezialisierte DBMS für jeden Verwendungszweck postulieren. Da sich SaaS fundamental von klassischen OLTP-Anwendungen unterscheidet, ist eine solche Forderung auch in diesem Szenario durchaus gerechtfertigt.

## 4 Wege zum Glück

In diesem Abschnitt wird das Modell eines Multi-Tenancy-DBMS entworfen. Die Grundidee, auf denen alle hier vorgestellten Methoden aufbauen, ist der konsequente Einsatz von Virtualisierungstechniken und Grid-Technologien, sowie die Ausnutzung der von Jim Gray postulierten Hauptspeicherlokalität. In diesem Zusammenhang ist jeder Mandant eine „virtuelle Maschine“, die auf dem Hypervisor „Multi-Tenancy-fähiges DBMS“ zur Ausführung gebracht wird.

Speziell für den Einsatz von Grid-Technologien spricht die geringe Hardwarekomplexität der beteiligten Komponenten. Auch andere Projekte, wie Googles *BigTable* [CDG+08], verwenden aus gleicher Motivation diese Technologien. BigTable dient als hochgradig skalierbarer Datenspeicher, der auf einer Vielzahl einfacher Rechner verteilt wird, mit Verzicht auf ein relationales Datenmodell. BigTable speichert Informationen in einer Zelle, die über die Dimensionen Zeile, Spalte und Zeitstempel referenziert wird. Zellen werden so abgelegt, dass semantisch verwandte Dimensionen physikalisch möglichst benachbart platziert werden. Amazons *Dynamo* [DHJ+07] und *SimpleDB* [Sim] verfolgen einen ähnlichen Ansatz, gewähren allerdings andere Garantien hinsichtlich der Datenkonsistenz. Im Gegensatz zu diesen einfachen Key-Value-Stores stellt *PNUTS* [CRS+08] ein relationales Datenmodell zu Verfügung. Auch hier stehen Skalierbarkeit, Verteilung und Hochverfügbarkeit an erster Stelle, jedoch werden, um das zu erreichen, die Zugriffsmöglichkeiten und die Konsistenzgarantien verglichen mit traditionellen DBMS eingeschränkt.

Allen diesen Projekten ist gemeinsam, dass sie sehr stark von Verteilung und Replikation

Gebrauch machen und ihr Datenmodell danach ausrichten. Je nach den Anforderungen an die Replikation (beispielsweise, wie oft sie stattfindet) und damit verbunden auch an die Konsistenz der Replikate, wird das Datenmodell gewählt und die operationalen Garantien, wie man sie von konventionellen DBMS kennt, entweder gewährt oder eingeschränkt. Für Multi-Tenancy-DBMS können ähnliche Verfahren verwendet werden.

## 4.1 Flexible Datenbankschemas

Für die bessere Anpassbarkeit der SaaS-Applikation an die Bedürfnisse der einzelnen Mandanten, muss das DBMS Verfahren bereitstellen, die eine Änderung des Schemas zur Laufzeit erlauben. Flexible Schemas ermöglichen die *spontane Schemaanpassung* durch den Mandanten, ohne dass ein Datenbankadministrator des Service Providers notwendig ist. Hierfür werden zwei mögliche Ansätze vorgestellt. Zunächst sollen flexible Schemas mit Hilfe von bereits vorhandenen Technologien in modernen DBMS implementiert werden, ehe auf einen radikaleren Ansatz eingegangen wird, der umfangreiche Änderungen am Datenbankkernel erfordert.

### 4.1.1 Vorhandene Ansätze in aktuellen Datenbanksystemen

Einige moderne Datenbanksysteme besitzen die Fähigkeit, manche Schemaänderungen zur Laufzeit auch ohne Datenreorganisation durchzuführen, indem sie die notwendigen Änderungen zunächst nur auf den Metadaten ausführen. Demnach lassen sich Schemaänderungen in zwei unterschiedliche Typen aufteilen: auf der einen Seite befinden sich alle leichtgewichtigen Schemaänderungen, mit denen lediglich eine Anpassung des Data Dictionaries einhergeht. Zu diesem Typ gehören zum Beispiel das Anlegen einer neuen Tabelle oder das Hinzufügen von Spalten, die NULL-Werte enhalten darf. Auf der anderen Seite befinden sich die schwergewichtigen Operationen, die neben einer Änderung des Data Dictionaries auch eine Datenreorganisation erforderlich machen. Hierzu zählen die Änderung des Datentyps von Spalten oder das Hinzufügen von Attributen, deren Werte nicht undefiniert sein dürfen.

Aus diesen Fähigkeiten lässt sich eine mögliche Herangehensweise ableiten, mit deren Hilfe flexible Schemas implementiert werden können: alle unkritischen Operationen, d.h. alle diejenigen Operationen, die nur auf dem Data Dictionary arbeiten, werden unmittelbar durchgeführt, während Operationen, die eine Datenreorganisation, wie im Falle des neu hinzugefügen NOT-NULL-Attributs, oder eine Datenüberprüfung, falls der Datentyp eines Attributes geändert wird, erforderlich machen, erst zu einem späteren Zeitpunkt ausgeführt werden, an dem sich das System in einer Schwachlastphase befindet.

Darüber hinaus gibt es weitere Ansätze in modernen Datenbanksystemen, die speziell für flexible Schemas entwickelt worden sind. Beispielsweise beinhaltet Microsoft SQL Server 2008 eine Unterstützung für flexible Schemas, indem eine spezielle Speichermethode für dünn besiedelte Attribute (*Sparse Columns*) implementiert wird [ACGL+08]. Mittels IBM pureXML [pur] können auf ähnliche Weise flexible Schemas implementiert werden, indem man jedem Tupel einen Bereich für semistrukturierte Daten hinzufügt. Beide Varianten haben darüber hinaus den Effekt, dass sie Speicherplatz sparen, da der Aufwand für die Speicherung von NULL-Werten verringert wird.

### 4.1.2 *Tenant Swizzling* und *Working Set*-Modell

Anstatt Daten auf Pagelevel zu lesen, bietet es sich an, viel größere Datenblöcke auf einmal zu lesen. Der Umfang solcher Datenblöcke kann beispielsweise eine gesamte Buchungsperiode oder einen gesamten Mandanten umfassen. Für eine bessere Ressourcenauslastung empfiehlt es sich, die Datenrepräsentation im Arbeitsspeicher von der des Hintergrundspeichers zu entkoppeln. So ist gewährleistet, dass für jeden Speicherort die optimale Darstellung gewählt werden kann. Beim Transport der Daten vom oder zum Arbeitsspeicher müssen diese nun transformiert werden. Diese Transformation nennen wir *Tenant Swizzling*, angelehnt an das Pointer Swizzling in objekt-orientierten DBMS [KK95].

Auf die Objekte ist ein schneller Zugriff gewährleistet, solange diese nicht vom Hintergrundspeicher gelesen werden müssen. Ein solcher Zugriff wird nun – verglichen mit konventionellen DBMS – deutlich teurer, da die Daten erst vom Hintergrundspeicher gelesen, interpretiert und transformiert werden müssen. Ändert sich ein Objekt, so muss diese Transformation rückwärts durchlaufen werden, um das Objekt zu persistieren. Wird ein Write-Ahead-Log (WAL) geführt, ist eine solche Aktion jedoch nicht unmittelbar nach der Änderung erforderlich, sondern kann verzögert werden. Mit Tenant Swizzling können wegen der Entkopplung von Hauptspeicher- und Hintergrundspeicherrepräsentation ebenfalls flexible Schemas implementiert werden. Zudem kann die eingangs geforderte Hauptspeicherlokalität gewährleistet werden.

Kann aufgrund der Hauptspeichergröße nicht der gesamte Mandant in den Arbeitsspeicher geladen werden, dürfen nur diejenigen Teile geladen werden, die zum unmittelbaren *Working Set* des Mandanten gehören. Dieses ist je nach Applikation unterschiedlich und besteht beispielsweise aus der aktuellen Buchungsperiode. Wird ein Mandant aktiv, so werden die Daten des Working Sets mittels Tenant Swizzling vom Hintergrundspeicher geholt, interpretiert und online gebracht. Alle anderen Daten werden als archivierte Daten betrachtet und stehen nur noch lesend und mit erhöhter Zugriffszeit zur Verfügung. So wird gewährleistet, dass alle Daten eines Mandanten auf jeden Fall verfügbar sind und z.B. in analytischen Anfragen verwendet werden können. Sobald der Mandant deaktiviert wird, werden die Hauptspeicherstrukturen wieder persistiert.

Aus den Daten von Salesforce.com [McK07] lässt sich ableiten, dass selbst größere Mandanten selten mehr als 1 GB Daten in einer CRM-Applikation vorhalten, was eine sehr populäre, aber nicht die einzig mögliche Applikation für SaaS ist. Das hier vorgestellte Szenario erlaubt es demnach, eine respektable Anzahl von Mandanten auf ein entsprechend ausgestattetes System zu arrangieren, zumal das Working Set deutlich unter 1 GB liegen dürfte.

Aus Performancegründen kann weiterhin der Einsatz eines Bufferpools notwendig sein, wobei damit jedes Objekt doppelt im Hauptspeicher vorliegt, einmal objektorientiert und einmal im Hintergrundspeicherformat. Deshalb müssen beim Einsatz eines Bufferpools die Kosten für den erhöhten Hauptspeicherbedarf und die Vertragsstrafen bei nicht eingehaltenem SLA abgewogen werden. Einen solchen Ansatz verfolgt SAPs *liveCache*. Ist der Hauptspeicher ausreichend, um die Daten des gesamten Mandanten aufnehmen zu können, kann auf Bufferpools verzichtet werden.

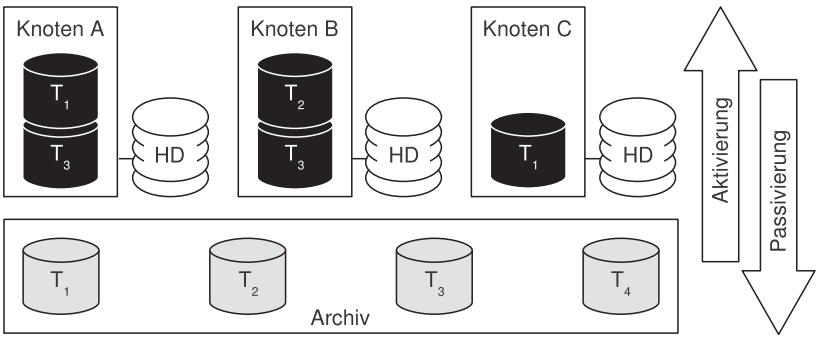



Abbildung 1: Multi-Tenancy DBMS

### 4.2 Grid-Infrastruktur und Near-Line-Storage

Ähnlich wie *Greenplum* [Gre] setzen wir auf eine Grid-Infrastruktur bestehend aus Knoten geringer Hardwarekomplexität. Gemäß Abbildung 1 bilden diese Knoten eine Farm von lose gekoppelten Shared-Nothing-Systemen, die Zugriff auf einen gemeinsamen Archivspeicher haben, dessen einzige Aufgabe es ist, passive Mandanten zu speichern; aktive Mandanten verwenden die lokalen Platten des Knotens. Für den Archivspeicher ist kein schnelles, und somit teures SAN erforderlich, ein Near-Line-Storage-System ist ausreichend. Ähnlich einem Cluster erhält man mit der Farm die Fähigkeiten zu Überbuchbarkeit und horizontalen Skalierbarkeit, bei gleichzeitig reduziertem Administrationsaufwand.

Zwecks besserer Verfügbarkeit wird eine Master/Slave-Replikation auf Mandantenebene eingesetzt, was nicht notwendigerweise bedeutet, dass zwei auf demselben Master aktive Mandanten auch auf demselben Slave allokiert sind. Da man aus Performancegründen auf Random-I/O auf der Logdatei verzichten möchte, darf nicht für jeden Mandanten eine eigene Logdatei geführt werden, sondern nur genau eine pro Knoten. Für die Replikation auf Mandantenebene muss nun ein *partielles Log Shipping* implementiert werden, das nur die Logeinträge des jeweiligen Mandanten an die Replikate weiterleitet. Da gemäß der eingangs aufgestellten Anforderungen eine Transaktion niemals die Grenzen von Mandanten überschreiten kann, ist ein solches partielles Log Shipping möglich, ohne dass benachbarte Mandanten davon betroffen sind. Zudem befinden sich bei der Verwendung von einem Container pro Mandant, wie es in Abschnitt 3.2 beschrieben ist, niemals Daten von verschiedenen Mandanten auf einer gemeinsamen Speicherseite. Ob synchrones oder asynchrones Log Shipping zum Einsatz kommt, hängt von dem mit dem Mandanten vereinbarten SLA, aber auch von der räumlichen Verteilung der Replikate ab: Innerhalb eines Data Centers erfolgt synchrone Replikation, während Replikation für Desaster Recovery in ein entferntes Data Center besser asynchron erfolgt.

Bei der Aktivierung eines Mandanten werden der Master und ein oder mehrere Slaves instantiiert, indem der Datenbestand vom Archivspeicher geladen wird. Soll zur Laufzeit ein zusätzlicher Slave gestartet werden, so lädt der Knoten den – mittlerweile veralteten Datenbestand – vom zentralen Archivspeicher und macht einen Roll-Forward mit Hilfe der Logdaten des Masters. Dies setzt voraus, dass eines der beteiligten Replikate einen

Logdatensatz bereithält, der sich auf die im zentralen Archivspeicher abgelegte Datenausprägung bezieht. Fällt ein Knoten der Farm aus, so muss sichergestellt werden, dass alle Mandanten, für die der ausgefallene Knoten die Masterrolle übernommen hat, wieder einen eindeutigen Master zugewiesen bekommen. Beim Recovery nach einem Fehlerfall kann auch die Kopie auf dem gemeinsam genutzten Archivspeicher genutzt werden.

Solange ein Mandant passiv ist, kann dieser beliebigen Knoten zugewiesen werden, so dass eine statische Lastverteilung ohne großen Aufwand realisiert werden kann. Ist der Mandant hingegen aktiv, so wird die dynamische Lastverteilung dadurch realisiert, dass der überlastete Master seine Führungsrolle an einen der beteiligten Slaves abgibt. Da ein Slave keine Anfragelast zu schultern hat, kann damit die Last feingranular auf Mandantenebene verteilt werden. Allerdings schränkt dieses Verfahren die Wahl des Zielknotens auf diejenigen Knoten ein, die ein Replikat des betroffenen Mandanten besitzen. Sollte die Umschaltung der Rollen nicht den gewünschten Effekt erzielen, so bietet sich an, ein Replikat auf einem neuen Slave zu erstellen und diesen dann als Master zu wählen. Anschließend kann die Anzahl der vorgehaltenen Kopien wieder reduziert werden. Mit dieser Erweiterung kann eine Migration auf einen neuen Knoten zur Laufzeit durchgeführt werden. Ebenfalls können der Farm neu hinzugefügte Knoten sofort genutzt werden.

## 5 Zusammenfassung und Ausblick

Mit dieser Arbeit wurde ein Anforderungskatalog für den Einsatz von DBMS in Multi-Tenancy-Anwendungen und SaaS skizziert. Einige der Forderungen lassen sich mit unmodifizierten konventionellen DBMS durch den Einsatz von generischen Strukturen, wie beispielsweise Chunk Tables, bereits jetzt realisieren. Einige unserer Arbeiten beschäftigen sich mit der Speicherung dieser generischen Strukturen mittels verschiedener Speichermethoden, wie konventionelle Tabellen, Chunk Tables, XML und dünn belegte Tabellen.

Andere Anforderungen stehen erst mit einem modifizierten Datenbankkernels bereit, wie beispielsweise Schemaänderungen zur Laufzeit. Einige der in Abschnitt 3.2 genannten Modifikationen wurden bereits in MaxDB 7.7 [Maxb] implementiert [Maxa].

Ein großes Forschungsvorhaben ist die Umsetzung eines DBMS, wie es in Abschnitt 4 vorgestellt worden ist. Dabei wird insbesondere ein Augenmerk auf die Administration der Farm mittels eines Adaptiven Controllers gelegt, der den Zustand aller Knoten und aller Mandanten überwacht und auf die Einhaltung vereinbarter SLAs achtet. Neben proaktiver Aktivierung eines Mandanten spielen auch Migration und Lastverteilung eine große Rolle. Der adaptive Controller soll zudem in der Lage sein, die (Hoch-) Verfügbarkeit der Mandantendaten in Einklang mit den SLAs zu garantieren.

Weitere Fragestellungen beschäftigen sich mit dem Fall, dass ein Mandant die Grenzen einer einzelnen Maschine sprengt. Anstatt die Daten zu partitionieren und somit ein synchrones System zu erhalten, könnte durchaus auch der Einsatz von relaxierten Konsistenzmodellen, wie es beispielsweise *PNUTS* [CRS+08] bietet, gerechtfertigt sein. In einem solchen asynchronen System geht man das Risiko von inkonsistenten Daten ein, und hofft darauf, dass diese Inkonsistenzen behoben werden können.

## Literatur


[ACGL+08] Srini Acharya, Peter Carlin, César A. Galindo-Legaria, Krzysztof Kozielczyk, Pawel Terlecki und Peter Zabback. Relational Support for Flexible Schema Scenarios. In *VLDB*, Seiten 1289–1300, 2008.

[AGJ+08] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper und Jan Rittinger. Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques. In *SIGMOD Conference*, Seiten 1195–1206, 2008.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes und Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[CRS+08] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver und Ramana Yerneni. PNUTS: Yahoo!'s Hosted Data Serving Platform. In *VLDB*, Seiten 1277–1288, 2008.

[DB2] Data Integration and Composite Business Services, Part 3: Build a Multi-Tenant Data Tier with Access Control and Security. `http://www.ibm.com/developerworks/db2/library/techarticle/dm-0712taylor/`.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall und Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *SOSP*, Seiten 205–220, 2007.

[GKS+08] Daniel Gmach, Stefan Krompass, Andreas Scholz, Martin Wimmer und Alfons Kemper. Adaptive Quality of Service Management for Enterprise Services. *TWEB*, 2(1), 2008.

[Gra03] Goetz Graefe. Sorting And Indexing With Partitioned B-Trees. In *CIDR*, 2003.

[Gre] Greenplum. `http://www.greenplum.com/`.

[GT07] Heike Gursch und Werner Thesing. No-Reorganization Principle – Data Storage Without I/O Bottlenecks. `http://maxdb.sap.com/training/internals_7.6/reorgfree_EN_76.pdf`, 2007.

[JA07] Dean Jacobs und Stefan Aulbach. Ruminations on Multi-Tenant Databases. In *BTW*, Seiten 514–521, 2007.

[KK95] Alfons Kemper und Donald Kossmann. Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis. *VLDB J.*, 4(3):519–566, 1995.

[LMP+08] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim und Sang-Woo Kim. A Case for Flash Memory SSD in Enterprise Database Applications. In *SIGMOD Conference*, Seiten 1075–1086, 2008.

[Maxa] MaxDB – Internals. `http://maxdb.sap.com/training/#internals`.

[Maxb] MaxDB – The SAP Database System. `http://maxdb.sap.com/`.

[McK07] Todd McKinnon. Plug Your Code in Here – An Internet Application Platform. `www.hpts.ws/papers/2007/hpts_conference_oct_2007.ppt`, 2007.

[pur] XML Database - DB2 pureXML. `http://www.ibm.com/db2/xml/`.

[Sim] Amazon SimpleDB. `http://aws.amazon.com/simpledb/`.

[SMA+07] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem und Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). In *VLDB*, Seiten 1150–1160, 2007.

[WEEK04] Martin Wimmer, Daniela Eberhardt, Pia Ehrnlechner und Alfons Kemper. Reliable and Adaptable Security Engineering for Database-Web Services. In *ICWE*, Seiten 502–515, 2004.

# Die Migration von Hibernate nach OpenJPA: Ein Erfahrungsbericht

Uwe Hohenstein, Michael C. Jäger

Corporate Research and Technologies, Siemens AG
Software & Engineering, Architecture (CT SE2)
Otto-Hahn-Ring 6
D-81730 München
{uwe.hohenstein | michael.c.jaeger}@siemens.com

**Abstract:** Persistenzframeworks sind komplexe Softwarelösungen für die Speicherung und das Auffinden von Objekten in relationalen Datenbanken. Immer wichtiger für die Weiterentwicklung solcher Frameworks werden Erfahrungen, die mit ihrem Einsatz gesammelt werden. Aus diesem Grund soll der vorliegende Beitrag die Architektur eines hochverfügbaren Systems mit firm-realtime Anforderungen vorstellen und Erfahrungen erläutern, die bei der Migration eines zugrunde liegenden Persistenzframeworks gesammelt wurden. Die Migration betrifft eine Telekommunikationsmiddleware, welche zunächst das weit verbreitete objekt/relationale Persistenzframework Hibernate eingesetzt hat. Da Hibernate mit einer Patentklage in den USA konfrontiert wurde, entstand eine negative Auswirkung auf jegliche Produkte, die Hibernate enthalten und im US-Markt ausgeliefert werden. Aus diesem Grund wurde der Austausch von Hibernate durch das Alternativprodukt OpenJPA für die Telekommunikationsmiddleware forciert.
Diese Arbeit stellt die Migration von Hibernate nach OpenJPA vor, nennt die wichtigsten Herausforderungen sowie deren Lösungen und zeigt dabei, dass trotz konzeptioneller Ähnlichkeiten bedeutende Unterschiede zwischen Hibernate und OpenJPA in der Verwendung bestehen, welche teilweise den Erfolg der Migration in Frage stellten.

## 1 Einleitung

Persistenzframeworks vereinfachen den Zugriff aus einer objektorientierten Programmiersprache wie Java auf ein relationales Datenbanksystem (DBS). Sie stellen Werkzeuge für die persistente Speicherung von objektorientierten Daten in relationalen Tabellen mittels einer objekt/relationalen (O/R) Abbildung dar. Ein Persistenzframework stellt Entwicklern eine objektorientierte Programmierschnittstelle (Application Programming Interface, API) zur Verfügung, mit der dieser Objekte speichern kann. Die Extraktion der Daten aus dem Objekt und die Kommunikation mit der Datenbank übernimmt das Persistenzframework. Umgekehrt erlaubt das Framework die Abfrage von Objekten mit einer objektobjektorientierten API, ohne SQL formulieren zu müssen.

Insbesondere das Java-basierte Persistenzframework Hibernate [Hib] hat in der Vergangenheit in der Java-Szene zunehmend an Beliebtheit gewonnen [Be05]. Der Reiz von Hibernate liegt sicherlich darin, dass es ein frei verfügbares Open-Source-Projekt ist, das

eine Vielzahl von DBSen unterstützt. Neben den etablierten Produkten wie DB2, Oracle, Sybase oder SQL Server kann Hibernate auch mit den üblichen Open-Source-Produkten wie PostgreSQL oder MySQL und auch weniger bekannten Systemen wie Interbase problemlos arbeiten. Ein weiterer Grund für die Beliebtheit sind sicherlich die geringen Performanzeinbussen gegenüber einer direkten Datenbankanbindung mit JDBC. Aus technischer Sicht kommt hinzu, dass Hibernate erlaubt, hochgradig Einfluss auf das Verhalten und die Performanz nehmen zu können. Eine mangelnde Einflussnahme wird häufig beim konkurrierenden Standard Java Data Objects [JDO] und entsprechenden Produkten bemängelt [MR02,BaS05].

Mitte des Jahres 2006 wurde durch das Unternehmen Firestar eine Patentverletzung von Red Hat, dem Lieferanten von Hibernate, in den Vereinigten Staaten beklagt: Firestar hält ein Patent auf O/R-Abbildungen unter Benutzung einer Datei, in der die Beziehungen zwischen Objekten und Tabelleneinträgen definiert sind (vgl. [HOD+00]). Auch wenn dieses Patent technisch gesehen aufgrund bekannter vorangegangener Arbeiten (z.B. durch das Enterprise Objects Framework von NeXT aus dem Jahre 1994 [Ne94]) nicht tragfähig scheint, musste erwartet werden, dass auch bei klaren technischen Verhältnissen die Rechtssprechung unerwartete Entscheidungen treffen kann. Was zunächst als ein Problem für Red Hat erscheint, entpuppt sich als größeres Problem: Jede Software, die mit Hibernate in die USA ausgeliefert wird, ist von der Patentklage ebenfalls erfasst, da die Redistribution die Rolle eines Lieferanten impliziert.

Somit wurde dies auch zu einem Problem für Siemens Enterprise Communications (SEN), welches Hibernate in deren Middleware namens OpenSOA [Str07] integriert. OpenSOA implementiert eine Plattform für Dienste, mit denen eine Service-orientierte Architektur (SOA) bei der Entwicklung von Anwendungen im Kommunikationsumfeld realisiert wird. Die Plattform bündelt Dienste und Funktionen, die Gegenstand jeglicher Kommunikationsanwendungen sind, und ermöglicht durch diese Form der Wiederverwendung eine effiziente Entwicklung von Anwendungen. Um die Bewegungsfreiheit von SEN auf dem amerikanischen Markt zu erhalten bzw. auszubauen und um Produkte dem Kunden weiterhin ohne Risiken anzubieten, entschloss sich SEN, Hibernate durch ein anderes Persistenzframework zu ersetzen. Neben der Patenklage bestand ein weiterer Grund in den kritischen Bedingungen zur verwendeten GNU Lesser General Public License (LGPL).

Aus diesen rechtlichen Rahmenbedingungen entstand nun der Bedarf für die Migration. Die vorliegende Arbeit erläutert dieses Vorhaben und liefert dabei Einsichten über den tatsächlichen Grad der Kapselung einer Datenbank unter Verwendung eines Persistenzframeworks. Der nächste Abschnitt 2 stellt das OpenSOA-Projekt vor und liefert die technischen Hintergründe für die Migration. Abschnitt 3 geht auf die Persistenzarchitektur von OpenSOA ein. Das generelle Vorgehen bei der Migration aus organisatorischer Sicht wird in Abschnitt 4 erläutert, während Abschnitt 5 wichtige Aspekte der Migration aus technischer Sicht vorstellt. Abschnitt 6 präsentiert dann eine Auswahl von speziellen Problemen, die zu Beginn der Migration nicht erwartet wurden; diese Probleme machen deutlich, wie weitreichend der Austausch eines Persistenzframeworks sein kann. Analog dazu erläutert Abschnitt 7 die dazugehörigen Lösungen. Die Arbeit endet mit einer Zusammenfassung der zusammengetragenen Erkenntnisse.

## 2 Die OpenSOA-Software

OpenSOA umfasst eine Dienstplattform, die auf Basis einer OSGi-Laufzeitumgebung [OSGi] implementiert ist. Da die OSGi-Spezifikation nicht alle Funktionen einer Service-orientierten Architektur abdeckt, wurden proprietäre Erweiterungen erstellt oder bereits erhältliche Software hinzugenommen. Besonderes Augenmerk wurde hierbei auf die Anforderungen an Performanz und Skalierbarkeit für Kommunikationsanwendungen gelegt. Hierbei dient ein OGSi-Container als leichtgewichtige Basis. Um darauf basierend beispielsweise Replikationstransparenz in einem Verbund von Computern herzustellen, wurde ein zu OSGi konformer Life Cycle Manager entwickelt, der den Zugriff auf Komponenten über mehrere Container hinweg innerhalb des Verbunds ermöglicht („Distributed OSGi"). Weitere Ergänzungen umfassen beispielsweise eine Infrastruktur für Nachrichten, Fehlermanagement und ein Persistenzsystem. Letzteres wird gesondert im folgenden Kapitel behandelt.

OpenSOA realisiert in einer Service-orientierten Architektur (SOA) Basisdienste, die genereller Bestandteil von Kommunikationsanwendungen sein können. Beispiele hierfür sind eine Benutzerverwaltung, die Erfassung von Präsenszeiten usw. Kommunikationsanwendungen können ein E-Mail Client, eine Web-Anwendung, die dem Benutzer die Verwaltung des eigenen Telefons ermöglicht, oder auch eine Administrationsumgebung für PBX-Vermittlungssysteme (Private Branch Exchange) sein. Die implementierte Architektur gliedert sich in folgende Schichten (Abbildung 1):

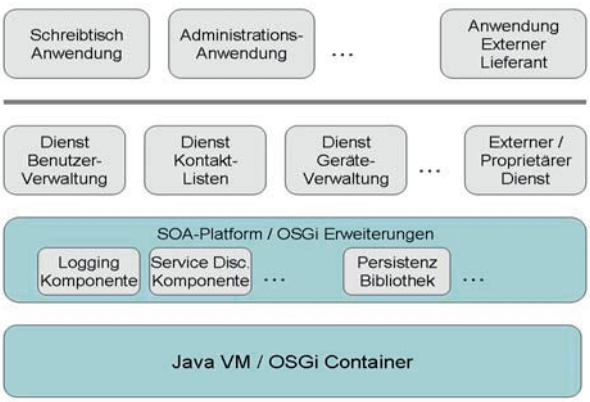



Abbildung 1: Überblick Architektur von OpenSOA

Auf Basis einer Java-Laufzeitumgebung kommt ein OSGi-Container zum Einsatz. Dieser wird durch weitere Software, proprietär oder kommerziell erhältlich, zu einer SOA-Plattform ergänzt. Auf dieser SOA-Plattform werden Dienste bereitgestellt, die von einer oder mehreren Kommunikationsanwendungen genutzt werden.

Für die Entwicklung von Anwendungen bietet OpenSOA den Vorteil der Bereitstellung einer Produkt-Plattform, die die Implementierung von Kommunikationsanwendungen durch Wiederverwendung dieser Basisfunktionalität erleichtert und effizienter gestaltet.

Auf der Seite der Anwendung ist darüber hinaus nicht nur die Wiederverwendung im Rahmen der Entwicklung von eigenen Produkten geplant – auch Anwendungen fremder Hersteller sollen Basisdienste nutzen können, um zusätzlich eigene Dienste auf der Plattform bereitzustellen.

Vom Umfang her ist die Implementierung der SOA, also die Erweiterungen zur derzeit existierenden OSGi-Implementierung, deutlich weniger umfangreich als beispielsweise eine Laufzeitumgebung, welche die JEE-Spezifikation implementiert. Betrachtet man zusätzlich die Basisdienste, die Standardfunktionen für Kommunikationsanwendungen implementieren, ergibt sich von der Anzahl der Java-Klassen her ein ähnlicher Umfang wie bei gängigen JEE-Implementierungen (ca. 6000 bis 7000 Klassen bei OpenSOA, Der JBoss AS 4 von Red Hat umfasst ca. 6300 Klassen).

## 3 Architektur der Persistenzsystems in OpenSOA

Der vorangegangene Überblick über die Architektur von OpenSOA ist nicht auf das Persistenzsystem eingegangen, das in klassischen 3-Schichten Architekturen eine der drei Schichten konstituiert. Für einen Dienst auf OpenSOA ist es nicht zwingend notwendig, Daten persistent zu speichern. Viele Basisdienste nutzen allerdings das Persistenzsystem zum Beispiel für die Verwaltung von Kontaktlisten oder um die Kontakte und Telefonnummern eines Teilnehmers speichern. OpenSOA und dessen Dienste sind in der objektorientierten Programmiersprache Java implementiert und als DBS kommen relationale Systeme zum Einsatz. Wie eingangs erwähnt, wird bei OpenSOA das Persistenzframework Hibernate für die Kopplung verwendet. Zudem wird ein Connection Pool verwendet, da die Verwaltung von parallel laufenden Verbindungen zur Datenbank erwartet wird. Daher kommuniziert das Persistenzframework nicht direkt mit dem Datenbanktreiber, sondern mit einem Connection Pool. In einer OpenSOA-Installation werden ca. 200-300 gleichzeitige Verbindungen verwaltet.

Ein Persistenzframework deckt viele Aspekte ab und mittlerweile kann dessen Implementierung als intuitiv komplex erachtet werden. Dementsprechend hat der Entwickler viele Möglichkeiten einen bestimmten Sachverhalt zu realisieren. Darüber hinaus bietet es viele Konfigurationsmöglichkeiten, um auf spezielle Anforderungen für Performanz oder Integrität der Daten Einfluss zu nehmen. Um diese Punkte für die Entwicklung von Diensten auf OpenSOA zu standardisieren, wurde eine Bibliothek implementiert, die im Wesentlichen zwei Bereiche abdeckt:

1. Eine Art Programmier-Template erledigt das Anfordern einer Datenbankverbindung, die Aufrufe für das Öffnen und Schließen einer Transaktion, das Verhalten bei Ausnahmefällen während der Anfragebearbeitung oder eine Wiederholung der Anfrage bei Verbindungsproblemen.
2. Des Weiteren implementiert die Bibliothek ein zentralisiertes Vorgehen zur Konfiguration des Persistenzframeworks. Auf diese Weise kann die Installation der Software leichter automatisiert werden.

Insgesamt besteht das gesamte Persistenzsystem, auf das der Entwickler eines OpenSOA Dienstes zugreift, aus mehreren Schichten, die sich wie folgt anordnen:

1. Eine **Template-Bibliothek** für standardisierte Nutzung der Abbildungssoftware und einheitliche Konfiguration: Diese Bibliothek fasst wiederkehrende Aufgaben wie Session- und Transaktions- wie auch Fehlerbehandlung zusammen uns ist ein proprietärer Teil von OpenSOA ist somit eine Art API für den Entwickler.
2. Die **Hibernate Software**, ein Persistenzframework, welches die Abbildungsfunktion implementiert: Dieses Softwareprodukt wurde durch die Template-Bibliothek gekapselt. Zur Programmierung der Persistenzlogik wird Hibernate innerhalb der Templateaufrufe benutzt, also um Daten zu speichern oder abzurufen.
3. Ein **Connection Pool**, der die Verbindungen verwaltet, die durch den JDBC Treiber zur Datenbank aufgebaut werden, wird durch Hibernate (bzw. OpenJPA) genutzt. Grundsätzlich hat der Entwickler keinen Kontakt zu dieser Schicht.
4. Ein **JDBC-Treiber**, der die Kommunikation mit der Datenbank über eine Java-API ermöglicht. Auch wenn ein Entwickler primär mit der Template-Bibliothek und Hibernate arbeitet, kann ein direkter Zugriff auf JDBC für spezielle Aufgaben, die Hibernate nicht anbietet wie z.B. Anfragen mit Optimizer Hints, nötig sein.
5. Das **Datenbanksystem:** Als DBSe werden MySQL, PostgreSQL und SOLID unterstützt. SOLID besitzt hierbei die besondere Rolle, dass es eine Hot Standby Installation unterstützt, die bereits von Siemens PBX-Lösungen genutzt wird. In OpenSOA wie auch in anderen Java Anwendungen erfolgt die Interaktion mit dem DBS grundsätzlich über den JDBC Treiber.

Abbildung 2 fasst die Bestandteile der Persistenz-Architektur graphisch zusammen. Die grauen Pfeile in der Abbildung deuten an, über welche Schichten der Architektur ein Dienst auf andere Teile der SOA-Infrastruktur in OpenSOA zugreift. Diese Architektur ist damit auch Ausgangspunkt der Ersetzung von Hibernate durch OpenJPA.

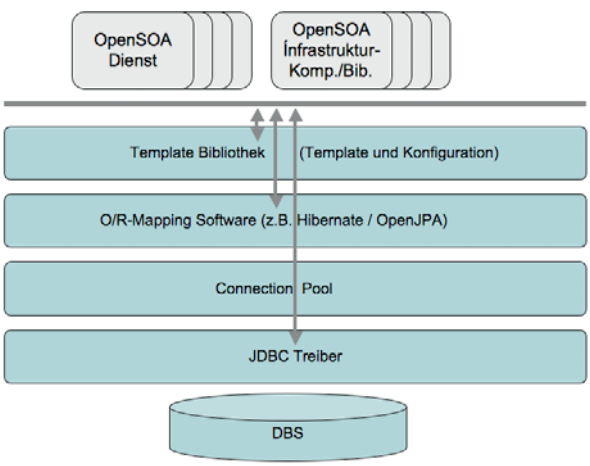



Abbildung 2: Architektur des Persistenzsystems in OpenSOA

## 4 Allgemeine Ersetzungsstrategie

Nachdem die Migration beschlossen wurde, erfolgte das Vorgehen nach einem iterativen, heuristischen Phasenmodell, das sich in folgende sechs Aktivitäten gliedert:

**1. Projektdefinition:** Im Rahmen der Projektdefinition wurde zügig ein geeigneter Ersatz für Hibernate ermittelt. Grundsätzlich existiert eine große Auswahl verschiedener Frameworks für die O/R-Abbildung, die auch für den Einsatz in OpenSOA geeignet wären. Die Überlegungen, die zu einer schnellen Entscheidung führten, sind im folgenden Abschnitt dargelegt. Zusammenfassend ist die Wahl auf OpenJPA gefallen, welches aus dem Produkt Kodo von BEA Systems hervorgegangen ist.

**2. Grobplanung:** Nachdem der Entscheidung für ein Produkt, wurden im Detail die kritischen Punkte zusammengestellt, die durch ein Proof-of-Concept zu verifizieren sind. Es musste auch bedacht werden, dass JPA-Implementierungen fehlerhaft oder unzureichend dokumentiert sind, da der Standard JPA wie auch das Produkt OpenJPA in der Version 0.9.7 zum damaligen Zeitpunkt relativ neu waren. Dadurch gliederte sich die Grobplanung in zwei Teile: Zum einen wurde auf Basis der Produktauswahl das generelle Vorgehen definiert. Zum anderen mussten die kritischen Punkte systematisch zusammengetragen werden, um diese möglichst frühzeitig ausräumen zu können bzw. im negativen Fall den Schwenk auf eine andere Abbildungssoftware zu ermöglichen.

**3. Testmigration:** Anschließend wurde die zentrale Template-Bibliothek in Verbindung mit dem umfangreichsten OpenSOA-Dienst umgesetzt. Die Testmigration diente zur Überprüfung der zuvor als kritisch ermittelten Punkte und ging auch darüber hinaus: 1) Um die Umstellung auf OpenSOA in der Vorbereitung möglichst umfangreich abzusichern, sollte OpenJPA über die kritischen Punkte hinaus evaluiert werden. Dies wurde mit der Testmigration erfüllt: Es stellten sich neue Probleme heraus, die bei der Planung nicht bedacht wurden. 2) Ein weiterer Vorteil lag darin, dass ein Überblick darüber gewonnen werden konnte, welche Arbeiten im Detail zu bewerkstelligen sind. Dadurch wurde die Feinplanung der Umstellung präziser und die Automatisierung möglicher Schritte konnte evaluiert werden. 3) Schließlich wurde mit der Umstellung der Template-Bibliothek ein wichtiger Teil der Migration bereits geleistet, da im günstigen Fall die einzelnen Dienste und Systembestandteile das Persistenzsystem mittels der Template-Bibliothek nutzen.

**4. Feinplanung:** Auf Basis der vorangegangen Schritte wurde die Feinplanung der Migration durchgeführt. Dieser Schritt kann als heuristische Iteration zum zweiten Schritt gesehen werden. Wie eingangs erwähnt, stellt OpenSOA eine umfangreiche Software dar, mit der Folge, dass die einzelnen Bestandteile nicht zu einem Stichtag umgestellt werden konnten. Parallel dazu vollzog sich die Weiterentwicklung von OpenSOA, so dass bei einer Umstellung von ein bis zwei Wochen pro Subsystem die Arbeit der Entwickler hätte für Wochen unterbrochen werden müssen, es sei denn, man würde massiv auseinander gelaufene Versionen der Quelltexte später manuell zusammen führen, was sehr aufwändig wäre. Daher musste ein Plan erstellt werden, nach dem die Dienste schrittweise umzustellen waren. Dieser Plan resultierte aus den Abhängigkeiten der einzelnen Subsysteme untereinander. Darüber hinaus musste das Vorgehen mit den betroffenen Entwicklern kommuniziert werden.

**5. Migration:** Anschließend erfolgte die Migration schrittweise. Um die „normale“

Entwicklungsarbeit nicht zu unterbrechen, konnten nicht alle Komponenten zu einem singulären Termin umgestellt werden. Somit wurde in Kauf genommen, dass Entwicklungszwischenstände von OpenSOA eine gemischte Umgebung, teils auf OpenJPA, teils auf Hibernate basierend, nutzten. In dieser Zeit liefen Dienste in der OGSi-Laufzeitumgebung parallel jeweils mit Hibernate und OpenJPA.

**6. Verifikation:** Zur Verifikation dienten die verschieden gestuften Softwaretests im Entwicklungsprojekt von OpenSOA. Dies sind Unit-Tests auf Ebene von Klassen, eine sogenannte Functional Test Suite, die bestimmte Use Cases implementiert, sowie Integrationstests, bei denen OpenSOA mit den darauf entwickelten Anwendungen kombiniert und getestet wird. Darüber hinaus existierten Performanztests, die unterschiedliche Anforderung an die zu verarbeitende Last simulieren, und Feldtests, in denen eine OpenSOA-basierte Anwendung im Hause für die alltägliche Kommunikation eingesetzt wird. Die Verifikation des Funktionsumfangs und damit der erfolgreiche Abschluss der Migration scheinen zunächst trivial: Die funktionalen Tests müssen bei einer OpenJPA-basierten Version ebenso fehlerfrei abschneiden wie bei der Hibernate-Version. Die Herausforderung der Verifikation der Migration bestand jedoch darin, dass sich OpenSOA noch in der Weiterentwicklung befand. An dieser Stelle wird der heuristisch, iterative Charakter des Migrationsvorhabens deutlich.

## 5 Technische Aspekte der Migration

Die rein technische Lösung, der Patentklage zu begegnen, besteht darin, eine Alternative zu Hibernate in die OpenSOA-Software zu integrieren, schließlich ist eine Vielzahl an Persistenzframeworks verfügbar. Dabei ist aber nicht sichergestellt, ob andere Produkte nicht ebenfalls von der Patentklage betroffen sind und ob eine Klage gegen sie erhoben wird. Dennoch schien dies als der praktikabelste Weg, insbesondere weil viele Produkte einem gängigen Standard folgen und somit einen erneuten Wechsel des Produkts ermöglichen würden. Sollte ein Produkt von der Patentklage eingeholt werden, besteht die Möglichkeit, mit moderatem Aufwand auf ein Alternativprodukt auszuweichen. Standardkonformen Produkten ist sicherlich der Vorzug gegenüber anderen denkbaren Alternativen wie iBATIS mit proprietären Schnittstellen zu geben. Als Standards kommen derzeit Java Data Objects [JDO] und die Java Persistence API [JPA] als Bestandteil der EJB 3.0-Spezifikation in Frage. Dabei ist JPA im Vergleich zum JDO-Standard aktueller [MR02] und derzeit auch aus folgenden Gründen die bessere Wahl:

- JPA ist ein Teil des Standards für EJB3-Applikationsserver, er erlaubt also eine leichtere Migration von Stand-alone-Applikationen in EJB3-Applikationen und zurück. JPA-konforme Produkte sind benutzbar unter Java SE und Java EE.
- JPA beinhaltet nach eigenen Aussagen das „Beste“ aus Hibernate, TopLink, JDO und EJB2.1-Container-Managed Persistence. Gerade am JDO-Standard und an JDO-Produkten ist häufig Kritik geübt wurden, da Ersterer zu wenig mächtig ist und Zweitere viele (notwendige) nicht-standardkonforme Erweiterungen aufweisen.
- Für eine Ersetzung von Hibernate ist sicherlich relevant, dass JPA eine gewisse Hibernate-Nähe aufweist, insbesondere bei der Anfragesprache JPQL.

Im Projekt fiel die Wahl für einen Hibernate-Ersatz somit auf ein JPA-konformes Pro-

dukt, im genauen auf OpenJPA, welches als Projekt der Apache Software Foundation entwickelt wird. Es wird als Open Source Software entwickelt und vertrieben und basiert auf dem Produkt Kodo von BEA Systems. Im Jahr 2006 hat sich BEA entschlossen, den Grossteil des Produkts Kodo als Apache Software Projekt öffentlich weiter zu entwickeln, wohingegen sich die von BEA Systems weiterhin angebotene, kommerzielle Version durch Werkzeuge und Analysesoftware abhebt. Auch wenn sich Red Hat und Firestar im Mai 2008 bezüglich der Patentklage geeinigt haben, wurde die Migration abgeschlossen, zum einen, da OpenJPA unter der Apache Software Lizenz vertrieben wird. Diese bietet Vorteile gegenüber der LGPL von Hibernate, wenn es um den Umgang mit Erweiterungen geht. Zum anderen ist die Standardkompatibilität bedeutend, auch wenn Hibernate einen JPA-kompatiblen Aufsatz bietet, aber ansonsten proprietäre APIs bietet. Diese potenzielle Austauschbarkeit ist für industrielle Anwendungen ein wichtiges Argument.

### 5.1 Hibernate Grundlagen

Ein Persistenzframework wie Hibernate oder OpenJPA benötigt Information darüber [Be05], welche Java-Klassen und welche Felder persistent sind und wie die Java-Klassen auf Datenbanktabellen, Felder auf Tabellenspalten, Assoziationen auf Fremdschlüssel etc. abzubilden sind. Das kann entweder über XML-Spezifikationen in einer Mapping-Datei oder über Java-5-Annotationen in den Persistenz-Klassen erfolgen. In OpenSOA wurde Hibernate ausschließlich mit XML-Spezifikationen verwendet. Die Mapping-Dateien `Customer.hbm.xml` und `Order.hbm.xml` in Abbildung 3 beschreiben die Abbildung der Java-Klassen aus Abbildung 4.

```
<class name="Customer" table="C">
  <id name="id" column="cid" type="long">
    <generator class="native"/>
  </id>
  <property name="name" column="cname"/>
  <set name="orders" cascade="delete">
    <key column="custId"/> <one-to-many class="Order"/>
  </set>
</class>
```

```
<class name="Order table="O">
  <id name="oid">
  ...
</class>
```

Abbildung 3: Beispiel einer hbm-Mapping-Datei

```
class Customer {
  int    id;
  String name;
  Set    orders;
  <entsprechende get/set-Methoden>
}
```

```
class Order {
  int    oid;
  String part;
  <entsprechende get/set-Methoden>
}
```

Abbildung 4: Beispiel für dazu bezügliche Java-Klassen

Aufgrund der Mapping-Datei werden mittels der Angabe von `table="C"` Objekte der Klasse `Customer` in eine korrespondierende Tabelle `C(cid,cname)` abgelegt. Persistente Attribute werden mit `<property>` ausgezeichnet und mit `column=` auf Tabellenspalten abgebildet, beispielsweise `name` auf Spalte `cname`. Zudem wird ein eindeutiger Identifikator (OID) mit `<id>` ausgezeichnet. Alle Objekte der Klasse sind über ihren id-Wert unterscheidbar. Die Aufträge vom Typ `Order` gelangen analog in eine Tabelle `O(oi,part,custid)`. Die 1:n-Beziehung `Customer—Order` ist in `Customer.hbm.xml` als `<set name="orders"> <key column="custId"/> <one-to-many class="Order"/>`

vereinbart. Hierdurch erhält Tabelle `O` eine Spalte `custId` zur Repräsentation der Beziehung, die als Fremdschlüssel auf den Primärschlüssel `cid` von `C` verweist. Hibernate ermöglicht dabei eine umfangreiche Steuerung der Abbildung von Klassen und Beziehungen auf Tabellen, wie [Pl04] systematisch aufzeigt. Das folgende Quelltextbeispiel zeigt, wie mit der Hibernate API programmiert wird.

```
SessionFactory sf = new Configuration().buildSessionFactory();
Session mySession = sf.openSession();  // repräsentiert DB-Verbindung
Transaction tx = mySession.beginTransaction();
Customer c = new Customer(4711);
mySession.save(c);
Query q = mySession.createQuery("SELECT c FROM Customer c");
List result = q.list();
Customer c2 = mySession.get(Customer.class, 4711);
tx.commit();
mySession.close();
```

Nach dem Öffnen einer Session, d.i. im Prinzip eine Verbindung zum DBS, mit `openSession` wird eine Transaktion begonnen. In dieser wird zunächst ein Objekt der Klasse `Customer` angelegt, das anschließend mit einem `save`-Aufruf in der Datenbank gespeichert wird. Eine Anfrage liefert danach alle Java-Objekte der Klasse `Customer`. Alle gelesenen Objekte werden in einem Cache verwaltet [HoB06]. Ein zweiter Zugriff mit dem Aufruf `get` holt direkt den Kunden mit der Nummer 4711, der sich bereits aufgrund der vorangegangenen Anfrage im Cache befindet. Die Transaktion wird zum Schluss mit `commit` abgeschlossen und die Datenbankverbindung durch das Schließen der Session freigegeben. Auf die Behandlung von Ausnahmefällen wurde hier verzichtet.

### 5.2 Das Grundprinzip der Migration

Die Handhabung ist in OpenJPA recht ähnlich:

```
EntityManagerFactory emf =Persistence.createEntityManagerFactory("DB1");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
Customer c = new Customer(4711);
em.persist(c);
Query q = em.createQuery("SELECT c FROM Customer c");
List result = q.list();
Customer c2 = mySession.find(Customer.class, 4711);
tx.commit();
em.close();
```

Die Programmierung erfolgt mit Klassen `EntityManagerFactory`, `EntityManager` und `EntityTransaction`, die eine vergleichbare Funktionalität wie `SessionFactory`, `Session` bzw. `Transaction` bieten. So erfolgt in OpenJPA das Speichern eines Objekts mit `EntityManager.persist()` statt `Session.save()` usw. Daher ist eine direkte Umsetzung der Hibernate-Schnittstelle einfach möglich.

Bei einer Umstellung von Hibernate nach OpenJPA können die zu persistierenden Java-Klassen unverändert bleiben. Allerdings müssen die Mapping-Dateien umgestellt werden, da die XML-Formate eine unterschiedliche Syntax aufweisen. Eine XSLT-Transformation zur Überführung ist allerdings recht einfach zu erstellen. Kleinere konzeptionelle Unterschiede bestehen. Z.B. müssen transiente Attribute einer Klasse explizit als `<transient …/>` in der Mapping-Datei gekennzeichnet werden.

Im Grunde gibt es zwischen Hibernate und OpenJPA sehr viele Gemeinsamkeiten in grundlegenden Konzepten mit geringen syntaktischen Unterschieden. Das trifft auch für einige Besonderheiten wie Notifizierungen von Datenbankereignissen oder ein Konzept zu, dass die Austauschbarkeit von DBSen ermöglicht, aber dennoch proprietäre Konzepte eines einzelnen DBS effizient nutzt. In Hibernate kümmert sich ein spezielles Dialect-Konzept um die bestmögliche Umsetzung der Hibernate-Features auf adäquate DBS-spezifische Mittel. In OpenJPA gibt es ein korrespondierendes Dictionary-Konzept. Der Satz an vorgefertigten Dictionary-Klassen ist geringer als bei Hibernate, umfasst aber alle bekannteren DBSe, wobei beide SOLID nicht unterstützen.

Auch wenn viele Unterschiede durch einen Wrapper-Ansatz abgefangen werden können, treten doch ein paar Probleme auf, die eine weiter gehende Behandlung erfordern.

## 6 Probleme bei der Migration

Im Folgenden richtet sich der Fokus auf diverse Probleme, die Verzögerung bei der Umstellung von Hibernate auf OpenJPA bereiteten, wobei die Ergebnisse von [ViS07] hier ausgelassen werden. Die Erkenntnisse gelten im Prinzip auch für den JPA-Standard.

### 6.1 Unterschiede bei der Konfiguration

In Abschnitt 5.1 wurde bereits die Konfiguration mit Hibernate sichtbar. In Hibernate erhält man eine `SessionFactory` über ein `Configuration`-Objekt:

```
SessionFactory sf = new Configuration().buildSessionFactory();
```

Die `SessionFactory` repräsentiert eine Datenbank und verwaltet die zugehörige Mapping-Information. Hibernate benutzt eine Konfigurationsdatei, typischerweise mit Namen hibernate.cfg.xml, die von `buildSessionFactory()` gelesen und verarbeitet wird. Diese Konfigurationsdatei enthält typische Datenbankparameter wie die Datenbank-URL, den Treiber, Benutzer, Passwort und verweist mit `<mapping>` auf die persistenten Klassen. Sie kann über einen expliziten Pfad referenziert werden.

```
<property name="connection.url"> jdbc:mysql://localhost:3306/db </property>
<property name="connection.driver_class"> com.mysql.jdbc.Driver
</property>
<property name="dialect"> org.hibernate.dialect.MySQLDialect </property>
<property name="connection.username"> itsMe </property>
<property name="connection.password"> myPassword </property>
<property name="c3p0.max_size"> 50 </property> <!—c3p0 Connection Pool -->
<mapping resource="Customer.hbm.xml"/>         <!—Mapping-Dateien -->
```

OpenJPA verfährt ähnlich, führt aber eine sogenannte PersistenceUnit als logischen Datenbanknamen ein. Der Name wird der `createEntityManagerFactory`-Methode mitgegeben. In der Konfigurationsdatei persistence.xml lassen sich zu mehreren PersistenceUnits die Verbindungsoptionen (Treiber, URL etc.) vereinbaren. Hier steht auch ein Verweis auf die Mapping-Datei orm.xml bzw. die annotierten Klassen.

```
<persistence-unit name="DB1" transaction-type="RESOURCE_LOCAL">
  <class> Customer </class>
    <properties>
      <property name="openjpa.ConnectionDriverName"
                value="com.mysql.jdbc.Driver"/>
      <property name="openjpa.ConnectionURL"
```

```
                value="jdbc:mysql://localhost:3306/db"/>
    </properties>
  </persistence-unit>
</persistence>
```

OpenJPA erwartet die Dateien persistence.xml und orm.xml in einem META-INF-Verzeichnis innerhalb des classpath. Das ist nicht immer vorteilhaft. Da im Projekt OpenJPA in einem OSGi-Container benutzt wird, wird die Datenbank-Konfiguration Bestandteil der Deployment-JAR-Datei. Somit wird das projektinterne Deployment-Prinzip verletzt, dass das DBS, z.B. die IP-Adresse und der Port, jederzeit nach dem Deployment im Container beim Kunden einstellbar ist. Eine solche Änderbarkeit nach dem Deployment erfordert nun eine Nachbearbeitung der JAR-Datei, was auf dem Deployment-Rechner nicht möglich ist. Somit musste nach Möglichkeiten gesucht werden, DBS-spezifische Daten außerhalb des Deployments zu verwahren, aber dennoch die OpenJPA-Initialisierung zu gewährleisten.

### 6.2 Connection Pool

Zu den Best Practices von Hibernate zählt es, eine Session zu beenden, sobald die Transaktion mit `commit()` oder `rollback()` beendet wird. Es mag verwundern, dass nicht mehrere Transaktionen mit einer Session abgewickelt werden, aber der Grund liegt darin, dass bei Transaktionsende der Datenbestand in der Datenbank und im Session-Cache auseinander laufen; wenn andere Transaktionen dieselben Daten in der Zwischenzeit in der Datenbank geändert haben, werden diese Änderungen im Cache nicht sichtbar. Da eine Session eine Datenbankverbindung repräsentiert, deren Auf- und Abbau zeitaufwändig ist, verwaltet Hibernate einen Connection Pool, der logisch freigegebene Datenbankverbindungen nicht physisch freigibt, sondern für nachfolgende Benutzungen in einen Pool stellt. Daher sind keine Performanzeinbussen durch permanentes Öffnen und Schließen zu befürchten. Hibernate bietet eine vorgefertigte Konfiguration für den c3p0-Connection Pool. Die Handhabung ist sehr einfach über Einträge `<property name="c3p0.min_size">10</property>` in der Konfigurationsdatei vorzunehmen.

Die voreingestellte OpenJPA-Konfiguration benutzt standardmäßig keinen Connection Pool, was zu den angesprochenen Performanzeinbußen führt. Allerdings können DBCP oder c3p0 hinzugeschaltet werden. Bei der Verwendung von DBCP ist die Property `openjpa.ConnectionDriverName` statt `value="com.mysql.jdbc.Driver"` auf `value="org.apache.commons.dbcp.BasicDataSource"` zu setzen. Der eigentliche MySQL-Treibername ist dann zusammen mit der Datenbank-URL und den anderen Eigenschaften zu einer `openjpa.ConnectionProperties` zusammenzufassen:

```
<property name="openjpa.ConnectionProperties"
 value="DriverClassName=com.mysql.jdbc.Driver,
 Url=jdbc:mysql://localhost:3306/db,Username=itsMe,Password=myPassword"/>
```

Dieser Unterschied ist zunächst einmal nicht dramatisch, wirkt sich aber später noch beim Failover (siehe Abschnitt 6.7) negativ aus. Problematisch sind auch die Unterschiede bei den Konfigurationsparametern des Pools wie Initialgröße, minimale und maximale Größe und deren Bedeutung. Diese Parameter haben unterschiedliche Wirkung in Hibernate und OpenJPA. Der erste Versuch, DBCP zu konfigurieren, ergab z.B. ein unerwartetes oszillierendes Verhalten, bei dem der Pool trotz permanenter Last in kurzer zeitlicher Abfolge permanent geschrumpft und wieder angewachsen ist.

### 6.3 Lebenszyklus von Objekten

In Hibernate wird oft aus Performanzgründen ein Objekt als Muster zum Löschen benutzt, um einen vorangehenden Lesezugriff auf das Objekt zu vermeiden:

```
Customer c = new Customer(4711);
session.remove(c);  // Löschen des Objekts mit id=4711
```

Existiert ein Objekt mit der id=4711, so wird es gelöscht, andernfalls passiert nichts. Das funktioniert in OpenJPA so nicht, da durch den Konstruktor ein neues Objekt erzeugt wird, das zunächst temporär ist, da ein Persistieren mit `persist()` noch nicht erfolgt ist. Somit liegt bei `remove()` kein persistentes Objekt vor, welches zu löschen wäre, und es wird keine Datenbankoperation ausgelöst. Analoges gilt für ein Zurückschreiben des Objekts mit `update()`. Hier behandelt OpenJPA das Objekt ebenfalls als neu, so dass sich das DBS mit einer Eindeutigkeitsverletzung beklagt.

### 6.4 Unterschiede bei Anfragen

Obwohl die Hibernate Query Language HQL und OpenJPA's Pendant JPQL im Großen und Ganzen sehr ähnlich sind, treten dennoch im Praxisbetrieb Unterschiede auf, die mehr oder weniger Probleme aufwerfen können. Hierzu gehören kleinere syntaktische Unterschiede. Ein Vergleich mit `!=` ist in OpenJPA als `<>` zu notieren. Statt `SELECT COUNT(*) FROM Customer` ist die Form `SELECT COUNT(c) FROM Customer c` zu nehmen.

Im Gegensatz zu Hibernate sind in OpenJPA auch explizite Variablen in Pfadausdrücken erforderlich. Statt `SELECT c FROM Customer c WHERE name='Ms.Marple'` oder gar `FROM Customer WHERE name='Ms.Marple'` heißt es `SELECT c FROM Customer c WHERE c.name='Ms.Marple'`. Bei der Migration trat dabei das Problem auf, dass im GUI Bedingungen der Form `Feld=Wert` zusammengestellt werden, die direkt als `Feld=Wert` an Hibernate weitergereicht werden konnten. Mit OpenJPA ist eine Korrelationsvariable als `x.Feld=Wert` hinzuzufügen, die passend zur Klasse zu wählen ist.

Weitere syntaktische Unterschiede bestehen beim Eager Fetching. So lädt `SELECT x FROM Customer c JOIN FETCH c.orders o` in Hibernate mit dem Objekt `c` gleich die in Beziehung stehenden `Order`-Objekte mit aus der Datenbank in den Cache. In OpenJPA ist nur ein `JOIN FETCH c.orders` ohne Korrelationsvariable `o` möglich. Das Fehlen von `o` schränkt die Anfragemöglichkeiten ein. Das Lazy/Eager Fetching und die daraus resultierende Performanzproblematik ist ohnehin ein komplexes Thema.

Performanzkritisch ist auch, dass `DELETE FROM Customer c WHERE c.name='Ms.Marple'` in OpenJPA nicht funktioniert, weder mit noch ohne Variable `c`. OpenJPA produziert folgende Anfrage mit Selbstbezug, die in den meisten DBSen verboten ist:

```
DELETE FROM tab WHERE id IN (SELECT id FROM tab WHERE c.name='Ms. Marple')
```

### 6.5 Löschkaskadierung

Hibernate bietet eine flexible Steuerung, Datenbankoperationen wie das Löschen oder Speichern über Beziehungen zu kaskadieren. Eine `cascade`-Option ist für alle Beziehungsarten anwendbar. Zu kaskadierende Operationen sind `save-update` (Einfügen und Speichern), `delete` (Löschen), `all` (`save-update` und `delete`) sowie spezielle Formen

`delete-orphan` und `all-delete-orphan`: Während mit `cascade="delete"` für die `Customer-Order`-Beziehung vereinbart wird, dass mit einem `Customer`-Objekt auch die in Beziehung stehenden `Order`-Objekte gelöscht werden, sorgt die Option `delete-orphan` dafür, dass kein `Order`-Objekt ohne Vater existieren kann. Wird also die Beziehung zwischen dem `Customer`-Objekt und dem `Order`-Objekt aufgelöst, so verliert das Sohn-Objekt seine Lebensberechtigung und wird ebenfalls gelöscht. Bei `"delete"` bleibt das `Order` -Objekt gewissermaßen ohne Vater bestehen.

Eine Kaskadierung wird von OpenJPA ebenfalls unterstützt, mit Ausnahme von `delete-orphan`. Insofern muss die Migration einen adäquaten Ersatz bereitstellen.

### 6.6 Schlüssel-Generierung und Objektidentität

Jede persistente Java-Klasse erfordert einen Identifikator, der in Hibernate-Mappings mit `<id>` ausgezeichnet ist (vgl. Abb. 3) und Objekte der Klasse eindeutig identifiziert. Er wird auch als Primärschlüssel der zugrunde liegenden Tabelle benutzt. Hibernate unterstützt mehrere Strategien, die über `<generator>` ausgewählt werden. Beispielsweise kann Hibernate typische Mechanismen der DBSe wie Sequenzgeneratoren (z.B. SOLID) oder Autoinkrement-Spalten (z.B. MySQL) nutzen, um Schlüsselwerte zu belegen, die dann als Strategien `sequence` bzw. `identity` wählbar sind. Da die OpenSOA-Applikationen mehrere DBSe unterstützen müssen, insbesondere MySQL, SOLID und PostgreSQL, ist ein abstrakter, vom DBS unabhängiger Mechanismus nötig. Schließlich ist das Ziel eines O/R-Frameworks die Unabhängigkeit von DBSen, was eigentlich auch für Mapping-Dateien gelten sollte. Hibernate verfügt zu diesem Zweck über eine `native`-Strategie, die, je nachdem, was das zugrundeliegende DBS anbietet, entweder `sequence` oder `identity` auswählt.

OpenJPA bietet eine vergleichbare `auto`-Strategie, die ebenfalls OpenJPA entscheiden lässt, wobei es aber verfügbare Sequenzgeneratoren oder Autoinkrement-Spalten ignoriert und immer einen Hi/Lo-Mechanismus wählt, der High/Low-Werte in einer gesonderten Tabelle verwaltet. Das führt in OpenSOA natürlich zu Wertekonflikten bei bestehenden Datenbanken, da bereits durch Sequenzen oder Autoinkrement-Spalten vergebene Werte höchstwahrscheinlich nochmals mit Hi/Lo erzeugt werden. Im Prinzip kann man DBS-spezifische Mapping-Dateien pflegen, die je nach gewähltem DBS direkt `sequence` oder `identity` festlegen. Ein modellgetriebener Ansatz kann hier helfen, eine entsprechende Mapping-Datei zu erzeugen. Da die Mapping-Datei, wie bereits in Abschnitt 6.1 angesprochen, Bestandteil des Deployment-JARs sein muss, steht das wiederum im Konflikt zur Deployment-Strategie, die fordert, dass es ein DBS-unabhängiges Deployment gibt, so dass beim Kunden das DBS eingestellt werden kann. Eine Änderung der Deployment- und Installationsprozedur wäre wiederum sehr aufwändig. Das Problem wird noch dadurch verstärkt, dass sich einige OpenJPA-Konzepte nur als Annotation vorliegen, wie z.B. `delete-orphan`. Einerseits ist es mühselig, `delete-orphan` händisch zu implementieren, insbesondere wenn im Objektmodell Kaskaden über mehrere Stufen gehen. Nutzt man andererseits die `delete-orphan`-Option mit einer Annotation, so sind mehrere Quellcode-Varianten vorzuhalten, da die Mapping-Annotationen Bestandteil des Quellcodes sind. Die fehlende Unterstützung von `native` ist für die Migration daher schwerwiegend.

Des Weiteren bietet der JPA-Standard im Gegensatz zu Hibernate keine UUID-Generierung (weltweit global eindeutiger Identifier) und keine `increment`-Strategie (inkrementiere den höchsten Schlüsselwert der Tabelle). Auch hierfür sind Lösungen nötig.

### 6.7 Failover

Eines der DBSe, dass von OpenSOA zu unterstützen ist, ist SOLID. SOLID ist zwar weniger bekannt, aber dennoch im Bereich der Telekommunikation häufig vertreten, weil es ein interessantes Failover-Konzept bietet. So lassen sich zwei SOLID-Server installieren, ein Primärer und ein Sekundärer, deren Datenbanken sich synchronisieren. Stürzt der Primärserver ab, wird der Sekundärserver sofort zum Primärserver, der den Betrieb übernimmt. Damit Applikationen unabhängig davon sind, wer gerade Primärserver ist, müssen sie eine spezielle *dual-node* URL der Form `jdbc:solid://node1:1315,node2:1315/myusr/mypw` verwenden. Diese spezifiziert beide Server, auf node1 und node2. Dieses Failover-Konzept ist für das Projekt sehr bedeutend.

Da Hibernate keine Probleme mit der speziellen URL hatte, konnte man davon ausgehen, dass auch OpenJPA diese URL einfach zum JDBC-Treiber durchreicht. Bei der Testdurchführung stellte sich jedoch heraus, dass das Failover nicht funktionierte, es konnten überhaupt keine Verbindungen zur Datenbank aufgebaut werden. Tiefergehende Recherchen ergaben, dass die dual-node URL von OpenJPA verstümmelt wurde: Nur der erste Teil `jdbc:solid://comp1:1315` erreichte den SOLID-Server. Der Grund lag darin, dass OpenJPA alle Verbindungseigenschaften als `openjpa.ConnectionProperties` subsumiert, die URL, den Namen der Treiberklasse etc.:

```
Properties prop = new Properties();
String jpa = "Url=jdbc:solid://comp1:1315,comp2:1315/myusr/mypw,
              DriverClassName=solid.jdbc.SolidDriver, ...";
prop.setProperty("openjpa.ConnectionProperties", jpa);
EntityManagerFactory emf = provider.createEntityManagerFactory("db",prop);
```

Während der Analyse von `openjpa.ConnectionProperties` nimmt OpenJPA das Komma als Separierungssymbol und leitet somit die folgenden Einheiten ab:

```
Url=jdbc:solid://comp1:1315
comp2:1315/usr/pw
DriverClassName=solid.jdbc.SolidDriver
```

Da die zweite „dieser Einheiten“ nicht der Form *`property=value`* genügt, ignoriert OpenJPA sie schlichtweg; die URL wird zu `jdbc:solid://comp1:1315` gekürzt. Zur Lösung des Problems ist offensichtlich das Verhalten von OpenJPA so zu ändern, dass die URL intakt bleibt, was im Prinzip auf eine Änderung des Quellcodes hinausläuft.

### 6.8 Performanz

Das Lazy/Eager-Fetching-Prinzip ist für die Performanz von O/R-Applikationen sehr bedeutsam [BaH07]. Hibernate und OpenJPA unterscheiden sich hierbei konzeptionell als auch syntaktisch sehr stark. Generell sind die Möglichkeiten des JPA-Standards wesentlich geringer, was OpenJPA teilweise durch proprietäre Konzepte kompensiert.

Aus Performanzgründen ist es auch notwendig, OpenJPA mit einem eingeschalteten Query Compilation Cache zu betreiben, da ansonsten gleiche JPQL-Anfragen wiederholt

nach SQL transformiert werden. Im Zusammenspiel von OpenJPA und OSGi trat dabei das Problem auf, dass aufgrund eines Class Loading Problems Standard-Java-Klassen nicht gefunden wurden. Der Query Compilation Cache musste abgeschaltet werden.

## 7 Lösung der Probleme

Die zentrale Template-Bibliothek kann einen Teil der Migration abnehmen. Z.B. sind die in Abschnitt 6.1 erwähnten Konfigurationsprobleme dort zentral lösbar. Das Problem wird derart behoben, dass die Datenbankverbindungsdaten in eine externe properties-Datei ausgelagert und vom Template verarbeitet werden. Die Konfiguration ist somit nicht mehr Bestandteil des Deployments. Auch die Werte für den Connection Pool sind in der properties-Datei enthalten. Das Problem 6.2, geeignete Belegungen zu finden, lässt sich anhand der Dokumentation nicht lösen. Hier hilft nur das Ausprobieren und teilweise eine Analyse des Quellcodes, um die Wirkung der Parameter zu verstehen.

Ein zweiter zentraler Baustein bei der Migration war das Wrapping. Um die Umstellung von Hibernate auf OpenJPA auf möglichst wenige Quellcode-Änderungen zu reduzieren, wurde das Hibernate-API in einem eigenen Java-Package beibehalten, allerdings auf der Grundlage von OpenJPA implementiert. Um Hibernate auszutauschen, war lediglich der Import auf das neue Package zu legen.

Für alle Probleme, die dennoch zur Übersetzungszeit oder Laufzeit auftraten, mussten adäquate Lösungen bereitgestellt werden, die sich wie folgt kategorisieren lassen:

1. Benutzung nicht-standardkonformer OpenJPA-Erweiterungen: OpenJPA bietet einige proprietäre Erweiterungen an, die Hibernate-Funktionalität nachbilden. Obwohl diese die prinzipielle Austauschbarkeit von OpenJPA gefährden, wurde von ihnen Gebrauch gemacht, um Zeit bei der Migration zu sparen.
2. Änderung der Dictionary-Klasse: Wie auch Hibernate konzentriert OpenJPA die Abhängigkeit von DBSen in Dictionary-Klassen, die sich um die effiziente Umsetzung auf proprietäre Datenbankkonzepte kümmern.
3. Anwendungsprogrammierung: Obwohl der Wrapping-Ansatz syntaktische Unterschiede weitestgehend beheben konnte, musste in die Programmierung eingegriffen werden, um fehlende Hibernate-Funktionalität nachzuprogrammieren.
4. Nutzung der Aspekt-Orientierung: Für einige schwerwiegende Probleme wie das Failover reichten die vorangegangenen Mittel nicht aus. Mit der aspektorientierten Sprache AspectJ stand ein mächtiger Mechanismus bereit, diese Probleme schnell und elegant zu lösen.

### 7.1 Nicht-standardkonforme OpenJPA-Erweiterungen

Wie in Abschnitt 6.3 erwähnt, sieht der JPA-Standard keine UUID-Schlüsselgenerierung vor, OpenJPA bietet dennoch einen proprietären Mechanismus als Erweiterung der AUTO-Strategie: `<generated-value strategy="AUTO" generator="uuid-hex"/>`. Leider funktioniert das in der Version 0.9.7 von OpenJPA, mit der die Migration begann, nicht mit XML-Mapping-Dateien. Man kann aber auf folgende Annotation ausweichen:

```
@Id @GeneratedValue(strategy=GenerationType.AUTO,generator="uuid-hex")
```

Ebenso stellt OpenJPA einen nicht-JPA-konformen Extra-Mechanismus bereit, der eine `delete-orphan`-Kaskadierung über eine Annotation `@ElementDependant` (aber wieder ohne XML-Äquivalent) ermöglicht. Diese wurde benutzt, um aufwändige, händische Programmierung zu vermeiden.

### 7.2 Änderung der Dictionary-Klasse

Ein Persistenzframework sollte die Austauschbarkeit von DBSen ermöglichen, aber dennoch die Möglichkeit haben, proprietäre Konzepte effizient zu nutzen. Diesem Anspruch wird in OpenJPA durch ein spezielles Dictionary -Konzept Rechnung getragen, dass sich um die bestmögliche Umsetzung von OpenJPA auf adäquate DBS-spezifische Konzepte kümmert. Das performanzkritische DELETE-by-Query-Problem (vgl. 6.4) lässt sich über eine geänderte Dictionary-Klasse lösen, indem das DBS-spezifische-Dictionary so abgeändert wird, dass eine relationale SQL-Anweisung ohne Selbstbezug generiert wird:

```
DELETE FROM tab WHERE c.name='Ms. Marple'
```

### 7.3 Anwendungsprogrammierung

Teilweise musste in die Anwendungsprogrammierung eingegriffen werden, um semantische Unterschiede auszugleichen oder fehlende Hibernate-Funktionalität auszuprogrammieren. Ein Beispiel hierfür ist die fehlende UUID-Schlüsselgenerierung, die alternativ zur `@GeneratedValue`-Annotation (siehe 7.1) auch durch explizite Initialisierung mit einem UUID-Generator realisiert werden kann. Das Fehlen des `increment`-Mechanismus' ist ebenfalls nicht so dramatisch, da sich dahinter nur eine `SELECT MAX`-Anfrage verbirgt. Syntaktische Unterschiede in den Anfragesprachen und das Beheben von Performanzproblemen wurden ebenfalls in der Programmierung behoben.

### 7.4 Aspekt-Orientierte Programmierung

#### Simulation der `native`-Schlüsselgenerierung

Wie bereits erwähnt, ist es aus Gründen des projektinternen Deployments nicht möglich, verschiedene XML-Mapping-Dateien, eins für jedes DBS mit der jeweiligen Strategie `sequence` oder `identity`, bzw. bei Benutzung von Annotationen sogar mehrere Varianten von Java-Klassen vorzuhalten.

Die wesentliche Idee, dennoch einen abstrakten Mechanismus in OpenJPA einzubringen, besteht nun darin, das Verhalten beider Einzelstrategien `sequence` und `identity` so abzuändern, dass OpenJPA intern zur richtigen Strategie wechselt. Das heißt, ist `identity` zwar gewählt, wird aber vom DBS keine Autoinkrement-Spalte unterstützt, so wird intern `sequence` gewählt. Im Prinzip ist für diesen Ansatz eine Änderung des OpenJPA-Quellcodes nötig, der glücklicherweise als Open-Source vorliegt. Eine Quellcode-Änderung impliziert aber, dass der build-Prozess von OpenJPA verstanden und in den OpenSOA-build-Prozess integrierbar ist. Da dieses sehr aufwändig ist, wurde der Einsatz von AspectJ [La03] beschlossen.

Der folgende Aspekt modifiziert das OpenJPA-Verhalten entsprechend.

```
@Aspect
```

```
public class KeyGenerationAspect {
  private String db = null;
  private static final int JPA_STRATEGY_SEQUENCE = 2;
  private static final int JPA_STRATEGY_IDENTITY = 3;
  @Before("execution(*
           org..PersistenceProviderImpl.createEntityManagerFactory(..))
       && args(.., p)")
  public void determineDBS(final java.util.Properties p) {
    String str = p.getProperty("openjpa.ConnectionProperties");
    if (str.contains("Solid"))           db = "SOLID";
    else if (str.contains("mysql"))      db = "MYSQL";
    else if (str.contains("postgresql")) db = "POSTGRES";
  }
  @Around("call(*
           org.apache.openjpa.meta.FieldMetaData.getValueStrategy(..))
       && !within(com.siemens.ct.aspects.*)")
  public Object useAppropriateStrategy(final JoinPoint jp) {
    FieldMetaData fmd = (FieldMetaData) jp.getTarget();
    int strat = fmd.getValueStrategy();
    if (db.equals("SOLID") && strat==OPENJPA_STRATEGY_IDENTITY){
      fmd.setValueSequenceName("system");
      return JPA_STRATEGY_SEQUENCE;
    } ... // analog für "MYSQL"
    return strat;
} }
```

`KeyGenerationAspect` ist eine normale Java-Klasse, die mit `@Aspect` zu einem AspectJ-Aspekt wird. Trotz der Nutzung von AspectJ konnte die Entwicklungsumgebung mit Eclipse weiterhin mit einem gewöhnlichen Java-Compiler – also ohne AspectJ-Plugin AJDT – betrieben werden. Dieser Punkt war für das Projekt sehr bedeutsam.

Der Aspekt implementiert zwei Methoden: Die Erste `determineDBS` bestimmt das DBS und die Zweite `useAppropriateStrategy` ändert die Strategie, sofern nötig. Beide Methoden tauschen den DBS-Typ mittels einer lokalen Variablen `db` aus. Dadurch, dass die Methode `determineDBS` mit `@Before` annotiert ist, wird sie zu einem Before-Advice, deren Code *vor* sogenannten Joinpoints ausgeführt wird. Die Joinpoints legen fest, wo der Aspekt eingreifen soll. Diese Stellen sind als Zeichenkette in der Annotation spezifiziert: Jede Ausführung einer Methode `PersistenceProviderImpl.CreateEntityManagerFactory` mit einem `Properties`-Parameter wird abgefangen. Obwohl die Parameterliste "`(..)`" Methoden mit beliebigen Parametertypen fängt, schränkt `args(..,p)` implizit die Methoden auf solche ein, die einen `Properties`-Parameter haben, und bindet gleichzeitig eine Variable `p` an diesen Parameter. Die Variable erscheint auch in der Methodensignatur, wodurch der Zugriff auf Parameterwerte innerhalb des Methodenrumpfs möglich wird: Mit `p.getProperty("openjpa.ConnectionProperties")` kann die kommaseparierte `ConnectionProperties`-Liste erfragt und der Typ des DBSs ermittelt werden. Das Ergebnis wird in einer Aspekt-internen Variablen `db` abgelegt.

Die `useAppropriateStrategy` Methode benutzt das Wissen über den DBS-Typ und wechselt von der `identity` zur `sequence` Strategie im Falle SOLID. Der Aspekt kann somit als einfacher Mechanismus genutzt werden, Daten unter Advices auszutauschen, selbst wenn die abgefangenen Methoden bislang keinerlei Bezug aufweisen, z.B. weil sie in verschiedenen JARs liegen. Der `@Around` Advice `useAppropriateStrategy` ersetzt die ursprüngliche OpenJPA-Logik von `getValueStrategy` und entscheidet anhand des

DBS-Typs und des Ergebnisses von `fmd.getValueStrategy()`, die Strategie zu ändern. Diese Logik wird an allen aufrufenden Stellen aktiv. Der Parameter `jp` liefert Kontextinformation über den Joinpoint, insbesondere das Objekt `jp.getTarget()`, dessen Methode aufgerufen wird. In diesem Fall muss es ein `FieldMetaData`-Objekt `fmd` sein. Zu beachten ist, dass die Klausel `!within(...)` Rekursionen vermeidet: Sie verhindert, dass Aufrufe von `getValueStrategy` innerhalb des Aspekts selbst abgefangen werden.

In der Tat ist das eine Art von Quellcode-"Patching": Das Verhalten von 3rd-Party-Software, hier OpenJPA, wird modifiziert, allerdings ohne direkten Einfluss auf den Quellcode zu nehmen. Zu bemerken ist, dass hierfür der Quellcode nicht vorliegen muss. Der definierte Aspekt wirkt auf JAR-Dateien, in diesem Fall entsteht also eine neue openjar.jar Datei, welche die Aspektlogik enthält. Für den build-Prozess bedeutet das, dass nur ein zusätzlicher Schritt zur Erzeugung der JAR-Datei(en) nötig ist.

Dieses Vorgehen ist wesentlich einfacher als in OpenJPA eine neue `native` Strategie einzuführen. Hierdurch wären mehrere Bestandteile von OpenJPA betroffen wie die XML-Parser für Mapping-Dateien, die Analyse von Annotationen, die interne Verwaltung der Strategien als Metadaten und das Erzeugen der SQL-Operationén.

**Behebung des Failover-Problems**

Ein weiterer Aspekt löst das sehr kritische Failover-Problem elegant und schnell. Ein `@Around`-Advice fängt die Ausführung der „fehlerhaften" `parseProperties`-Methode ab und ersetzt sie durch eine korrigierende Logik:

```
@Around("execution(public static Options
       org.apache.openjpa.lib.conf.Configurations.parseProperties(String))
       && args(props)")
public Object parseProperties(final String s){
  Options opts;
  Analysiere den properties-String s korrekt und baue die URL
  korrekt zusammen;
  return opts;
}
```

Leider löst der Aspekt nur einen Teil des Failover-Problems: Von nun an wird zwar die URL korrekt zum JDBC-Treiber durchgelassen, aber der Failover vom Primär- zum Sekundärserver findet nicht statt. Der Grund hierfür liegt darin, dass OpenJPA die wichtige Verbindungsoption `solid_tf_level` verschluckt, die für jede Datenbankverbindung gesetzt sein muss. In OpenJPA kann man derartige Optionen im `prop`-String (siehe 6.7) mitgeben, es werden aber nur OpenJPA-Optionen, also solche, die mit"`openjpa.`" beginnen, zum JDBC-Treiber durchgelassen; andere werden schlichtweg ignoriert.

Ein weiterer Aspekt fügt jedem `SolidDriver.connect(...,Properties)`, also jeder Verbindungsanforderung, die `solid_tf_level` Option hinzu, indem er die Ausführung dieser Methode abfängt und den `Properties`-Parameter modifiziert:

```
@Before("execution(*solid.jdbc.SolidDriver.connect
                   (..,String,..,Properties,..)) && args(url, prop)")
public void addSolidTfLevel(String url,Properties prop) {
    if (url!=null && url.contains("solid"))
       prop.setProperty("solid_tf_level","1");
}
```

`(..,String,..,Properties,..)` spezifiert die relevanten Parameter. Die `args`-Klausel bindet entsprechende Variable `url` and `prop` an diese. Die Variable `url` ist nötig, um das DBS zu identifizieren, während `prop` zum Setzen der `solid_tf_level`-Option benutzt wird. Auch in diesem Fall wird eine externe JAR-Datei, der SOLID-JDBC-Treiber, modifiziert, deren Quellcode nicht verfügbar ist!

Erwähnenswert ist darüber hinaus, dass der Einsatz von Aspekten auch geholfen hat, das Failover-Problem in kurzer Zeit zu identifizieren. Folgender Aspekt übernimmt eine spezielle Form des Tracings, indem er nur relevante Ausgaben produziert:

```
@Before("execution(* *.*(..,String,..))")
public void myTrace(final JoinPoint jp) {
  Object[] args = jp.getArgs();
  for (Object param : args) {
   if (param instanceof String
   && param != null &&((String) param).contains("jdbc:solid:"))
       System.out.println("* In: " + jp.getSignature()
                          + "-> " + param.toString());
} } }
```

Dieser `@Before`-Advice fängt alle Ausführungen (`execution`) aller Methoden ab (die Wildcard `*` in `(* *.*)` bedeutet beliebiger Rückgabetyp, beliebige Klasse und Methode), die einen String-Parameter (`(..,String,..)`) besitzen, und prüft, ob der String eine SOLID URL enthält. Wenn dem so ist, wird die URL ausgegeben. Der Parameter `JoinPoint jp` liefert dabei Kontextinformation über den Joinpoint, z.B., `jp.getSignature` gibt die Signatur der abgefangenen Methode aus und `jp.getArgs()` die übergebenen Parameterwerte. Somit lässt sich der Übergang von einer korrekten zu einer abgeschnittenen URL sehr schnell in `Configurations.parseProperties` bestimmen:

```
* In: Options org.apache.openjpa.lib.conf.Configurations.parseProperties
  (String)
-> DriverClassName=solid.jdbc.SolidDriver,Url=
  jdbc:solid://host1:1315,host2:1315/usr/pw,defaultAutoCommit=false,initia
  lSize=35,maxActive=35,maxIdle=35,minIdle=10,minEvictableIdleTimeMillis=6
  0000,timeBetweenEvictionRunsMillis=60000,defaultTransactionIsolation=4
* In: boolean solid.jdbc.SolidDriver.acceptsUR L(String)->
  jdbc:solid://host1:1315
```

## 8 Zusammenfassung

Diese Arbeit hat gezeigt, dass die Migration von Hibernate nach OpenJPA mit moderatem Aufwand möglich ist. In dem vorgestellten Projekt kommt entgegen, dass das Persistenzframework in einer Template-Bibliothek gekapselt ist, so dass durch deren Umstellung bereits ein Großteil der Arbeiten an zentraler Stelle erledigt wird. Die Erfahrung zeigt auch, dass eine zügige Projektfindung in Verbindung mit einem heuristischem, iterativen Vorgehen zu einem fundierten und erfolgreichen Ergebnis führt.

Nichtsdestotrotz soll diese Arbeit betonen, dass eine Reihe von kniffligen Problemen auftraten, z.B. das Fehlen der nativen Schlüsselgenerierung oder das Failover-Problem, die im Vorfeld nicht ohne weiteres zu erwarten waren. Die verwendeten Lösungsstrategien sollten hierbei verdeutlichen, dass sehr unterschiedliche Ansätze den gewünschten Erfolg bringen können. Darüber hinaus sind Probleme aufgetreten, die an

sich keine besondere Verbindung mit der eigentlichen Domäne aufweise. Als Beispiel sei das Problem von OpenJPA mit der Ladereihenfolge von Klassenpfaden aufgrund eines „Split Packages“ im OSGi-Umfeld zu nennen (siehe Abschnitt 6.8). Dennoch stellte sich keines der Probleme als unüberwindbar hinaus, so dass die erfolgreiche Migration als Empfehlung für den Einsatz von OpenJPA gewertet werden kann.

Trotz der speziellen Natur einzelner Probleme, muss natürlich abschließend betont werden, dass der maßgebliche und grundlegende Erfolgsfaktor für die Migration die umfangreiche Testinfrastruktur war. Generell gilt, dass das Aufspüren von Problemen aufwändiger ist als deren Lösung. Ohne die vorhandenen Tests wäre eine Verifikation der Migration unmöglich gewesen, d.h. die Lauffähigkeit der OpenJPA-basierten Variante von OpenSOA wäre nicht nachweisbar gewesen. Dies unterstreicht ein weiteres Mal die Wichtigkeit von Softwaretests.

## Referenzen


[BaH07] J. Bartholdt, U. Hohenstein: Caching und Transaktionen in Hibernate für Fortgeschrittene. JavaSpektrum 2007

[BaS05] M. Bannert, M. Stephan: JDO mit Kodo - ein Praxisbericht. JavaSPEKTRUM, Sonderheft CeBIT 2005

[Be05] U. Bettag: Und ewig schläft das Murmeltier: Persistenz mit Hibernate. JavaSPEKTRUM, Sonderheft CeBIT 2005

[De05] F. von Delius: JDO und Hibernate: zwei Java-Persistenztechnologien im Vergleich. Javamagazin 6/2005

[EFB01] T. Elrad, R. Filman, A. Bader (eds.): Theme Section on Aspect-Oriented Programming. CACM 44(10), 2001

[Hib] Hibernate Reference Documentation. http://www.hibernate.org /hib_docs/v3/ reference/en/html/

[HoB06] U. Hohenstein, J. Bartholdt: "Performante Anwendungen mit Hibernate" in JavaSPEKTRUM 2/06

[JDO] JSR-000012 JavaTM Data Objects Specification. http://jcp.org/aboutJava/ communityprocess/first/ jsr012/

[JPA] Java Persistence API. http://java.sun.com/javaee/technologies/persistence.jsp

[KB04] G. King, C. Bauer: Hibernate in Action. Manning 2004

[La03] R. Laddad: AspectJ in Action. Manning Publications Greenwich 2003

[MR02] K. Müllner, K.-H. Rau, S. Schleicher: Java-basierte Datenhaltung: Ein kritischer Vergleich (Teil 1). JavaSPEKTRUM 9/10 2002

[Ne94] Jon Udell: Next's Enterprise Objects Framework, Byte Magazine, July 1994

[PaT06] S.E. Pagop, M. Tilly: “Caching Hibernate“, JavaMagazin 3/4 2006

[Pl04] M. Plöd: Winterschläfer: Objketrelationales Mapping mit Hibernate. Javamagazin 8/2004

[Str07] W. Strunk: The Symphonia Product-Line. Java and Object-Oriented (JAOO) Conference, Arhus, Denmark, 2007

[HOD+00] R. Heubner, G. Oancea, R. Donald, J. Coleman: Object Model Mapping and Runtime Engine for Employing Relational Database with Object Oriented Software, United States Patent 6,101,502, Appl No. 09/161,028,August 2000

[ViS07] D. Vines, K. Sutter: Migrating Legacy Hibernate Applications to OpenJPA and EJB 3.0. http://www.ibm.com/developerworks/websphere/techjournal/0708_vines/0708_vines.html

# Demo-Programm

# Retrieving metadata for your local scholarly papers

David Aumueller

University of Leipzig
david@informatik.uni-leipzig.de

**Abstract:** We present a novel approach to retrieve metadata to scholarly papers stored locally as PDF files. A fingerprint is produced from the PDF fulltext to query an online metadata repository. The returned results are matched back to identify the correct metadata entry. These metadata can then be stored in the PDF itself, indexed for a desktop search engine, and collected in a user's or community's bibliography. We think this hitherto missing link but with our tool now available data eases the organization of scholarly papers, and increases accessibility to one's collected academic content.

## 1 Motivation

Desktop search engines and the so-called semantic desktop depend on the availability of according metadata. When downloading PDFs from the web the according metadata attributes are lost unless sophisticated download managers are in place picking up the metadata from the repository the PDF is catalogued in. [Lu08] reviews a current tool that at least helps to collect the metadata of papers found on such scholarly repositories. **The missing link:** Papers locally stored as PDF often lack correct metadata and thus are hardly accessible even via current desktop search engines. E.g. the stored title in a PDF document often resembles the filename of the source document the PDF got produced from. The author/creator often is set to the login name of the user having created the PDF, or worse, the initial creator of the template the document was started on (cf. fig. 1).

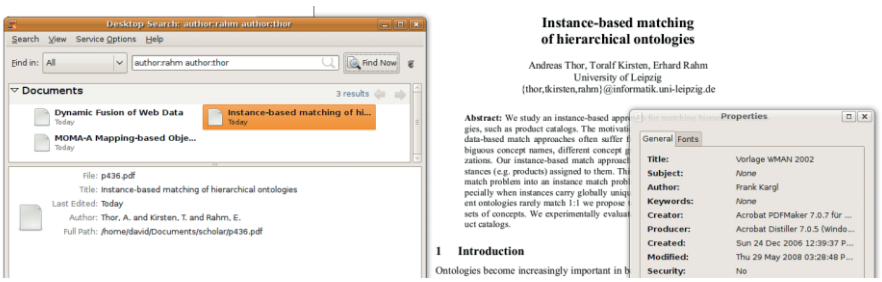



Figure 1: Illustrating the use of the retrieved metadata by desktop searching scholarly PDFs

For scholarly papers important metadata are authors, title, year, and reference (venue such as conference or journal), as available in metadata repositories such as the ACM digital library or Google Scholar (GS). Extracting such metadata directly from the PDF is difficult, cf. [Ha03], nevertheless GS, Citeseer etc. enlarge their repositories using extraction techniques. Generally, in the scholarly domain the mapping between the

metadata of an article and its fulltext as PDF file is indirectly given by following a download link on a webpage describing the article. This mapping though is lost when storing the file without reference to the source. Thus, local collections of articles grow without corresponding metadata. **Metadata to local files via web service:** We establish the mapping by matching the contents of local PDF files to a metadata repository. Analogue examples in other domains include the establishment of mappings between locally stored movie files and its corresponding film descriptions on movie web sites. Considering audio files, e.g. MP3-files, mappings to the corresponding metadata entries can be established using a variety of already existing tools that e.g. calculate a musical fingerprint and compare that to a user-contributed database on the web, cf. [HG04].

## 2 Approach – matching fulltext to metadata

Our novel approach to establish a mapping between locally stored scholarly articles in PDF file format and corresponding metadata entries consists of the following steps: converting the PDF to text, determining a suitable fingerprint, i.e. the query terms that are likely to locate the paper in a metadata repository, retrieving and parsing the resulting metadata entries, matching these with the document in question, and storing the according metadata. **PDF fingerprint:** As Google Scholar indexes the fulltext and not only the metadata of papers we can query this service using any identifying text fragment from the fulltext. Our approach takes a fragment from the beginning of the document which usually contains the content of identifying attributes, such as title and abstract excerpts. As we are using Linux' pdftotext we still have to replace some special characters such as ligatures with their counterparts in plain ASCII. Also, often author indices, such as [1, 2], need to be set off from the name itself. Thus, to be more robust, we restrict our search terms to phrases from the beginning of the document containing only words in the [a-z] character class. We experimented with the whole document head including title, authors, affiliations and email addresses but experienced issues regarding diverting textual representations locally and in the remote repository. If no match is found using the first bunch of phrases as query, it might be due to some extra information placed on the top of the document. In this case we skip the first phrase for a second query. Furthermore, at least the title of a document can more often also be acquired from general web search engines, as e.g. Yahoo also indexes PDF fulltexts and successfully extracts titles thereto. These titles could then be used to query the more specific, scholarly metadata repositories. **Result matching:** As querying a search engine will usually result in multiple different entries, the correct entry, if available, has to be matched to the document in question. We match back the result set to the fulltext of the local document by checking whether the title string is contained in the fulltext, preferably in the head, i.e. the fragment above the first mention of the keyword 'abstract'. For this match task we normalize both the title retrieved from the metadata service and the local fulltext alike. **Evaluation:** We evaluated our approach with local PDF files of the VLDB 2007 proceedings (91 papers in research track). For this set we yielded 100% accuracy, i.e. for each paper the corresponding metadata entry was found in Google Scholar. Generally, not every scholarly PDF file is indexed in Google Scholar, e.g. many articles from the BTW series are still missing. Surplus, PDF encryption or restrictions on reading may be in place hindering the extraction of text to create a query.

## 3 Application – using the scholarly metadata to local PDFs

**Bibliographies:** Metadata to a collection of scholarly PDFs can be maintained in local or remote databases, e.g. in the form of BibTex entries. As with our tool collected BibTex entries link back to the local PDF file, these fulltexts can be opened instantaneously from within the bibliography manager, e.g. JabRef. Community based bibliographies, such as the ones driven e.g. by Caravela [AR07], could be augmented with papers and their corresponding metadata including extracted abstract simply by uploading or pointing to PDF files. **Desktop search:** Generally, the more metadata attributes available in search engines the more expressive queries the users can pose. The retrieved metadata to local scholarly articles in PDF documents can be integrated in desktop search engines. Searching for locally stored papers e.g. via author name, title, year, and/or venue becomes possible. As proof-of concept we implemented a 'filter' for 'beagle', a popular desktop search engine for Linux. Figure 1 illustrates this by showing a rendered PDF in the background, its improper PDF metadata in the property window (right), and the article found by desktop searching for its real metadata (left).

## 4 Demonstration

On-site demonstration attendees may download scholarly PDFs off the net or supply otherwise and use our tool to retrieve according metadata. We showcase results within a bibliography manager as well as a desktop search engine (querying for some metadata to see whether the supplied PDFs are correctly indexed). The approach can be scrutinized step-by-step, i.e. converting the PDF to text, determining the query terms, inspecting the search results, matching the result entries to the document, and storing the according metadata. For a first experience we set up a demo on labs.dbs.uni-leipzig.de, incorporating our "PDF MEtadata Acquisition Tool" (pdfmeat), currently coded in Perl.

Querying web services to retrieve metadata for local files is quite new and applicable to other domains as well, e.g. to index movie details to locally stored films by querying an online movie database via cleaned filenames. Here, we presented our approach to establishing a mapping between local scholarly PDFs and its corresponding metadata entries and described how this data may increase accessibility to one's academic content.

## References


[AR07] Aumueller, D., Rahm, E., Caravela: Semantic Content Management with Automatic Information Integration and Categorization. ESWC, 2007

[Ha03] Han, H. et al. Automatic document metadata extraction using support vector machines. Digital Libraries, 2003

[HG04] Howison, J., Goodrum, A. Why can't I manage academic papers like MP3s? The evolution and intent of Metadata standards. Colleges, Code and Intellectual Property Conference, 2004

[Lu08] Lucas, Daniel V. A product review of Zotero. Master's thesis, University of North Carolina at Chapel Hill, School of Information and Library Science, 2008

# OttoQL

Klaus Benecke
benecke@iws.cs.uni-magdeburg.de
Martin Schnabel

IWS/FIN, Otto-von-Guericke-Universität Magdeburg
Postfach 4120
39016 Magdeburg, Germany, Sachsen-Anhalt

## 1 Einführung

OttoQL ist eine Anfragesprache für strukturierte Daten. Hauptaugenmerk wird auf Einsteigerfreundlichkeit der Sprache gelegt. OttoQL im aktuellen Entwicklungsstand erlaubt es beliebige XML-Dateien, die über eine Dokumenttypdefinition (DTD) verfügen zu verarbeiten. Mit wenigen aber leistungsstarken Operationen können Daten verarbeitet werden. Die Operationen arbeiten mengenorientiert, die Daten werden also in ihrer Gesamtheit betrachtet - Iteration durch einzelne Elemente ist nicht notwendig. Für Einsteiger und Programmierer funktionaler Sprachen ist diese Vorgehensweise intuitiv. Programmierer imperativer Sprachen fühlen sich mit diesem Konzept anfangs unwohl, erkennen dann aber dessen Eleganz. Der große Vorteil ist, dass man nicht beschreibt, *wie* die Daten umgewandelt werden sollen, sondern in *was* die Daten umgewandelt werden sollen. Um das *Wie* kümmert sich OttoQL.

Die drei Grundoperationen sind die Erweiterung, die Selektion und die Umstrukturierung. Durch nacheinander Anwendung dieser Operationen kommt der Benutzer Schritt für Schritt zum Ziel. Die meisten OttoQL-Programme kommen mit diesen drei Grundoperationen aus. Zusätzlich verfügt die Sprache aber auch über Variablen, Schleifen, bedingte Ausführung, Funktionen und andere gewohnte Bestandteile.

Die Einsatzmöglichkeiten von OttoQL sind vielseitig. Fast überall wo Daten ausgewertet werden sollen, kann OttoQL gute Dienste leisten. Die Daten müssen nur als XML-Dateien vorliegen. Eine Anbindung an relationale Datenbanken ist wünschenswert, wurde aber aus Ressourcenmangel noch nicht umgesetzt. Um größere Datenmengen handhaben zu können, implementiert OttoQL ein einges Datenformat (H2O), das geeignete „XML-Sätze“ nach dem TID-Konzept adressiert.

Außerdem soll es einen visuellen Editor für OttoQL-Programme geben, der die Operationen grafisch darstellen und per Drag’n’Drop erzeugen kann.

## 2 Die drei Grundoperationen - ganz kurz

OttoQL sieht XML-Dateien als strukturierte Daten bzw. Tabellen an. Eine flache Tabelle ist eine gewöhnliche Tabelle mit Tabellenkopf und Daten in den Zellen (Abbildung 1). Die strukturierte Tabelle kann auch Tabellen (auch wiederum strukturiert) in den Zellen enthalten (Abbildung 2). Die Beschreibung dieser verschachtelten Tabellenköpfe erfolgt durch ein Schema oder eine Dokumenttypdefinition (DTD). Die Operationen können auf

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

Abbildung 1: flache Tabelle

| Artikel | Preis | M(Lager, Anzahl) | |
|---|---|---|---|
| Monitor | 124.98 | Lager | Anzahl |
| | | Braunschweig | 7 |
| | | Magdeburg | 11 |
| Tastatur | 28.95 | Lager | Anzahl |
| | | Berlin | 23 |
| | | Magdeburg | 42 |

Abbildung 2: strukturierte Tabelle

diese strukturierten Tabellen angewendet werden, um die Tabellen zu erweitern (ext), zu filtern (mit/sans) oder umzustrukturieren (gib).

- Die **Erweiterung (ext)** fügt Spalten hinzu. Als Argumente werden ein (optionaler) Spaltenname und ein Ausdruck benötigt. Ob die äußere oder eine weiter innen liegende Tabelle erweitert wird, erkennt *ext* anhand der Variablen im Ausdruck, die sich auf Spalten des Ausgangsdokumentes beziehen.
- Die **Selektion (mit/sans)** löscht Zeilen abhängig von einer Bedingung.
- Die **Umstrukturierung (gib)** (vgl. [1] und [2]) wandelt die Struktur einer Tabelle. Sie benötigt dazu keine Regeln, sondern allein die Angabe eines neuen Schemas bzw. einer neuen DTD. Sie ist damit eine sehr mächtige Operation. Zusätzlich kann die Umstrukturierung neue Spalten erzeugen indem Daten aus vorhandenen Spalten aggregiert werden. Spalten werden eliminiert, indem man sie im Zielschema weglässt.

## 3 Beispiele

Das folgende OttoQL-Programm demonstriert die Erweiterung:

```
ext doc("inventar.xml")
ext Wert := Preis * Anzahl
ext Gesamtzahl := sum(L(Anzahl)) at Preis
```

Das erste *ext* holt die Daten aus Abbildung 2. Nun wird die Spalte *Wert* aus dem Produkt von *Preis* und *Anzahl* berechnet. Die Spalte wird den inneren Tabellen hinzugefügt, weil die Variable *Anzahl* im Ausdruck aus den inneren Tabellen kommt. Die Berechnung der *Gesamtzahl* im letzten Schritt erzeugt eine neue Spalte in der äußeren Tabelle. Im Ausdruck steht zwar *Anzahl* aus den inneren Tabellen, die Listenbildung **L**(..) führt aber aus ihnen heraus. Das Ergebnis ist in Abbildung 3 in Tabellenform zu sehen.

| Artikel | Preis | Gesamtzahl | M(Lager, Anzahl, Wert) |
|---|---|---|---|
| Monitor | 124.98 | 18 | Lager: Braunschweig, Anzahl: 7, Wert: 874.86<br>Lager: Magdeburg, Anzahl: 11, Wert: 1374.78 |
| Tastatur | 28.95 | 65 | Lager: Berlin, Anzahl: 23, Wert: 665.85<br>Lager: Magdeburg, Anzahl: 42, Wert: 1215.9 |

Abbildung 3: Ausgabe

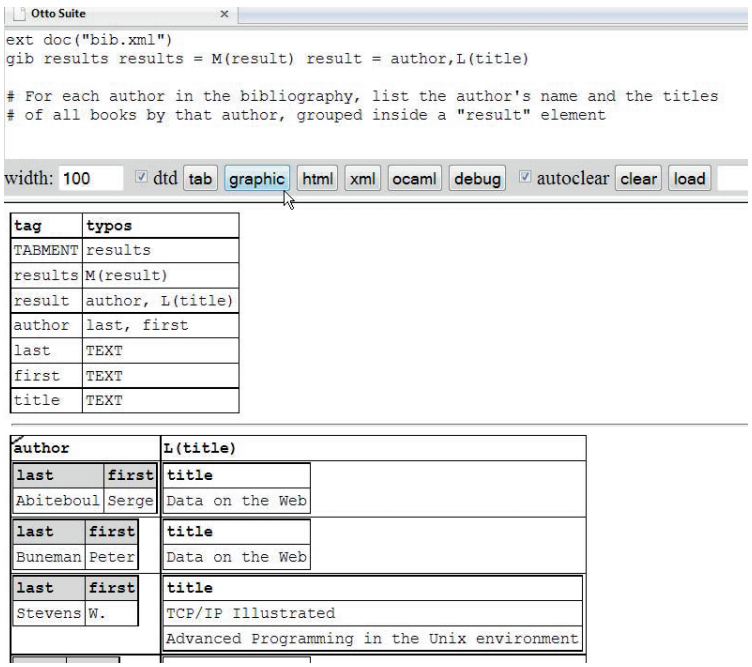



Abbildung 4: Screenshot der OttoQL-Oberfläche

Im Folgenden wird ein XML Query Use Case [3, 1.1.9.4 Q4] als OttoQL-Programm gelöst. Das XQuery-Programm erstreckt sich über 21 Zeilen und ist durch zwei verschachtelte FLWOR-Konstrukte gekennzeichnet. Explizit muss der Benutzer über einen XPath-Ausdruck angeben durch welche Struktur iteriert werden soll und das Ausgabedokument schablonenartig konstruieren.

Im Gegensatz dazu ist das OttoQL-Programm äußerst kurz und begnügt sich mit zwei Zeilen. In Abbildung 4 sieht man wie es in der OttoQL-Oberfläche ausgeführt wurde. Im ersten Schritt wird durch *ext* das Ausgangsdokument gelesen und im zweiten Schritt die Umstrukturierung *gib* ausgeführt. Allein durch die Angabe der Ziel-DTD erzeugt OttoQL die gleiche Ausgabe wie das XQuery-Programm. Für die DTD verwendet OttoQL eine der XML-DTD leicht erweiterte Beschreibung mit vereinfachter Syntax. Die Ziel-DTD $\left\{ \begin{array}{l} results \\ results = M(result) \\ result = author, L(title) \end{array} \right\}$ beschreibt die gewünschte Zielstruktur. Das Zieldokument ist vom Typ *results*, wobei *results* eine **M**enge von *result* ist. Menge bedeutet, dass die Elemente sortiert (in diesem Fall alphabetisch) und Duplikate eliminiert werden sollen. *result* ist ein Tupel aus *author* und *L(title)*. *L(title)* ist eine **L**iste und behält damit seine Ordnung und Elemente.

Die Ausgangs-DTD ist hier bewusst nicht angegeben. Genaue Kenntnis über sie ist für den Anwender nicht notwendig. Er muss nur wissen, welche Namen (also XML-Tags) die Daten haben. Das Programm würde sogar auf unterschiedlich strukturierten Ausgangsdaten funktionieren. In XQuery ließe sich so ein allgemeines Programm gar nicht oder nur sehr schwer realisieren lassen.

In Abbildung 4 sieht man unten die Ausgabe in Tabellenform inklusive der neuen DTD. Genau so kann man das Ergebnis auch als XML-Datei mit XML-DTD ausgeben und abspeichern.

## 4 Weiteres

Die Beispiele können nur einen kleinen Einblick in die Leistungsfähigkeit von OttoQL bieten. Sie kann als domänenspezifische Sprache angesehen werden, dessen Einsatzmöglichkeiten weit über das übliche Maß hinaus gehen. Niemand würde in XQuery weitreichende Berechnungen auf den Daten anfertigen wollen, sondern zu einem weiteren Spezialprogramm greifen. OttoQL nimmt für sich in Anspruch auch hier Lösungen zu bieten.

OttoQL wurde in Ocaml entwickelt, besitzt eine Weboberfläche und kann unter http://otto.cs.uni-magdeburg.de/otto/web/ ausprobiert werden.

## Literatur


[1] K. Benecke: *A Powerful Tool for Object-Oriented Manipulation*, in Proc. IFIP TC2/WG 2.6 Working Conference on Object Oriented Database: Analysis, Design & Construction, Windermere, UK, pp. 95-122, North Holland 1991

[2] K. Benecke: *Strukturierte Tabellen - Ein neues Paradigma für Datenbanken und Programmiersprachen*, Deutscher Universitätsverlag, Wiesbaden 1998

[3] ed.: D. Chamberlain et al., *XML Query Use Cases*, W3C Working Draft 23 March 2007, http://www.w3.org/TR/xquery-use-cases/#xmp-queries-results-q4

# TInTo: A Tool for View-Based Analysis of Stock Market Data Streams

Andreas Behrend, Christian Dorau, and Rainer Manthey

University of Bonn, Institute of Computer Science III
Roemerstr. 164, D-53117 Bonn, Germany
{behrend,dorau,manthey}@cs.uni-bonn.de

**Abstract:** TinTO is an experimental system aiming at demonstrating the usefulness and feasibility of incrementally evaluated SQL queries for analyzing a wide spectrum of data streams. As application area we have chosen the technical analysis of stock market data, mainly because this kind of application exhibits sufficiently many of those characteristics for which relational query technology can be reasonably considered in a stream context. TinTO is a technical investor tool for computing so-called technical indicators, numerical values calculated from a certain kind of stock market data, characterizing the development of stock prices over a given time period. Update propagation is employed for the incremental recomputation of indicator views defined over a stream of continuously changing price data.

## 1 Technical Analysis of Stock Market Data

Technical analysis (TA) is concerned with the prediction of future developments of stock market prices. TA uses so-called technical indicators, numerical values derived from the past development of prices of a certain stock. In principle, indicators are functions applied to the price history of a certain stock and a point in time. A technical analyst is usually interested in the change of indicator values over a certain time period in order to automatically derive buy or sell signals for stocks. The stream of time-stamped price data is an ordered append-only stream provided by a stock market such as the New York stock exchange. In general, indicator computation is based on a division of a stocks price history into consecutive time intervals $t_i$ of equal length. Interval length is user-defined and usually ranges from 1 second to as long as one year. For each time interval, the mean average is determined representing the typical price TP of stock $s$ with respect to $t_i$. A technical indicator definition is based on these values and an additional user-defined parameter $n$ specifying the number of consecutive time intervals under consideration. As an example, consider the commodity channel index (CCI) which is an overbought/oversold indicator, whose value typically oscillates around zero:

$$\mathrm{CCI}_n(s,t_i) := \frac{\mathrm{TP}(s,t_i) - \frac{\sum_{l=0}^{n-1} \mathrm{TP}(s,t_{i-l})}{n}}{\frac{0.015}{n} \cdot \sum_{k=0}^{n-1} |\mathrm{TP}(s,t_i) - \frac{\sum_{l=0}^{n-1} \mathrm{TP}(s,t_{i-l-k})}{n}|}$$

Figure 1: Main window of TinTO

Readings above +100 imply an overbought condition, while readings below -100 imply an oversold situation. When the CCI moves above -100, this is interpreted as the beginning of a positive price trend and, thus, may serve as a buy signal. The position should be closed when the CCI reaches +100 as it enters an overbought condition. CCI values can be analogously used as short selling signals.

Indicator definitions may be based on other ones, forming a kind of indicator hierarchy. In this way, it is possible to combine the positive effects of different indicator types. To which extent technical indicators are meaningful for predicting future price developments, however, remains quite questionable and no statistical significant forecast power has been proved so far. Nevertheless, technical indicators represent a quite general way of analyzing a numerical data stream and can be seen as continuous queries of considerable complexity. As these (potentially recursive) queries are general aggregation functions, they are quite representative for a broad class of stream analysis problems. Although there are various commercial implementations of technical indicators, technical analysts are often interested in building their own indicator system in order to develop a personalized and unique trading strategy. To this end, the appropriateness of new parameter values is investigated and new indicators are invented in order to improve forecast power. A possible way of achieving a flexible and extendible trading system is to implement indicators using a declarative language like SQL.

## 2 The TinTo System

The acronym TInTo is used as an abbreviation for Technical Investor Tool indicating its task as automatic trading system based on indicator signals. Initially, TInTo has been a Visual Basic (VBA) application based on MS Access only, but is currently being reprogrammed as a web-based application using Java and Oracle. TInTo manages a user-defined

portfolio of indices, commodities, and stocks whose historical as well as intraday quotations are provided by `http://finance.yahoo.com` for stock markets worldwide. As a frontend it uses the shareware visualizer ChartDirector [Cha06], a tool supplying a VBA library of well-established methods for drawing financial charts (cf. Figure 1).

TInTo provides a simple SQL view editor for specifying arbitrary technical indicators as predefined SQL queries (i.e., as views), evaluated directly over the underlying database containing timestamped price data. The advantage of view-based indicators is that the underlying definition can be easily recovered and modified while new indicators can be simply defined in form of view hierarchies. In addition, the application of SQL views allows the efficient computation of indicator values directly within the database where the portfolio and price data is usually stored, thus avoiding the well-known impedance mismatch. At present we experiment with some 30 standard indicators as proposed in [BL92] and four new ones where we tried to combine the positive effects of various indicators of different type. For testing the quality of trading signals, an evaluation tool is provided by TInTo which calculates the capital gains and losses achieved when a user trades a chosen commodity according to the signals generated by a selected indicator. To this end, a user must specify a time period over which the corresponding signals are calculated and percentaged gains and losses are accumulated.

Even though a considerable degree of analysis is reachable this way, hardly any streaming is involved yet. However, there are so-called intraday trading strategies which need to access a high frequency stream. The crucial step towards proper stream management in TinTO consisted in the addition of a simple VBA script automatically downloading a record of characteristic values per stock in the portfolio at regular intervals and appending the downloaded data to those already present in the database. This component generates a data stream with a frequency of up to one second pulled from Yahoo on demand.

## 3 Incremental Evaluation of Continuous Queries

The main goal of TinTo is to provide a performance analysis of incrementally evaluated SQL queries over data streams. In data stream research, it is widely believed that conventional relational database systems are not well-suited for dynamically processing continuous queries. We believe, however, that even conventional SQL queries can be efficiently employed for analyzing a wide spectrum of data streams. In particular, we think that the application of incremental update propagation considerably improves the efficiency of computing answers to continuous queries. Update propagation is not a new research topic but has been intensively studied for many years mainly in the context of integrity checking and materialized views maintenance [GM99]. The key idea is to transform each SQL view already at schema design time into a so-called *delta view*, a specialized version of the view referring to changes in the underlying tables, only. The original view definitions are employed only once for materializing their initial answers while the specialized versions are used afterwards for continuously updating the materialized results. Under the assumption that a great portion of the materialized view content remains unchanged, the application of delta views may considerably enhance the efficiency of view maintenance. We adopted

this idea for the TInTo system, using delta-view techniques for a synchronized update of indicator values. To this end, update statements are applied instead of the original indicator views for incrementally maintaining the materialized indicator values. In principle, these update statements can be automatically compiled from the original views. However, we do not have a full-fledged delta compiler for arbitrary SQL views yet, therefore we are performing our experiments with hand-compiled delta views for the time being. In [ABS08], however, the author et al. describe a way of deriving such specialized update statements systematically.

All indicator queries are based on a sliding window defined by the time span for which the user wants to observe indicator values. In TInTo, a user determines the window size by setting the time range attribute which may take values from 1 day to 20 years. As an example, consider CCI values for a period of one year:

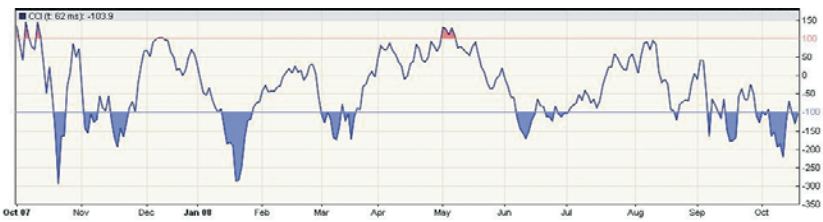



Figure 2: Continuous recomputation of CCI values

As soon as a new current time interval $t_c^{new}$ is considered, the oldest one and its corresponding indicator values are removed from the list of tuples to be displayed, and the new entry $CCI(t_c^{new}) = -103,9$ is calculated for $t_c^{new}$. The time needed for the incremental computation of $CCI(t_c^{new})$ is `62 ms` as indicated in the upper left corner of Figure 2. A corresponding non-incremental computation takes `336 ms` on average. At the demonstration site, we will show this significant performance gain for other window sizes and other types of indicators (e.g. including recursive ones). These measurements show that this kind of performance gain fully scales with the size of the sliding window. The performance results provide first evidence that incremental evaluation of SQL views represents a suitable approach for analyzing a wide spectrum of data streams.

## References

[ABS08] R. Manthey A. Behrend, C. Dorau and G. Schüller. Incremental View-Based Analysis of Stock Market Data Streams. In *IDEAS*, pages 269–275, 2008.

[BL92] C. Le Beau and D. Lucas. *Technical Traders Guide to Computer Analysis of the Futures Markets*. Irwin Professional (USA), 1992.

[Cha06] Chart Director, 2006. [Online; 09.10.2006].

[GM99] Ashish Gupta and Inderpal Singh Mumick. *Materialized Views: Techniques, Implementations, and Applications*. The MIT Press, 1999.

# NexusWeb – eine kontextbasierte Webanwendung im World Wide Space

Andreas Brodt, Nazario Cipriani
IPVS Universität Stuttgart
*Vorname. Nachname*@ipvs.uni-stuttgart.de

**Abstract:** Wir präsentieren NexusWeb, eine kontextbasierte Webanwendung, die ein Umgebungsmodell des Benutzers zeichnet. NexusWeb bezieht dazu Kontextdaten des World Wide Space, einer Föderation verschiedenster Datenanbieter. Die GPS-Position des Benutzers gelangt über eine standardisierte Browsererweiterung in die Webanwendung und steuert die Ansicht von NexusWeb. Wir zeigen verschiedene Szenarien sowohl auf einem mobilen Gerät als auch auf einem gewöhnlichen Laptop.

## 1 Motivation

Kontextbasierte Anwendungen greifen auf Informationen über die momentane Situation des Benutzers [Dey01] zu, um sich entsprechend anzupassen. Ein Beispiel sind ortsbasierte Dienste, die den Benutzer mit Daten seines aktuellen Aufenthaltsorts versorgen. Die Erhebung, Verwaltung und Bereitstellung von Kontextdaten ist meist mit hohem Aufwand verbunden, so dass es nahe liegt, erhobene Kontextdaten in ein allgemeines Weltmodell zu integrieren. Das Nexus-Projekt entwickelte hierfür das Konzept des *World Wide Space* [NGS+01], eine offene, föderierte Umgebung, in der verschiedenste Datenanbieter lokale Kontextmodelle zur Verfügung stellen, die in einer darüber liegenden Föderationsschicht zu einem globalen Kontextmodell integriert werden.

Bisherige Anwendungen für den World Wide Space wurden als schwergewichtige Clients implementiert, die auf dem jeweiligen Zielgerät installiert und konfiguriert werden mussten. In dieser Demonstration stellen wir NexusWeb vor, eine kontextbasierte Webanwendung, die, wie in Abbildung 1 dargestellt, ein Umgebungsmodell des Benutzers im Webbrowser visualisiert. NexusWeb setzt lediglich einen Webbrowser voraus und bedarf keiner Installation. Somit steht NexusWeb auf einer Vielzahl mobiler Systeme als auch auf Desktop-Computern zur Verfügung. Darüber hinaus kann NexusWeb auf lokale Kontextdaten des Clientgeräts zugreifen. Hierzu werden die W3C Delivery Context Client Interfaces (DCCI) [WHR+07] als standardisierte Kontextdatenschnittstelle des Webbrowsers verwendet. In der Demonstration zeigen wir, wie NexusWeb den GPS-Empfänger eines Nokia N810 Internet Tablets benutzt, um die eigene Position sowie umliegende Objekte aus dem World Wide Space auf einer Karte anzuzeigen. Sind DCCI oder Positionsdaten nicht verfügbar, so bleibt dem Benutzer die Möglichkeit, die Position manuell zu selektieren.

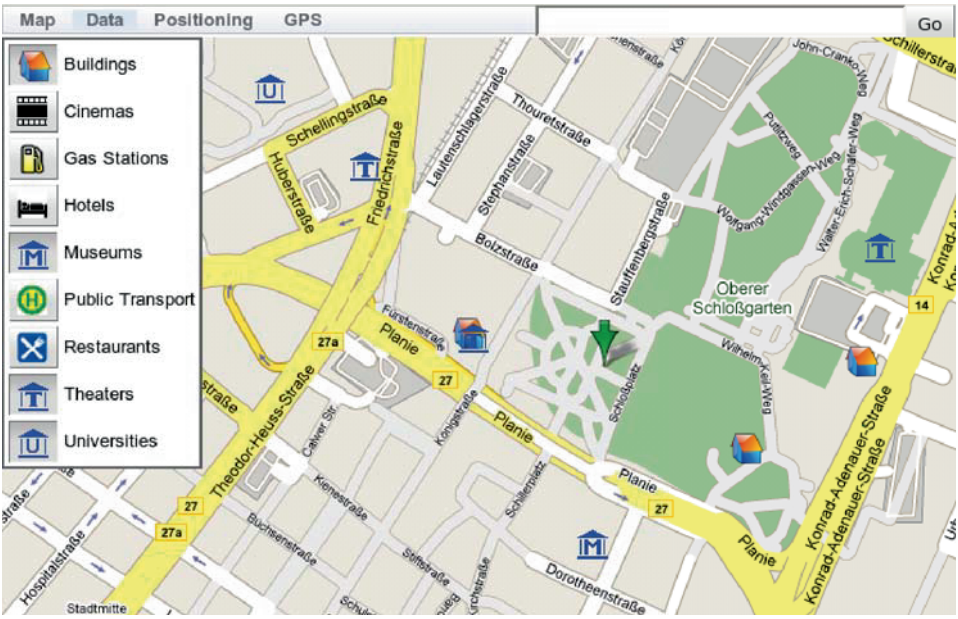



Abbildung 1: Screenshot von NexusWeb

## 2 Architektur

Abbildung 2 zeigt die Systemarchitektur von NexusWeb. NexusWeb residiert als Mehrwertdienst auf einem Föderationsknoten im Nexus-System, das in [NGS+01] beschrieben ist. NexusWeb besteht aus dem NexusWeb Client, den HTML-, CSS- und JavaScript-Dateien, die die Präsentationsschicht und die clientseitige Logik enthalten, sowie aus dem NexusWeb Service API, das die eine Schnittstelle zur Nexus-Föderation anbietet.

Um den NexusWeb Client mit der Position des Benutzers zu versorgen, wird der Mozilla-basierte Webbrowser eines Nokia N810 Internet Tablets um eine Kontextschnittstelle erweitert. Um Kontextdaten jeglicher Art für Webanwendungen verfügbar machen zu können, spezifizierte die W3C die *Delivery Context Client Interfaces* (DCCI) [WHR+07]. Wir benutzen die Telar DCCI-Implementierung aus unserer früheren Arbeit [BNSM08], die als Open Source frei zugänglich ist [Bro08]. Eine Anbindung an den GPS-Empfänger des Internet Tablets stellt die aktuellen Positionsdaten über die DCCI-Erweiterung bereit. Der NexusWeb Client meldet sich beim Start als Listener auf Positionsänderungen an, so dass er die Ansicht bei jeder Positionsänderung aktualisieren und, falls erforderlich, neue Objekte vom Nexus-System nachladen kann. Sind DCCI oder Positionsdaten nicht verfügbar, so bleibt dem Benutzer die Möglichkeit der manuellen Lokalisierung.

Um Objekte des World Wide Space zu laden, stellt der NexusWeb Client asynchrone HTTP-Anfragen an das NexusWeb Service API, das diese in der Augmented World Query Language (AWQL) an die Anfrageverarbeitung der Nexus-Föderation stellt. Die Anfrage-

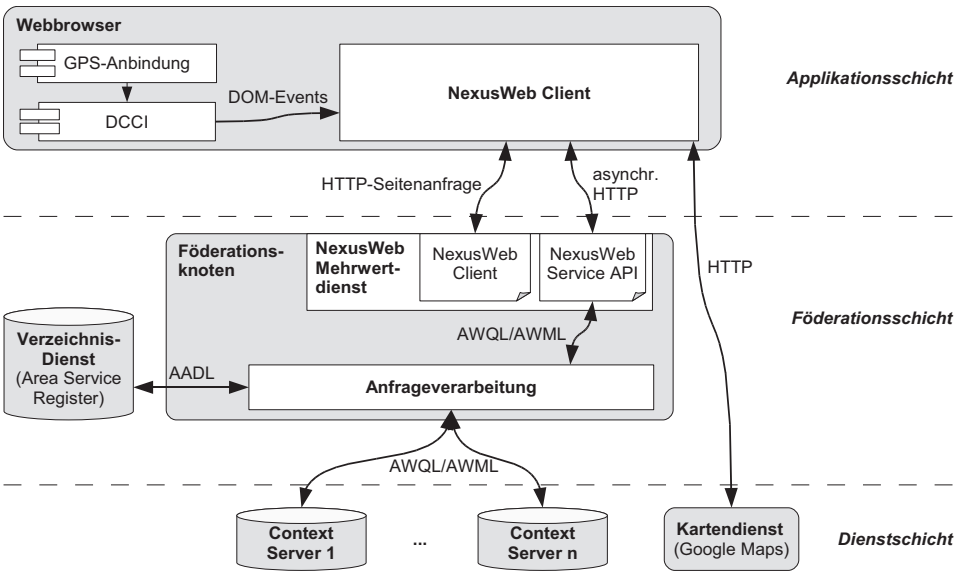



Abbildung 2: Die Systemarchitektur von NexusWeb

verarbeitung ermittelt mit Hilfe des Verzeichnisdienstes (Area Service Register) die von der Anfrage betroffenen Context Server und leitet anschließend diesen die Anfrage weiter. Die Teilergebnisse in Form der Augmented World Modeling Language (AWML) werden in der Anfrageverarbeitung unifiziert und gebündelt über das NexusWeb Service API an den NexusWeb Client gesendet. Der NexusWeb Client visualisiert die Antwort mit Mitteln der Google Maps API.

## 3 Demonstration

Wir demonstrieren vier Szenarien: Simulierte Lokalisierung, manuelle Lokalisierung, Objektselektion und – falls möglich – GPS-basierte Lokalisierung.

**Simulierte Lokalisierung** Um NexusWeb auch an Orten zu demonstrieren, an denen kein GPS-Empfang besteht, kann die GPS-Anbindung im Browser simuliert werden. Hierzu wurde eine weitere Browsererweiterung implementiert, die eine Trajektorie aus einer GPX-Datei einliest und über die DCCI-Erweiterung zeitlich korrekt wiedergibt. Der NexusWeb Client verhält sich dabei, als würde sich der Benutzer entlang der Trajektorie bewegen.

**Manuelle Lokalisierung** Das Szenarium der manuellen Lokalisierung demonstriert den Fall, dass entweder gar kein GPS-Empfänger vorhanden ist, oder dieses momentan keine Positionsdaten liefern kann. Die Ansicht steht auf einer serverseitig konfigurier-

ten Standardposition, kann aber vom Benutzer per Drag and Drop oder durch Eingabe eines Ortsnamens verändert werden. Manuelle Lokalisierung kann auf einem Nokia N810 als auch auf einem Laptop demonstriert werden.

**Objektselektion** Über die Objektselektion kann der Benutzer angeben, welche Objekttypen des World Wide Space angezeigt werden sollen, z. B. Hotels, Museen, etc. Von einer serverseitig konfigurierten Standardeinstellung ausgehend können weitere Objekttypen der Selektion hinzugefügt oder entfernt werden.

**GPS-basierte Lokalisierung:** Falls es am Ort der Demonstration möglich ist, mit dem eingebauten GPS-Empfängers eines Nokia N810 ausreichend GPS-Signale zu empfangen, so kann der NexusWeb Client auch mit echten Positionsdaten demonstriert werden.

## Literatur


[BNSM08] Andreas Brodt, Daniela Nicklas, Sailesh Sathish, and Bernhard Mitschang. Context-Aware Mashups for Mobile Devices. In *Web Information Systems Engineering - WISE 2008, 9th International Conference*, volume 5175 of *Lecture Notes in Computer Science*, pages 280–291, 2008.

[Bro08] Andreas Brodt. Telar DCCI website, 2008. `http://telardcci.garage.maemo.org`.

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[NGS+01] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A Model-Based, Open Architecture for Mobile, Spatially Aware Applications. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 117–135, London, UK, 2001. Springer-Verlag.

[WHR+07] Keith Waters, Rafah A. Hosn, Dave Raggett, Sailesh Sathish, Matt Womer, Max Froumentin, and Rhys Lewis. Delivery Context: Client Interfaces (DCCI) 1.0. Candidate recommendation, W3C, December 2007.

# Ein Tool-Set zur Datenbank-Analyse und -Normalisierung

Daniel Fesenmeyer, Tobias Rafreider, Jürgen Wäsch *
HTWG Konstanz

**Abstract:** In diesem Beitrag werden zwei Softwarewerkzeuge zur Datenbank-Analyse und -Normalisierung vorgestellt. *TANE-java* dient zur Extraktion von funktionalen Abhängigkeiten aus relationalen Datenbanken. *DBNormalizer* dient zur Normalisierung relationaler Datenbanken auf Basis funktionaler Abhängigkeiten. Ergebnis ist ein ausführbares SQL-Skript zur Schemamodifikation und Datenmigration. Die Werkzeuge können in der Datenbank-Ausbildung, aber auch in realen Projekten zum Refactoring existierender Datenbanken eingesetzt werden.

## 1 Einleitung und Motivation

Ein schwierig zu erfassendes Themengebiet im Bereich der relationalen Datenbanken ist die relationale Entwurfstheorie. Kommerzielle Datenbankentwurfswerkzeuge unterstützen den formalen Normalisierungsprozesse nicht oder nur unzureichend [F08, R08]. Die im Rahmen von [F08] analysierten freien "akademischen" Normalisierungswerkzeuge Database Normalizer (TU München [J04]) und Database Normalization Tool (Cornell University [S03]) arbeiten nicht auf realen Datenbanken, sondern nur auf vom Benutzer einzugebenden Relationenschemata. In diesem Beitrag werden zwei Softwarewerkzeuge zur Analyse und Normalisierung von relationalen Datenbanken auf Basis von funktionalen Abhängigkeiten vorgestellt: *TANE-java* und *DBNormalizer*.

*DBNormalizer* dient zur Normalisierung von relationalen Datenbanken und wird u.a. in der Datenbank-Ausbildung an der HTWG Konstanz verwendet, um den Studierenden die relationale Entwurfstheorie praktisch zu vermitteln. Neben der Nutzung in der Lehre ist aber auch der Einsatz in realen Projekten möglich. So können z.B. das Forward Engineering sowie die Analyse und das Refactoring von existierenden relationalen Datenbanken [AS06] im Rahmen von agilen Prozessen [A03] unterstützt werden.

Ein häufiger Kritikpunkt gegen den praktischen Einsatz der relationalen Entwurfstheorie ist der Aufwand zur Bestimmung von funktionalen Abhängigkeiten. Hier setzt das Werkzeug *TANE-java* an. Es extrahiert automatisch funktionale Abhängigkeiten aus einer relationalen Datenbankausprägung. Diese extrahierten funktionalen Abhängigkeiten können dann die Basis für die Normalisierung mittels *DBNormalizer* bilden. Ergebnis ist ein SQL-Skript für die Schemamodifikation und die Datenmigration, das dann auf der relationalen Datenbank ausgeführt werden kann (siehe Abbildung 1).

*Die Softwarewerkzeuge wurden im Rahmen von Diplomarbeiten an der HTWG Konstanz entwickelt [F08, R08]. Daniel Fesenmeyer arbeitet heute als Software Entwickler bei der Sybit GmbH.
*Kontakt:* juergen.waesch@htwg-konstanz.de

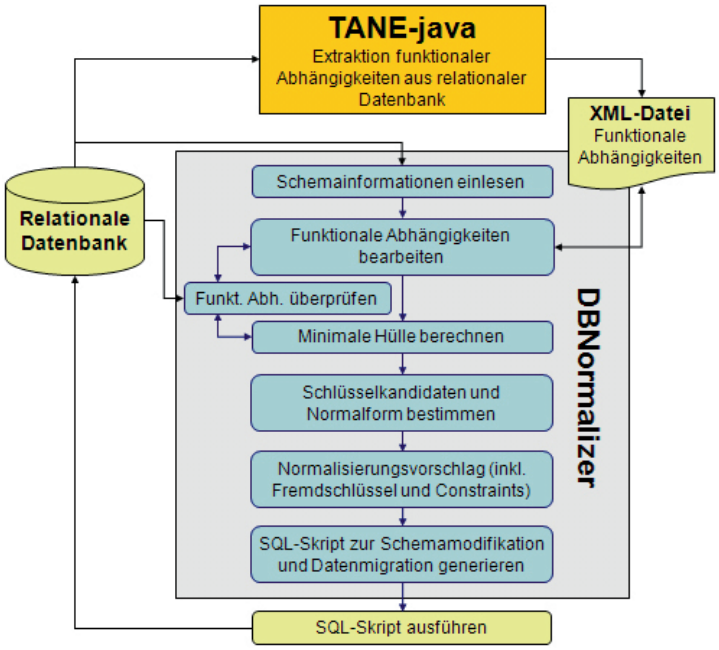



Abbildung 1: Normalisierungsprozess: Zusammenspiel zwischen *TANE-java* und *DBNormalizer*.

## 2 DBNormalizer

*DBNormalizer* ermöglicht die Normalisierung von Datenbankschemata bis zur Boyce-Codd-Normalform (BCNF). Das Werkzeug arbeitet auf realen Datenbanken, berücksichtigt im Gegensatz zu [J04, S03] auch Fremdschlüsselabhängigkeiten zwischen Relationen bei der Normalisierung und generiert einen Vorschlag zur Schemamodifikation als ausführbares SQL-Skript. Abbildung 2 zeigt die graphische Benutzungsoberfläche von *DBNormalizer*. Das Werkzeug basiert primär auf einem Synthesealgorithmus; weitere Details zur Realisierung sind in [WFR08, F08] zu finden. Im Augenblick bietet *DBNormalizer* folgende Funktionalität:

- Einlesen von Schemainformationen aus einer gegebenen relationalen Datenbank bzw. Eingabe und Bearbeiten von Datenbankschemata durch den Benutzer, falls keine relationale Datenbank verfügbar ist.
- Eingabe und Verwaltung von funktionalen Abhängigkeiten für die Relationen durch den Benutzer sowie Überprüfung der Gültigkeit von funktionalen Abhängigkeiten auf Basis der augenblicklichen Datenbankausprägung (falls vorhanden).
- Berechnung einer minimalen Hülle für funktionale Abhängigkeiten von Relationen, Berechnung von Attributhüllen, sowie Bestimmung *aller* Schlüsselkandidaten.
- Bestimmung der Normalform von Relationen (1NF, 2NF, 3NF oder BCNF). Dies beinhaltet die Berechnung, welche funktionalen Abhängigkeiten in einer Relation eine bestimmte Normalform verletzen.

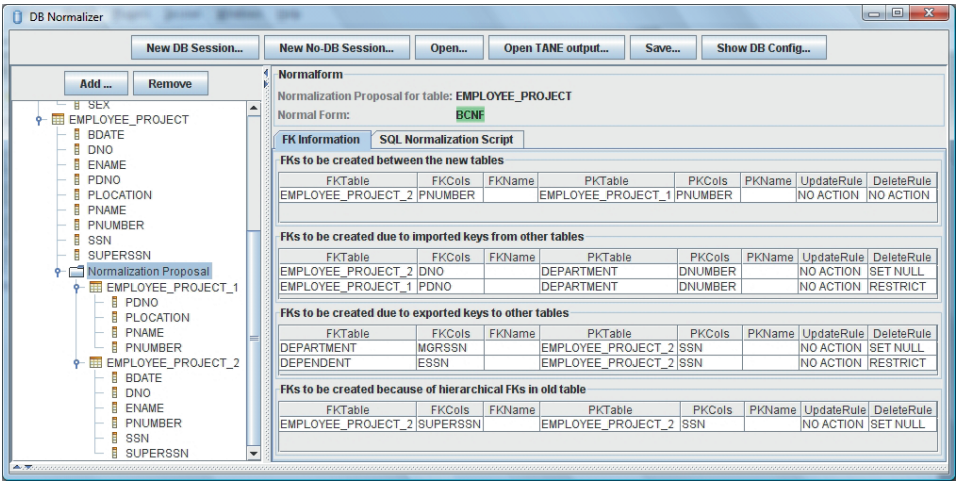



Abbildung 2: Graphische Benutzungsoberfläche von *DBNormalizer*.

- Erzeugen eines Normalisierungsvorschlags für ein Datenbankschema. Dies beinhaltet die Berechnung *aller* Schlüsselkandidaten für die neuen Relationen, die Berechnung der Fremdschlüsselbeziehungen für die Relationen des Normalisierungsvorschlags und die Bestimmung der Normalformen für die neuen Relationen (3NF oder BCNF).
- Erzeugung eines ausführbaren SQL-Skripts zur Transformation der Datenbank. Das SQL-Skript beinhaltet die Erzeugung der neuen Tabellen, die Datenmigration, das Löschen der alten Tabellen und das Erzeugen der Fremdschlüsselconstraints für die neuen Tabellen. Außerdem werden Views erzeugt, um existierende Anwendungen mit dem neuen Datenbankschema weiterhin nutzen zu können bzw. um die Portierung der Datenbankanwendungen zu vereinfachen. Im Falle von Oracle könnten für nicht-aktualisierbare Views auch INSTEAD-OF Trigger erzeugt werden.
- Import und Export von funktionalen Abhängigkeiten und Datenbankschemata als XML-Datei sowie Import der mittels *TANE-java* automatisch ermittelten funktionalen Abhängigkeiten einer relationalen Datenbankausprägung.

## 3 TANE-java

*TANE-java* ist ein Software-Werkzeug zur Extraktion von funktionalen Abhängigkeiten aus (großen) Datenbankausprägungen. Die extrahierten funktionalen Abhängigkeiten können als Eingabe für den Normalisierungsprozess in *DBNormalizer* dienen. *TANE-java* implementiert u.a. die in [H+99] vorgestellten Algorithmen zur Extraktion von funktionalen Abhängigkeiten. Diese Algorithmen sind durch das zur Suchraumeinschränkung verwendete Partitionierungsprinzip sehr effizient und ermöglichen auch die Extraktion von approximativen funktionalen Abhängigkeiten [H+99, R08].

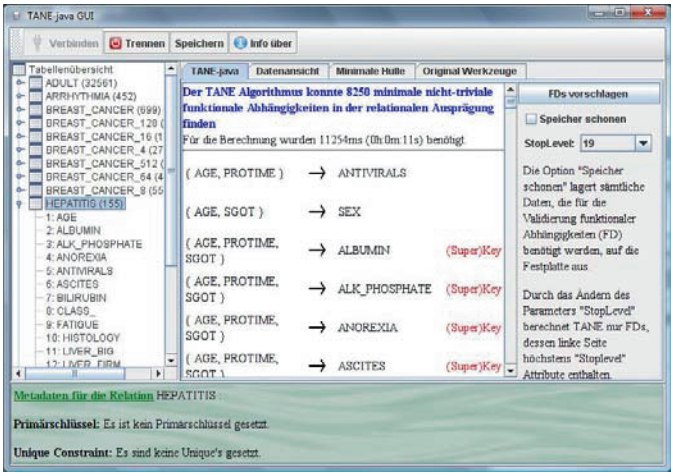



Abbildung 3: Graphische Benutzungsoberfläche von *TANE-java*.

Im Gegensatz zu der ursprünglichen C-Implementierung von TANE [TH] arbeitet *TANE-java* auf relationalen Datenbanken (nicht auf CSV-Dateien) und bietet eine graphische Benutzungsoberfläche (siehe Abbildung 3). Neben der Java-Implementierung der TANE-Algorithmen integriert das Werkzeug aber auch die ursprüngliche C-Implementierung. Hierzu werden zuerst die Daten aus der Datenbank in CSV-Dateien exportiert und dann die C-Programme gestartet. Diese wurden so modifiziert, dass sie nun ihre Ergebnisse in einem XML-Format liefern, das wiederum in *DBNormalizer* weiterverarbeitet werden kann (siehe Abbildung 1). Weitere Details zur Realisierung von *TANE-java* sind in [R08] zu finden.

## Literatur


[AS06] S.W. Ambler, P.J. Sadalage: Refactoring Databases - Evolutionary Database Design. Addison-Wesley, 2006.

[A03] S.W. Ambler: Agile Database Techniques. Wiley, 2003.

[F08] D. Fesenmeyer: Design und Implementierung eines Software-Tools zur Normalisierung relationaler Datenbanken. Diplomarbeit, HTWG Konstanz, 2008.

[H+99] Y. Huhtala, Y. Kärkkainen, P. Porkka, H. Toivonen: TANE – An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal, Vol. 42, No. 2, 1999.

[J04] E. Jürgens: Database Normalizer. Technische Universität München. http://home.in.tum.de/juergens/DatabaseNormalizer/index.htm/

[R08] T. Rafreider: Data-Mining Verfahren und Tools zur Unterstützung der Datenbankanalyse und Schemaoptimierung. Diplomarbeit, HTWG Konstanz, 2008.

[S03] S. Selikoff: The Database Normalization Tool. Educational Database Tools, Cornell University, 2003. http://dbtools.cs.cornell.edu/norm_index.html

[TH] TANE Homepage. http://www.cs.helsinki.fi/research/fdk/datamining/tane/

[WFR08] J. Wäsch, D. Fesenmeyer, T. Rafreider: Ein Normalisierungswerkzeug für die Datenbank-Ausbildung. Herbsttreffen der GI-Fachgruppe Datenbanken, Düsseldorf, 2008.

# Guided Navigation basierend auf SAP Netweaver BIA

Bernhard Jäksch, Robert Lembke, Barbara Stortz,
Stefan Haas, Anja Gerstmair, Franz Färber

SAP Netweaver BI - Walldorf
{b.jaeksch, r.lembke, barbara.stortz, stefan.haas, anja.gerstmair, franz.faerber}@sap.com

**Kurzfassung:** Interaktives Online Analytical Processing mit klassischen Operationen wie Drill-Down, Roll-Up und Pivoting gehören mittlerweile zum Standard-Repertoire von Analysewerkzeugen. Der Benutzer wird dabei auf sogenannten Drillpfaden (meistens als Hierarchien und Dimensionen bezeichnet) durch den Datenbestand geführt. Das Prinzip des „Guided Navigation" eröffnet eine alternative Art durch Datenbestände zu navigieren und Zusammenhänge zwischen einzelnen Dimensionen innerhalb eines Datenwürfels zu erkennen. Der Nachteil des Guided Navigation aus Sicht der Datenbanksysteme liegt jedoch in einer im Vergleich zum klassischen OLAP signifikant höheren Aggregationsleistung in allen Dimensionen pro Benutzerinteraktion. In dieser Demo wird ein System vorgestellt, welches das Prinzip der Guided Navigation auf Benutzeroberfläche umsetzt und basierend auf dem SAP Netweaver BIA die Aggregationsleistung erfüllt, die notwendig ist um ein interaktives Analyseverhalten auf großen Datenbeständen zu ermöglichen.

## 1 Einleitung

Die Definition von Hierarchien auf dimensionalen Strukturen stellt aktuell den Status Quo aller OLAP-Werkzeuge zur interaktiven Analyse von großen Datenbeständen dar. Entlang vordefinierter Drillpfade wird es dem Benutzer erlaubt, Datenbestände zu verfeinern (Drill-down) oder höhere Aggregate zu berechnen (Roll-Up). Der zentrale Nachteil dieser Interaktionsmethode besteht darin, dass der Benutzer vorab erahnen muss, hinter welchem Aggregat sich ein interessanter Zusammenhang versteckt um eine Drill-Down-Aktion auf dieses Aggregat auszulösen.

Das Prinzip der Guided Navigation geht einen alternativen Weg und ermöglicht dem Benutzer eine aufgefächerte Sichtweise auf die Dimensionen eines Datenwürfels. Die Navigation erfolgt dabei nicht mehr nur ausschließlich über vordefinierte Hierarchien, sondern über mögliche Zusammenhänge in den Fakt-Daten. Der Nutzwert der Guided Navigation wird insbesondere durch eine Aussage von Joseph Busch (http://www.taxonomystrategies.com) mit folgender Aussage unterstrichen: "Four independent categories [facets] of 10 nodes each can have the same discriminatory power as one hierarchy of 10,000 nodes." Konkret soll folgendes Beispiel, welches ebenfalls im Rahmen der Demo zur Veranschaulichung des Analyseprinzips und der dahinter stehenden Datenbanktechnik benutzt wird, das allgemeine Prinzip nochmals verdeutlichen:

Das Beispielszenario reflektiert einen Online-Shop für Bekleidung (e-Fashion) mit dimensionalen Attributen wie Jahr, Monat und Produktinformationen auf Artikel-, Gruppen- und Kategorienebene. Als Kennzahlen sind in dem multidimensionalen Schema Umsatzzahlen, etc. definiert, die auch gegen eine Geschäftsdimension (City, State-Ebene) ausgewertet werden können. Das Schema, d.h. die Kennzahlen und die zur Guided Navigation zur Verfügung stehenden Attribute werden in sogenannten InfoSpaces definiert. Abbildung 1 gibt einen Einblick in die Auswahl und Definition von InfoSpaces für das konkrete e-Fashion-Szenario.

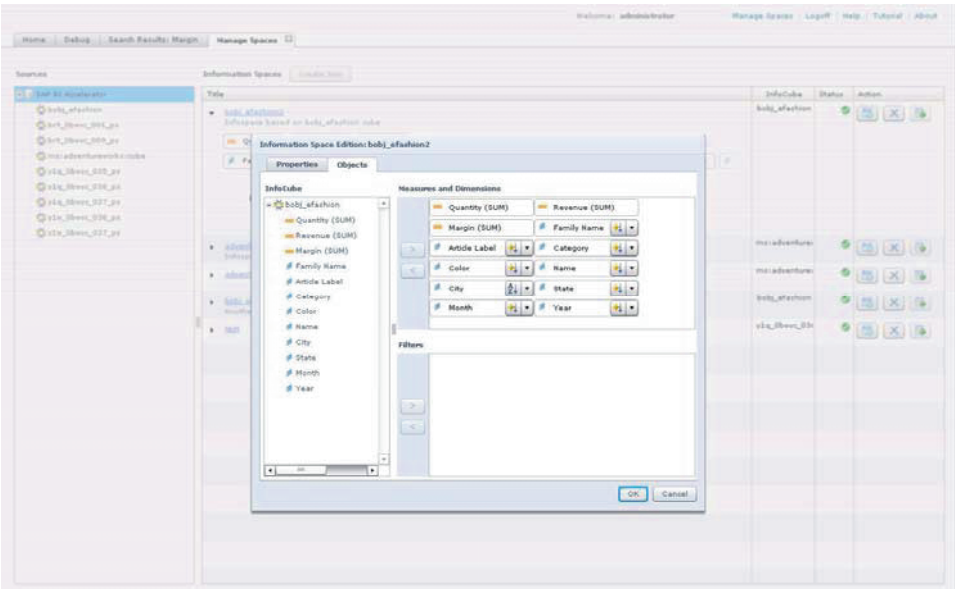



**Abbildung 1: Auswahl und Definition von InfoSpaces**

Das Prinzip der Navigation ist in Abbildung 2 dargestellt. Dabei werden Einschränkungen auf einzelnen beschreibenden Attributen vorgenommen (Jahr 2006, 2007). Das System ermittelt im Hintergrund zwei Ergebnisaspekte. Zum einen werden die dadurch ausgewählten Kennzahlen durch eine (normale, OLAP-übliche) Aggregation ermittelt und graphisch in unterschiedlichsten Formen dargestellt. Zum andern – und dies ist der wesentliche Vorteil einer Guided Navigation - werden nur die Ausprägungen in den verbleibenden Dimensionen ermittelt, die tatsächlich über mindestens einen Faktdatensatz zur den ausgewählten Dimensionsbereichen aufweisen. Im konkreten Szenario bleiben nur die Werte in den nicht von einem Benutzer selektierten Attributen (wie beispielsweise Produktfamilie) erhalten, die mindestens einen Verkauf in den Jahren 2006 und 2007 aufweisen konnten. Die Guided Navigation unterstützt somit nicht nur eine explizite Selektion durch den Benutzer (wie im OLAP üblich) sondern auch eine implizite Selektion durch eine Auswertung der Beziehung über die Faktdaten. Insbesondere bei dünnbesetzten Datenwürfeln hilft diese Art der Navigation enorm.

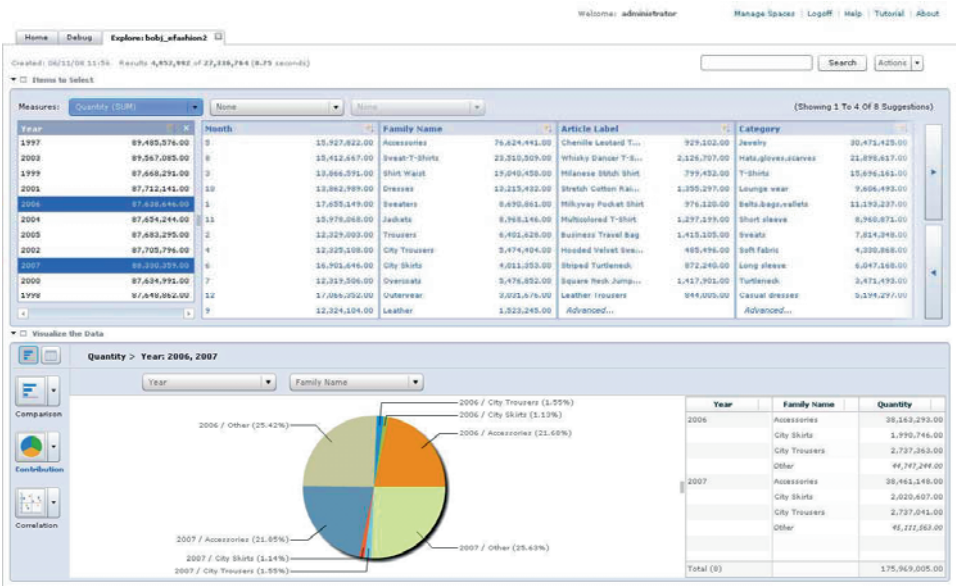



**Abbildung 2: Auswahl und implizite Selektion über Faktdatensätze**

Die Herausforderung für die Datenbankseite besteht nun darin, bei jeder Benutzerinteraktion pro beteiligtes Attribut zu bestimmen, wie oft eine Beziehung zum eingeschränkten Datenwürfel noch existiert. Jede Interaktion impliziert dadurch eine Gruppierung pro Dimensionsattribut; bei hoch-dimensionalen Würfel müssen entsprechend viele Aggregationsanfragen abgesetzt werden.

## 3 Datenbanktechnologie

Das Prinzip der Guided Navigation wird basierend auf der Datenbank-Engine SAP Netweaver BIA demonstriert. Dabei handelt es sich um ein hoch-paralleles System, welches relationale Star- und Snowflake-Schemata spaltenorientiert organisiert. Um eine hohe Anfrage-Performance zu erreichen werden die Daten auf der einen Seite zusätzlich zur vertikalen Spaltenaufteilung noch horizontal partitioniert und die dadurch entstandenen Partitionen auf unterschiedliche Rechnerknoten allokiert um eine maximale Parallelisierung von Datenbankanfragen zu erzielen. Das Produkt wird bis zu einer Größe von standardmäßig 32 Rechnerknoten mit je 2 CPU/4cores (Harpertown) vorkonfiguriert ausgeliefert. In einer Laborumgebung in Zusammenarbeit mit dem Hardware-Partner IBM wurde eine lineare Skalierung bis zu 140 Knoten nachgewiesen (http://www.sap.com/platform/netweaver/pdf/BWP_BI_Accelerator_WinterCorp.pdf).

Auf der anderen Seite verfolgt der SAP Netweaver BIA die Philosophie der Hauptspeicherdatenbanken, in dem der gesamte Datenbestand in komprimierter Form im Hauptspeicher gehalten werden kann. Zugriff auf Disk erfolgt nur beim initialen Befüllen der Hauptspeicherstrukturen nach dem Start. Die unterschiedlichen Komprimierungsverfahren basieren auf einem verzeichnisbasierten Ansatz, wobei die

eigentlichen Werte in einem Dictionary gehalten werden. Die Daten selbst, d.h. die Einträge einer Spalte einer Relation, bestehen dann entsprechend nur noch aus einer Sequenz von Codes, die auf den jeweiligen Eintrag im Verzeichnis verweisen. Der SAP Netweaver BIA selbst läuft des Weiteren als spezifische Konfiguration des SAP TREX Projektes, in welchem neben der BIA-Engine weitere Module spezifische Aufgaben wie Textsuche, Text-Mining etc. übernehmen können. Abbildung 3 gibt einen Überblick über die Komponenten des Gesamtsystems.

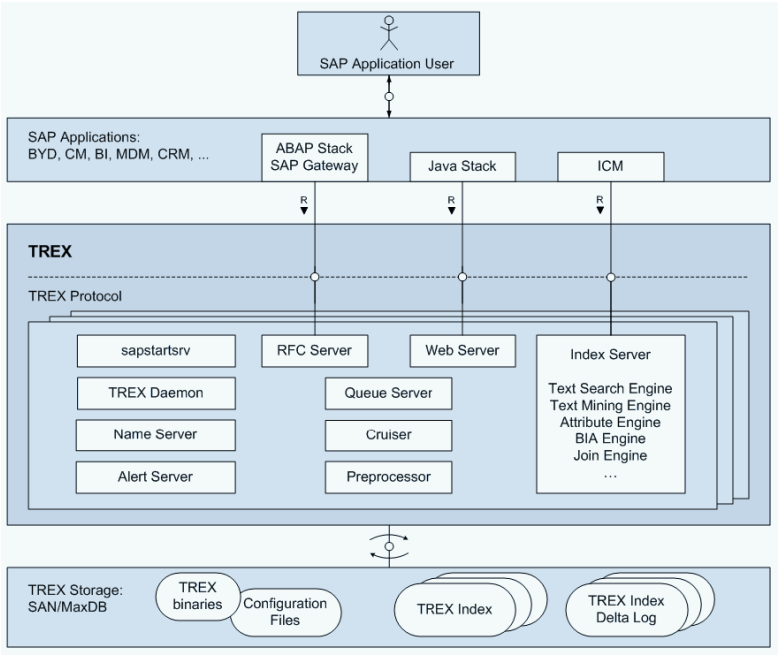



**Abbildung 3: Architektur der TREX-Infrastruktur**

## 2 Demonstration

In der Demo wird zum einen das User-Interface für eine Guided Navigation gezeigt. Dabei handelt es ich um eine vollständig Browser-basierte Benutzeroberfläche mit dem Komfort einer Rich Client-Applikation Zum anderen wird die Technologie des SAP Netweaver BIA erklärt. Der BIA ist eine spaltenorientierte Daten-Analyse-Engine, die verteilt über mehrere Rechnerknoten Datenbestände in komprimierter Form im Hauptspeicher hält und parallele Aggregationsoperation ausführt. Im Rahmen der Demo wird ein BIA auf einem lokalen System gezeigt. Zusätzlich besteht (bei Internetverbindung) die Möglichkeit auf große Datenbestände (>1Mrd Fakten) zuzugreifen und die Skalierbarkeit des BIAs durch die hochgradige Interaktion zu demonstrieren. In der Diskussion mit den Interessierten Besuchern werden entsprechende Konzepte (Komprimierung, Parallelisierung, etc.) erläutert, die diese Form der Interaktion ermöglichen.

# Ein kooperativer XML-Editor für Workgroups*

Francis Gropengießer, Katja Hose, Kai-Uwe Sattler
{francis.gropengiesser|katja.hose|kus}@tu-ilmenau.de

**Abstract:** In vielen Anwendungsgebieten, z.B. im Design oder in Medienproduktionsprozessen, hat sich XML als Format für den Datenaustausch etabliert. Zur Verarbeitung von XML-Daten in Mehrbenutzerumgebungen sind spezielle Werkzeuge nötig. Allerdings berücksichtigen die derzeit auf dem Markt verfügbaren Editoren nur unzureichend die Anforderungen kooperativer Arbeitsumgebungen. In dieser Arbeit wird daher ein transaktionsbasierter XML-Editor vorgestellt, der diese Anforderungen erfüllt. Er ist für den Einsatz in eng gekoppelten Systemumgebungen, auch Workgroups genannt, konzipiert. Der Editor erlaubt ein intuitives Bearbeiten von XML-Daten und macht Änderungen eines Nutzers an den Daten anderen Nutzern sehr früh zugänglich. Weiterhin ermöglicht er durch den Einsatz eines intelligenten Sperrprotokolls einen hochgradig parallelen Arbeitsprozess.

## 1 Einleitung

In vielen Anwendungsbereichen hat sich XML als Format für den Datenaustausch etabliert. Ein Beispiel ist der Postproduktionsprozess von Filmton mit dem räumlichen Soundsystem IOSONO (*http://www.iosono-sound.com/*). Die von den Autoren erstellten Soundszenen werden als XML-Dateien gespeichert. Das Ziel unserer Arbeit ist es, ein kooperatives Arbeiten von mehreren Autoren an denselben Soundszenen bzw. denselben XML-Dateien zu ermöglichen. Kooperativität bedeutet dabei, dass alle Autoren gleichberechtigt sind und jeder von ihnen Kenntnis über den aktuellen Stand des Projekts besitzt. Jeder Autor ist berechtigt, Lösungsansätze zum Projekt beizutragen, die wiederum von anderen Autoren auf Korrektheit überprüft werden. Es findet daher ein Informationsaustausch in beliebiger Richtung zwischen den Autoren statt.

Die aus der Forderung nach Kooperativität resultierenden Anforderungen, z.B. multidirektionaler Informationsfluss und frühe Sichtbarkeit von Änderungen am Datenbestand, sind nur schwer zu erfüllen. Beispielsweise bedeutet die Forderung nach einem beliebig gerichteten Informationsaustausch einen Verzicht auf Serialisierbarkeit. Diese dient jedoch den traditionellen Transaktionsmodellen als Korrektheitskriterium. Um alle Autoren immer auf dem aktuellen Stand des Projekts zu halten, sind Versionierungslösungen wie SVN oder CVS unzureichend, da in diesen Systemen Änderungen nur durch den expliziten Aufruf von Check-in und Update propagiert werden. In [GS08] ist eine detaillierte Beschreibung aller Anforderungen kooperativer Umgebungen sowie eine Einsatztauglichkeitsanalyse bekannter Transaktions- und Workflowmodelle zu finden.

*Das diesem Beitrag zugrunde liegende Projekt wurde mit Mitteln der DFG unter dem Föderkennzeichen SA782/15-1 gefördert.

In dieser Arbeit wird ein neuartiger XML-Editor präsentiert, der den Anforderungen kooperativer Medienproduktionsprozesse in eng gekoppelten Systemumgebungen (Workgroups) gerecht wird. Er stellt ein Konglomerat aus (i) klassischen Datenbanktechniken, wie z.B. Transaktionen, Sperrprotokolle und Transaktionsrecoverymodelle, (ii) einem grafischen XML-Editor, wie z.B. XMLSpy (*http://www.altova.com/de/*) und (iii) einem Kollaborationsmechanismus, ähnlich dem in SubEthaEdit (*http://www.codingmonkeys.de/subethaedit/index.de.html*), dar.

## 2 Systemarchitektur

Für die Modellierung einer Workgroup wurde eine *Client-Server-Architektur* gewählt, in der mehrere Nutzer (*Clients*) permanent mit einem zentralen Repository für die XML-Daten (*Server*) verbunden sind (Abbildung 1).

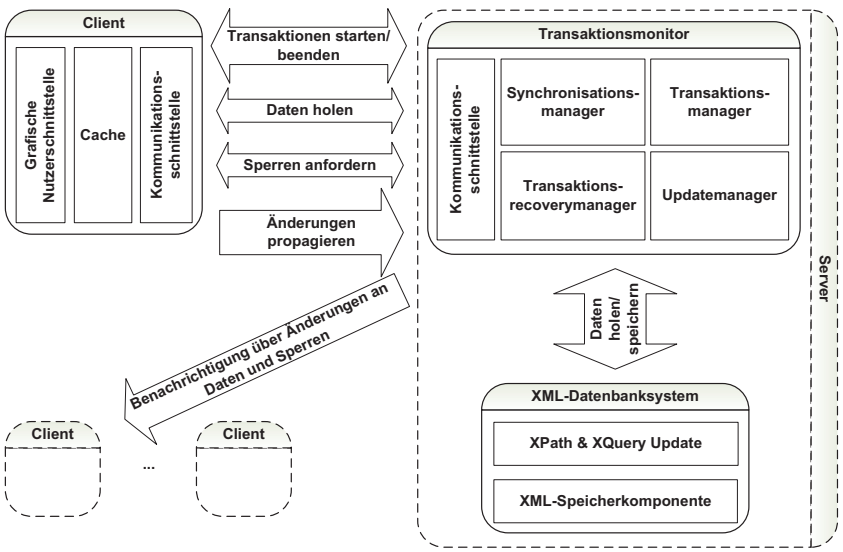



Abbildung 1: Systemarchitektur

Der *Server* besteht aus 2 Hauptkomponenten – dem *Transaktionsmonitor* und dem *XML-Datenbanksystem*. Der Transaktionsmonitor enthält vier Hauptkomponenten. Der *Transaktionsmanager* implementiert unser kooperatives Transaktionsmodel [GS08] basierend auf geschachtelten dynamischen Aktionen [NW94]. Weiterhin verwaltet er die Transaktionen aller verbundenen Clients. Der *Synchronisationsmanager* realisiert unser kooperatives Sperrprotokoll [GS08], das die Semantik der Operationen auf der Baumrepräsentation von XML-Dokumenten ausnutzt. Der *Transaktionsrecoverymanager* implementiert eine Transaktionsrecoverystrategie basierend auf einer Logging-Technik. Er wird benötigt, um Transaktionsfehler und Transaktionsabbrüche zu behandeln. Der *Updatemanager* realisiert das Publisher-Subscriber-Pattern. Jeder Client registriert sich beim Updatemanager, um über Änderungen jeglicher Art, z.B. an den Daten oder Sperren, benachrichtigt zu

werden. Die Registrierung erfolgt automatisch, sobald der Nutzer einen Teil eines XML-Dokuments herunterlädt, den er bearbeiten möchte. Treten Änderungen bezüglich dieses Dokumentteils auf, so wird der Nutzer darüber sofort in Kenntnis gesetzt.

Um XML-Daten dauerhaft zu speichern, macht der Transaktionsmonitor Gebrauch von einem *XML-Datenbanksystem*. Um Daten vom Server zu erhalten / anzufragen, wird *XPath* genutzt. Änderungen werden mit *XQuery Update* geschrieben. Da der Transaktionsmonitor lediglich die Behandlung von Transaktionsfehlern und Transaktionsabbrüchen übernimmt, bietet die *XML-Speicherkomponente* Möglichkeiten für ein Systemrecovery. Weiterhin übernimmt das XML-Datenbanksystem eine Transformation der XML-Daten in eine spezielle Baumstruktur [HH03, GS08], die die Grundlage für alle Verarbeitungsschritte bildet.

Die Clients besitzen einen lokalen *Cache*, in dem sie Kopien von den Teilen der XML-Dokumente speichern, die sie bearbeiten wollen. Alle Operationen werden zunächst auf diesen lokalen Kopien ausgeführt. Dies hat den Vorteil, dass im Falle eines Transaktionsabbruchs noch keine Daten zum Server gesendet wurden. Um ein komfortables Arbeiten zu ermöglichen, bietet der Client eine *grafische Nutzeroberfläche* (Abbildung 2). Diese stellt die zu bearbeitenden XML-Daten als Baumstruktur dar. Attributwerte oder Text können sehr einfach editiert werden, indem der betroffene Knoten selektiert wird und die Änderungen vorgenommen werden. Einzelne Knoten oder Teilbäume können ganz leicht per Drag'n'Drop verschoben werden. Weiterhin bietet diese Nutzerschnittstelle alle nötigen Funktionen an, um eine Verbindung zum Server herzustellen und Kopien der zu bearbeitenden Daten zu beziehen.

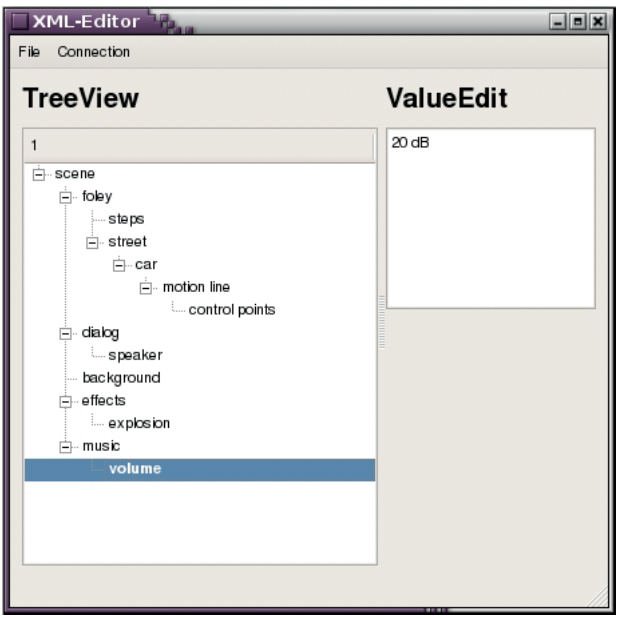



Abbildung 2: Grafische Nutzerschnittstelle

## 3 Eigenschaften

Ziel des in dieser Arbeit vorgestellten XML-Editors ist es, wie bereits in der Einleitung erwähnt, die kooperative Verarbeitung von XML-Daten in Mehrbenutzer-Medienproduktionsprozessen zu unterstützen. Um diesen Zweck zu erfüllen, bietet er folgende Eigenschaften:

- Änderungen eines Nutzers werden sofort anderen Nutzern angezeigt. Dadurch besitzen alle Nutzer immer Kenntnisse über den aktuellen Stand des Projekts.
- Ein Nutzer kann sehen, welche Teile eines Dokuments von anderen Nutzern zeitgleich bearbeitet werden. Dies wird ihm durch farbliche Hervorhebungen in der grafischen Repräsentation der Daten angezeigt.
- Nutzer können einzelne Arbeitschritte verwerfen, ohne ihren gesamten Arbeitsfortschritt zu verlieren.
- Das verwendete Sperrprotokoll fordert keine strikte Serialisierbarkeit, wodurch ein Informationsaustausch in beliebiger Richtung möglich ist. Es nutzt die Semantik der Operationen auf der speziellen Baumstruktur der XML-Daten aus und erzielt dadurch hochgradig parallele Arbeitsabläufe.

## 4 Demonstration

Bei der Demonstration wird ein Workgroup-Szenario im kleinen Rahmen realisiert. Mehrere Benutzer (z.B. die Teilnehmer der Konferenz) können gleichzeitig eine Soundszene bearbeiten. Dabei wird der Umgang mit der grafischen Oberfläche und dem System im Allgemeinen verdeutlicht. Dies umfasst grundlegende Schritte wie Verbinden mit dem Server, Herunterladen der zu bearbeitenden Daten und Ausführung von Operationen (Updates) auf den Daten. Ein besonderer Schwerpunkt liegt darauf, die Kooperativität des Systems zu demonstrieren. Es wird gezeigt, wann Änderungen propagiert werden, wie bei Transaktionsabbrüchen verfahren wird und wie Aktivitäten anderer Nutzer grafisch visualisiert werden. Weiterhin soll gezeigt werden, warum es vorteilhaft ist, Transaktionen, konkret unser spezielles Transaktionsmodell, zu verwenden. Ein letzter Schwerpunkt liegt auf der Demonstration unseres Sperrprotokolls. Unter anderem wird gezeigt, wie die Semantik der Operationen auf der speziellen Baumstruktur ausgenutzt wird.

## Literatur


[GS08] Francis Gropengießer und Kai-Uwe Sattler. An Extended Cooperative Transaction Model for XML. In *PIKM'08 icw CIKM'08*, Seiten 41–48, 2008.

[HH03] Michael Peter Haustein und Theo Härder. taDOM: A Tailored Synchronization Concept with Tunable Lock Granularity for the DOM API. In *ADBIS*, Seiten 88–102, 2003.

[NW94] Edgar Nett und Beatrice Weiler. Nested Dynamic Actions - How to Solve the Fault Containment Problem in a Cooperative Action Model. In *Symposium on Reliable Distributed Systems*, Seiten 106–115, 1994.

# MyMIDP and MyMIDP-Client: Direct Access to MySQL Databases from Cell Phones

Hagen Höpfner and Jörg Schad and Sebastian Wendland and Essam Mansour

International University in Germany
School of Information Technology
Campus 3, D-76646 Bruchsal, Germany

*hoepfner@acm.org* {*Joerg.Schad*|*Sebastian.Wendland*|*Essam.Mansour*}*@i-u.de*

## 1 Introduction and Motivation

Cell phones are no longer merely used to make phone calls or to send short or multimedia messages. They more and more become information systems clients. Recent developments in the areas of mobile computing, wireless networks and information systems provide access to data at almost every place and anytime by using this kind of lightweight mobile device. But even though mobile clients support the Java Mobile Edition or the .NET Micro Framework, most information systems for mobile clients require a middleware that handles data communication. Oracle Lite [Ora04a, Ora04c, Ora04b] and IBM's DB2 Everyplace [IBM04a, IBM04b] use a middle-ware approach for synchronizing data between client and server. Microsoft's SQL Server CE [Mic08] needs Active Sync and Sybase Adaptive Server Anywhere [Syb08] either uses SQL-Remote and its message oriented replication or MobiLink as a session based approach. All these systems are designed for handling replicated data [KRTH07] but not for simple client/server data access. In previous works [CIIH07, ICH07] we used a simple web service that forwards queries to the server and returns the result to the requesting client using an HTTP-connection. However, this approach is comparable to the middle-ware solutions and requires additional software (the web service) that might be an additional point of failure. Java's JDBC provides a standard way to access databases in Java, but this interface is missing in Java ME. In this paper we present our implementation of an MIDP-based Java ME driver [HSWM09] for MySQL similar to JDBC that allows direct communication of MIDP applications to MySQL servers without a middleware. We illustrate the usage of the driver by our prototype MySQL client for MIDP enabled mobile phones.

## 2 Overall Architecture

Before we started the development we set ourselves four design goals: (1) keep the driver API as near to the JDBC specification as possible, (2) keep the `.jar`-file size below 32kB

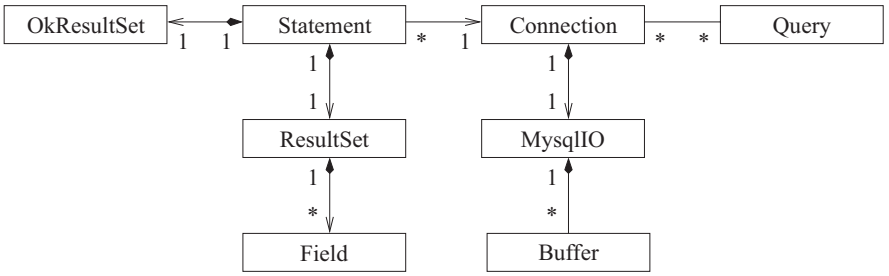



Figure 1: Class Overview

– half the popular 64kB limit for cell phones – to leave enough space for the application, (3) keep the implementation code as simple and performant as possible. These goals were mostly achieved. Our current development version provides database access sufficient for most applications in just 27kB. (In comparison, the MySQL JConnector JDBC driver has more than 500kB.) On the other hand we had to cut short on some aspects like parametrized queries and meta data usage. Figure 1 shows the basic class diagram of the driver.

The *Buffer* class is responsible for encoding and decoding packet fields as well as for the conversion between MySQL and Java data types. The *MysqlIO* class handles the communication with the database server. The *MysqlIO* class uses two *Buffer* instances, one for sending and one for receiving, to which it has exclusive access, ensuring strict task separation between the classes. The *Connection* class is very similar to the *Connection* interface of JDBC. It owns an instance of *MysqlIO* and uses it to provide connection specific methods like opening and closing a connection and changing the database. It also works as factory for *Statement* and *Query* objects. The *Statement* class is very similar to the JDBC *Statement* interface. It provides methods to execute database queries and fetch the the result. For this it implements the packet sequence logic necessary, but relies on the instance of the *MysqlIO* class hold by the *Connection* class factory for doing all packet processing. The *Query* class provides basic functionality for parametrized queries. The *OKResultSet* is an simple query information storage and is solely used by the *Statement* class. It is not directly available to the application developer but must be accessed through methods provided by the *Statement* class. The *ResultSet* class performs the same job as the JDBC *ResultSet* interface, providing exactly the same row-pointer based access methods. It uses an array of *Field* class instances to store and process all column specific data. In fact, the *ResultSet* does only act as a facade to the *Field* class, managing the row dimension of the database result set. The *Field* class provides column wise storage for database result sets and meta data. A large number of simple methods provide access to specific column- and meta data. It is only used internally by the *ResultSet*. Three additional helper classes provide a number of static methods for common tasks not really part of the driver (like string operations). The *Constants* class contains all the necessary constants. Finally, there is one *SQLException* class used throughout the driver.

## 3 Using MyMIDP

Every MIDP 2.0 compatible device should be able to use the driver when the following, additional requirements are fulfilled: It must support socket connections (optional in MIDP 2.1). It must support JSR 177[1] (needed for MySQL authentication via SHA-1). It should have at least one megabyte of free heap memory (depending on the implementing application).

Since the driver API is a very similar to JDBC, a developer familiar with JDBC will not have any problems using our driver. And even developers new to database APIs will find our driver easy to use as it always follows four steps: (1) Create a database connection, (2) Create and execute a database statement, (3) Process the result set, (4) Close the connection. Steps two and three can be repeated in case more than one query must be executed. For illustration purposes the following listing shows a short usage example:

```
import de.iu.db.mysql.mini.Connection;
import de.iu.db.mysql.mini.ResultSet;
import de.iu.db.mysql.mini.Statement;
import de.iu.db.mysql.mini.exceptions.SQLException;

public class Demo {

    public static void main(String[] args) {

        try {
            // connecting to database 'catsanddogs' on server test.somenetwork.net:3006, user 'test', pwd 'run'
            Connection con = new Connection("test.somenetwork.net", 3006, "test", "run", "catsanddogs");
            // retrieve some data
            Statement st = con
                .createStatement
                ("SELECT name, age, owner FROM dogs");
            ResultSet rs = st.executeQuery();
            // loop through the result set
            for (; rs.current() < rs.getResultCount(); rs.next()) {
                // access the data using row and column pointer
                System.out.println("The Dog " + rs.getAsString(0) + " ("+ rs.getAsInt(1) + ") is owned by "
                    + rs.getAsString(2));
            }
            // adding some data
            long count = st.executeUpdate("INSERT INTO dogs (name, age, owner)" + "VALUES ('Angel', 12, 'Charlie')");
            System.out.println("Added " + count + " datasets with message: "+ st.getMessage());
            // end the session
            con.close();
        } catch (SQLException e) {
            // do some error handling
            e.printStackTrace();
        }
    }
}
```

**Main differences to JDBC** There are a few important usage differences to JDBC we would like to point out. First, the driver does not use the JDBC style connection URL but method parameters for simplicity and performance reasons. Second, the *Statement* object can be reused thus improving garbage collection. Third, it is not possible to execute multiple statements at the same time as they use the same *MysqlIO* class instance and thus share buffers.

[1] JSR 177: Security and Trust Services API for J2ME™: `http://jcp.org/en/jsr/detail?id=177`

## 4 A Prototype Implementation: MyMIDP-Client

As a prove of concept we implemented a prototype client (see Figure 2) that utilizes the MyMIDP driver. After connecting to a MySQL server the client provides query templates. The user can choose between select, insert, update, and delete. As these templates only support typing in queries the client only preinitializes the query string with the staring keyword of the query. The user has to complete the query string manually but might also remove the key word provided by the templates. After submitting the query to the server, the result is displayed. Due to the limited size of the display we decided to add a unique ID to each tuple and to display the first attribute only. Then, by selecting this ID, the user can display the tuple completely.

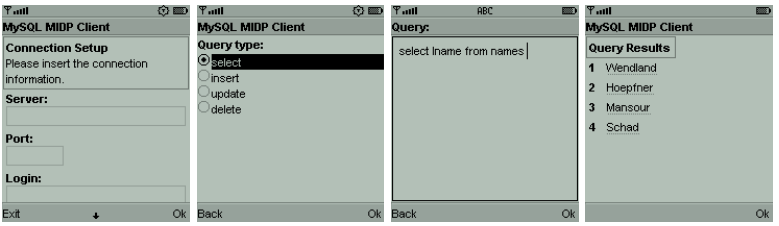



Figure 2: The MyMIDP-client in action

The MyMIDP sources and the MyMIDP-client prototype are GPL licensed and available at `http://it.i-u.de/dbis/MyMIDP`.

## References


[CIIH07] Alexandru Caracaş, Iulia Ion, Mihaela Ion, and Hagen Höpfner. Towards Java-based Data Caching for Mobile Information System Clients. In Birgitta König-Ries, Franz Lehner, Rainer Malaka, and Can Türker, editors, *MMS 2007: Mobilität und mobile Informationssysteme; Proceedings of the 2nd conference of GI-Fachgruppe MMS*, volume P-104 of *LNI*, pages 97–101, Bonn, Germany, 2007. GI, Köllen Druck+Verlag GmbH.

[HSWM09] Hagen Höpfner, Jörg Schad, Sebastian Wendland, and Essam Mansour. MyMIDP: An JDBC driver for accessing MySQL from mobile devices. In *Proceedings of the 1st International Conference on Advances in Databases (DB 2009), March 1-6, 2009 - Gosier, Guadeloupe/France*. IEEE, 2009. accepted for publication.

[IBM04a] IBM Corporation. *IBM DB2 Everyplace Application and Development Guide Version 8.2*, August 2004.

[IBM04b] IBM Corporation. *IBM DB2 Everyplace Sync Server Administration Guide Version 8.2*, August 2004.

[ICH07] Iulia Ion, Alexandru Caracaş, and Hagen Höpfner. MTrainSchedule: Combining Web Services and Data Caching on Mobile Devices. *Datenbank-Spektrum*, 21:51–53, May 2007.

[KRTH07] Birgitta König-Ries, Can Türker, and Hagen Höpfner. Informationsnutzung und -verarbeitung mit mobilen Geräten – Verfügbarkeit und Konsistenz. *Datenbank-Spektrum*, 7(23):45–53, 2007. in German.

[Mic08] Microsoft Corporation. `http://msdn.microsoft.com/library/`, 2008.

[Ora04a] Oracle Corporation. *Oracle Database Lite, Administration and Deployment Guide 10g (10.0.0)*, June 2004.

[Ora04b] Oracle Corporation. *Oracle Database Lite, Developer's Guide 10g (10.0.0)*, June 2004.

[Ora04c] Oracle Corporation. *Oracle Database Lite, SQL Reference 10g (10.0.0)*, June 2004.

[Syb08] Sybase Inc. `http://www.sybase.com/ianywhere/products`, 2008.

# Streaming Web Services and Standing Processes

Steffen Preißler, Hannes Voigt, Dirk Habich, Wolfgang Lehner
Technische Universität Dresden
Lehrstuhl für Datenbanken
dbgroup@mail.inf.tu-dresden.de

**Abstract:**
Today, service orientation is a well established concept in modern IT infrastructures. Web services and WS-BPEL as the two key technologies handle large structured data sets very inefficiently because they process the whole data set at once. In this demo, we present a framework to build standing business processes. Standing business processes rely on item-wise data set processing, exploit pipeline parallelism and show a significantly higher throughput than the traditional WS-BPEL approach.

## 1 Introduction

Today an increasing number of IT infrastructures are built as a *service-oriented architecture* (SOA). In SOA, independent systems provide their functionality as interoperable services. Systems group theses services in business processes and package them as further interoperable services. Web services and WS-BPEL are two key technologies to realize a SOA. The Web service specification [W3C02] offers standardized structures for self-description and message exchange and is, therefore, the well established standard for interoperable services. WS-BPEL [OAS07] provides workflow constructs to build fully-fledged processes with service calls as core activities.

Within the THESEUS research project [BMW07], the TEXO use case aims to built a SOA-based platform where services are tradable and business value networks can be established. As partner in TEXO, we investigate the efficient processing of large structured data sets in SOA environments. Our approach is a new type of process, which is called *standing business process*. It builds on top the workflow constructs of WS-BPEL, but exploits pipeline parallelism for large data set processing. Thereby large data sets can be considered as a stream of equally structured messages or a given set of equally structured data items. [LCF08] already discussed that the throughput of large data set processing can be increased significantly by exploiting pipeline parallelism. Traditional approaches only map each single item (or message) to a single process instance with a still step-wise execution model and single service calls. These approaches limit the processing semantic to single-item operations. However, common business process operations such as aggregations involve more than one data item.

As an example, consider the stock-ticker process in Figure 1, which handles incoming RSS-Feed messages. Whenever a message arrives, only interesting stocks are selected for

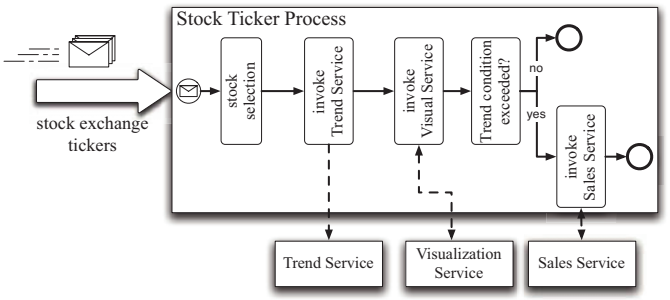



Figure 1: Sample stock ticker process

further processing. The selected stocks in a message are evaluated over a time window using a service (`Trend Service`) that monitors different stock trends and keeps a history of their values. The service returns the stock trend over the requested time window and the process sends this information to a visualization service, that displays the trend on a customer's dashboard. Afterwards the process compares the trend to a predefined value. If the trend exceeds a certain threshold, the `Sales Service` service will be triggered, otherwise nothing is done.

This type of application scenario cannot be executed efficiently with classic means. Traditionally, each ticker message triggers the explicit creation and execution of one process instance. Thereby each message is processed in its own context and a common context for ,e.g., in-house stock trend computation can not be exploited. Furthermore the start of a message's processing is delayed, until the process instance of the previous message has been executed successfully to ensure temporal integrity. This leads to a significantly lower throughput of incoming messages.

In this demo we introduce the novel notion of standing business processes that realize pipelined and context-preserving data set processing in the SOA world. To show the applicability of our concept, we present a framework to model and execute such standing processes. Our approach establishes real pipeline parallelism, increases the processing throughput and does not restrict the processing semantic.

## 2 Streaming in service-oriented environments

Traditional business processes in WS-BPEL follow an *instance-based execution model*. Every incoming message creates a dedicated process instance, which is executed isolated from all other instances. This type of process execution is not efficient for processing large amounts of incoming, equally structured messages that semantically belong to one context. However, in WSBPEL it is possible to express one context for a set of incoming messages with the help of a *while* loop and *correlation sets*. The *while* loop has to enclose all corresponding control flow activities for one message. Additionally, *correlation sets*

route messages to specific process instances. The disadvantages of this approach are (i) the explicit modeling of a control flow loop with one iteration for every message, (ii) the still step-wise execution within the loop where only one activity is running and all others are idle and (iii) the need for a static value within the messages' body to use *correlation sets* and to map, thereby, messages with specific values to a specific process instance.

In order to use pipelined parallelism in combination with business process types like our stock ticker process, the process engine requires the *pipes and filters execution model*. In *pipes and filters* every activity of a process is executed as a single thread and each edge between two activities contains a queue, which buffers data that belong to specific messages. Additionally the semantic and the functionality of each control flow operator are adapted to work with input and output queues and to realize one common process context for all messages.

Service invocation and execution within a standing process needs adaptation, too. The *pipes and filters execution model* implies that services are called item-wise. Consequently, with the traditional service invocation pattern, data items lose their context on service side, since one service instance is created for every item. To preserve the context of the items on service side, a standing process pushes the message queue embracing *one* service invocation down to the service instance. In addition to that, the service execution is enhanced to process these items in a stream-based fashion. By this means the standing process adds streaming semantic to the service call. This enables the service to return already processed data items while still receiving request items. This stream-based service execution eliminates the overhead of single message creation compared to traditional item-wise service invocations and preserves the context of the items at the same time. [PVHL09] discusses this approach in more detail.

The visual modeling of standing processes with our framework heavily corresponds to the modeling of standard processes (see Figure 2). Similar to standard processes, the user chooses from a set of operators provided by the framework and orchestrates them to a workflow definition. What differs in the visual representation are the queue symbols between two connected activities. These symbols represent the already mentioned message queues, so that the user can configure them at design time (e.g., maximum queue size). When executing a standing process, our framework visualizes a running process instance by displaying the modeled graph and augmenting its graphical components with information about the current workload for every queue, the average execution time for every operator as well as path counters when utilizing switch operators.

## 3 Demo Details

Fundamentally, the demonstration will consist of two parts. In the first part, we demonstrate the execution of standing processes with our developed standing process engine. For this, we prepared a set of predefined processes. As an example, Figure 2 shows a screenshot of the stock-ticker process discussed in the introduction.

These demo part shows the applicability of our standing process concept and the imple-

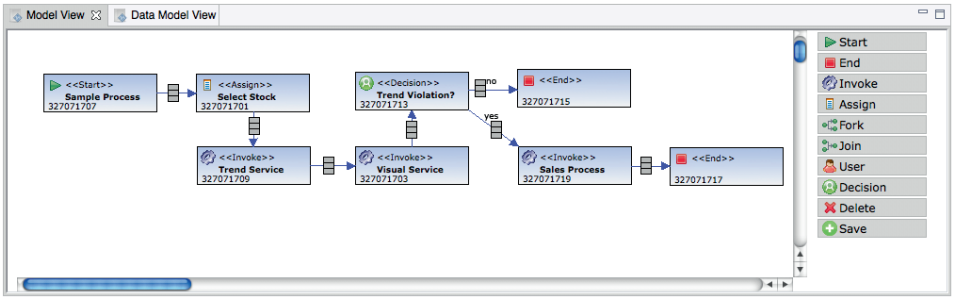



Figure 2: Screenshot of stock-ticker process

mented framework within several scenarios. In addition to the prepared processes, visitors of our demo desk will also have the possibility to experience the orchestration of new standing process definitions. This gives the visitors an understanding of the whole modeling approach and the benefits of standing processes.

In the second part, we present the implementation and usage of stream-based services. For this, we prepared stream-based services, traditional services, and an execution front-end to experimentally show the benefit of our approach. Furthermore, we describe our extension on the Web Service interface description, which allows us to identify and use stream-based services with our standing process framework. We welcome visitors of our demonstration desk to create new stream-based Web services and show the easy usage of our service framework. In this case, visitors of the demo will get an in-depth understanding of our developed concept.

## 4 Acknowledgements

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference "'01MQ07012"'. The authors take the responsibility for the contents.

## References

[BMW07] Bundesministerium für Wirtschaft und Technologie BMWi. THESEUS Programme, 2007. `http://theseus-programm.de/`.

[LCF08] Melissa Lemos, Marco A. Casanova, and Antonio L. Furtado. Process pipeline scheduling. *J. Syst. Softw.*, 81(3):307–327, 2008.

[OAS07] OASIS. Web Services Business Process Execution Language 2.0 (WS-BPEL), 2007. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

[PVHL09] Steffen Preissler, Hannes Voigt, Dirk Habich, and Wolfgang Lehner. Stream-based Web Service Invocation. In *BTW*, 2009.

[W3C02] World Wide Web Consortium W3C. Web Service specifications, 2002. `http://www.w3.org/2002/ws/`.

# End-to-End Performance Monitoring of Databases in Distributed Environments

Stefanie Scherzinger, Holger Karn, Torsten Steinbach
IBM Deutschland Research and Development, Böblingen
{sscherz, holger.karn, torsten}@de.ibm.com

**Abstract:** This demonstration features the IBM *DB2 Performance Expert for Linux, Unix and Windows*, a high-end database monitoring tool that is capable of end-to-end monitoring in distributed environments. Performance Expert closely tracks the execution of database workload, from the point when the workload is issued in a database application, to its actual execution by the database server. This enables us to break down the response time as experienced by the user into time spent waiting for connections, within database drivers, the network, and finally, within the database server. From a data management point-of-view, the main challenge here is the handling of large amounts of performance data. We show how this challenge is met by continuous aggregation, and the definition of so-called *workload cluster groups*, which organize and narrow down the data of interest.

## 1 Motivation

Whether configuring, optimizing, or administrating databases and database applications, it is vital to maintain statistics on the database health and performance. The data collected during database monitoring is the basis for long-term planning of resources, as well as for short-term problem determination.

For instance, consider users who complain about long response times of a database application. In a typical customer environment, Java[†] applications access a database remotely. Locating the causes of performance problems in such a distributed setting can be a serious challenge. The response time, as experienced by the users, can actually be broken down into the time spent inside the application server, the database drivers, the network, the operating system, and finally, the database server. Within the individual layers, it can be helpful to further itemize the costs for particular tasks, such as locking or I/O.

Performance bottlenecks may lurk in any of these layers. While the database administrator (DBA) is often among the first in line for pinpointing a performance issue, the DBA typically only knows the database side of the problem.

There is a variety of dedicated tools for monitoring each of these layers, such as for monitoring the performance of the operating system, the network traffic, or the database itself. Yet when the layers are observed in isolation, a considerable amount of time is spent on consolidating the views of different monitoring tools.

For instance, when Java applications do not register with the database, DBAs may find

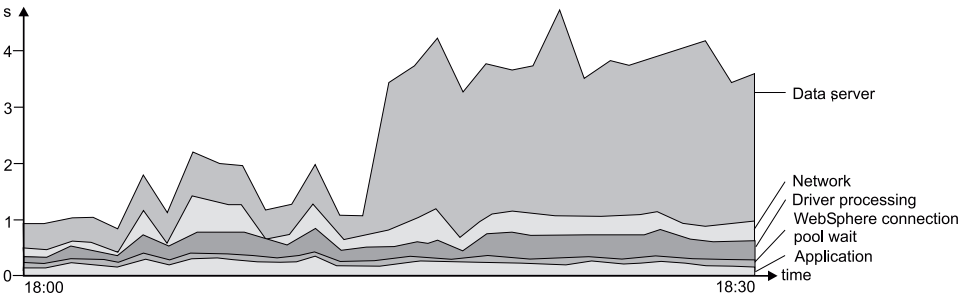



Figure 1: Breakdown of response times in a rainbow chart.

it hard to map these seemingly nameless applications to the applications that users are complaining about. However, if network problems are responsible for delays, monitoring the database in isolation will not reveal the problem. Another example concerns connection pooling in applications. If a connection pool is exhausted, users experience sluggish performance, yet the DBA cannot find anything wrong with the database.

What is missing here is a holistic view on the execution of database workload from the one end, that of the application, to the other, namely the database server. That is, we need an *end-to-end* approach to database performance monitoring.

## 2 End-to-end Performance Monitoring

In this demonstration, we present end-to-end performance monitoring for database applications. This approach is featured in the *Extended Insight Feature* of IBM's *DB2 Performance Expert for Linux, Unix and Windows* [IBM08, KBS08]. DB2 Performance Expert is a high-end monitoring tool for DB2* installations. The initial release of the Extended Insight Feature focuses on WebSphere* and other Java† applications accessing DB2 data on Linux†, Unix†, or Windows† platforms.

DB2 Performance Expert closely tracks the propagation of database workload through the major execution layers, so that DBAs can readily see where the processing time is spent. A graphical user interface features *rainbow charts*, as shown in Figure 1. The x-axis depicts the time passed during monitoring, whereas the y-axis shows runtimes in seconds.

The rainbow chart provides a comprehensive view by breaking down the database transaction response time into the time spent inside the application, wait times inside connection pools, the processing inside database drivers, the network, and finally, the time spent within the database server itself. This allows the administrator to observe the end-to-end runtime behavior of the incoming database workload over time.

*Trademarks of IBM in USA and/or other countries.

†Other company, product, or service names may be trademarks or service marks of others.

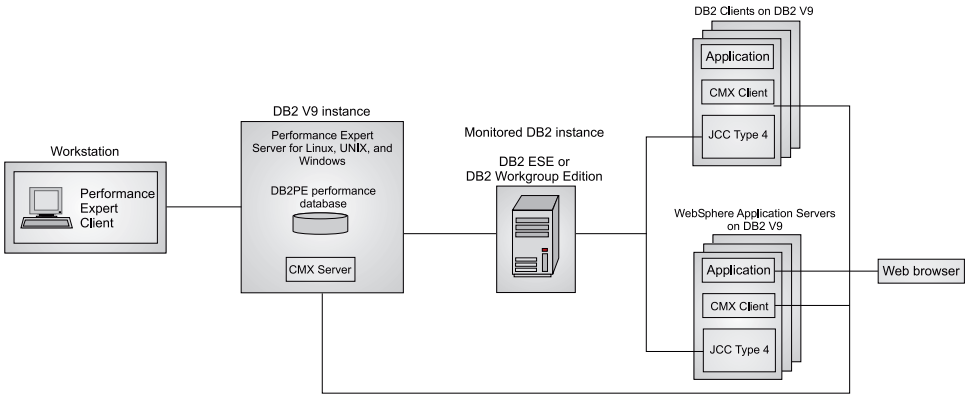



Figure 2: Distributed system architecture.

**Data management challenges.** The handling of large amounts of performance data with little overhead to the monitored system poses a challenge. End-to-end performance statistics are generated at the level of single transactions as well as the execution of single SQL statements. In DB2 Performance Expert, this data management challenge is effectively addressed in two ways.

- Users define *workload cluster groups* based on DB2 connection attributes. Performance data is then filtered and aggregated according to the DBA's interests. For instance, the data can be clustered by the names of users who issued the workload, and filtered for specific applications.
- Historical monitoring data is repeatedly aggregated over specific *data retention periods*. DBAs can access performance data months and even years later. Historical data is aggregated over longer timeframes, and thus more compactly, than more recent performance data.

**System architecture.** Figure 2 shows the distributed system architecture of DB2 Performance Expert. A DB2 instance is shown in the center. This instance is remotely monitored by an installation of *Performance Expert Server*. All monitoring data is stored persistently in the DB2PE performance database, maintained by *Performance Expert Server*.

DBAs interact with the *Performance Expert Client*, a graphical user interface that remotely accesses the performance database and visualizes its contents. By defining workload cluster goups, DBAs obtain customized views over the end-to-end monitoring data.

End-to-end monitoring is shown for two Java applications. These generate database workload on the monitored DB2 instance. The Java applications need not be recompiled or even relinked for end-to-end monitoring. Rather, they are merely configured to use JCC Type 4 as their database driver, and have a so-called *CMX Client* (short for *Client Management Extension*) installed on site.

The *CMX Clients* are responsible for tracking and collecting end-to-end monitoring data. They perform a pre-aggregation step on the collected data, and forward it to the *CMX Server* at regular intervals. The *CMX Server* is part of *Performance Expert*, and stores the end-to-end statistics in the performance database.

## 3 Demonstration Summary

In our demonstration, we set up an environment according to Figure 2, where we (1) generate database workload with a Java application, (2) monitor the execution of the workload on a DB2 instance, and (3) integrate these observations with end-to-end monitoring data. Then we show how users may systematically search out performance bottlenecks:

- We demonstrate how DBAs may define workload cluster groups. These are the means to view end-to-end monitoring data for specific applications, users, or clients.
- We show the breakdown of the response time in rainbow charts, as displayed in Figure 1. Rainbow charts are a comprehensive means for DBAs to quickly identify processing layers that may host performance bottlenecks.
- In case the database is harboring performance problems, DBAs can exploit the classic monitoring facilities of DB2 Performance Expert. These are custom-tailored towards performance tracking of DB2 installations.
- We query for the top-$k$ SQL statements, sorted by crucial criteria such as the end-to-end response time.
- We show how the characteristics of different data servers may be compared with regard to the average network time or the number of opened connections.
- We generate database workload for live-monitoring, but also show the stepwise aggregated historical performance data. History data is crucial for recognizing long-term changes in the system.

In summary, this demo will feature the monitoring functions of *DB2 Performance Expert for Linux, Unix and Windows*, with a focus on its novel end-to-end monitoring capabilites.

## References

[IBM08] IBM. "DB2 Performance Expert for Linux, Unix and Windows", 2008. http://www-01.ibm.com/software/data/db2imstools/db2tools/db2pe/db2pe-mp.html.

[KBS08] Holger Karn, Ute Baumbach, and Torsten Steinbach. "Sneak peek: End-to-End Database Monitoring using IBM DB2 Performance Expert". *IBM Database Magazine*, (4), 2008.

**Acknowledgements.** This work is the combined effort of our colleagues from the *DB2 Performance Expert for Linux, Unix and Windows* teams in Böblingen and Krakow. Many thanks.

# Now it's Obvious to The Eye—Visually Explaining XQuery Evaluation in a Native XML Database Management System

Andreas M. Weiner, Christian Mathis, Theo Härder, and Caesar Ralf Franz Hoppen
Databases and Information Systems Group
Department of Computer Science
University of Kaiserslautern
67653 Kaiserslautern, Germany
{weiner, mathis, haerder, hoppen}@cs.uni-kl.de

**Abstract:** As the evaluation of XQuery expressions in native XML database management systems is a complex task and offers several degrees of freedom, we propose a visual explanation tool—providing an easily understandable graphical representation of XQuery—for tracking the XQuery evaluation process from head to toe.

## 1 Introduction

In recent years, XML gained a lot of attention as a means for exchanging structured and semi-structured data. Native XML database management systems (XDBMSs) are a promising approach for storing and managing such documents in a transactional way. Having a closer look at XQuery—the dominant query language for XML—reveals that it is an extremely powerful, but at the same time, a very complex query language. In this work, we present the *XPlain* tool for visually explaining the evaluation of XQuery expressions in *XTC (XML Transaction Coordinator)* [HH07]—our prototype of a native XDBMS. Using our tool, we can track the complete XQuery evaluation process beginning at the translation of the query into an internal representation, ranging over the application of several rules for algebraic optimization, and ending in a query execution plan which is executed using the query evaluation engine of XTC.

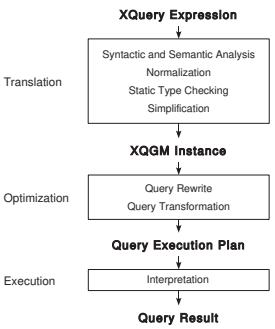



Figure 1: The XTC query evaluation process

We are not aware of any tool that allows to follow all stages of the XQuery evaluation process from the beginning to the end in a catchy way that is even easy to understand for XQuery novices and non-database experts. Our visual explanation tool supports different types of users in improving their work: (1) Developers of XML query optimizers can immediately see the impact of rewrite and optimization rules on subsequent query graphs, (2) Lecturers benefit from our self-explanatory graphical query representation and can use it to teach undergraduate XQuery classes, and (3) Database administrators can focus solely on the query execution plan and speed-up query evaluation by creating new indexes or by activating or deactivating different rewrite or optimization rules.

## 2 Related Work

Compared to the work of Rittinger et al. [RTG07], which empowers a relational query optimizer to evaluate XQuery expressions and visualizes only QEPs, we are able to illustrate every step in the query evaluation process. Furthermore, by sticking to a rule-based approach, we can re-configure our query optimizer even at runtime.

## 3 Architectural Issues

Figure 1 shows the three stages of the XTC query evaluation process: *translation*, *optimization*, and *execution*. During the translation stage, an XQuery statement is checked for syntactical and semantical correctness. These checks are followed by a normalization phase, where semantically equivalent queries are mapped to a common normal form expression according to the formal semantics of XQuery.

Before the normal form expression is mapped to the so-called *XML Query Graph Model (XQGM)* [WMH08][1], we perform static type checking and apply several simplification rules to remove redundant parts of the query. For example, Figure 2 shows a graphical representation of the XPath path expression doc("auction.xml")//site//mail which was exported using XPlain. Because an XQGM instance is equivalent to a logical algebra expression, it allows to perform algebraic optimization. Based on an XQGM graph provided as input for the optimization stage, several rewrite rules, e. g., query unnesting [Mat07] and join fusion [WMH08] are applied, resulting in a semantically equivalent structure which can be evaluated more efficiently than the initial one. In the query transformation step, a rewritten XQGM instance is mapped to a *Query Execution Plan (QEP)* (physical algebra expression). Finally, the QEP is executed by direct interpretation using the well-known open-next-close protocol [Gra93].

We developed our query optimizer following a strictly extensible rule-based approach, i. e., every modification of an XQGM instance (e. g., by algebraic rewrite) is specified by a rule consisting of a pattern and an action part. Patterns are identified by our generic pattern matching engine and the actions are applied by a transformation engine. Consequently, we can (1) easily extend our system by adding new rules and (2) switch on and off specific simplification, rewrite, and logical-to-physical mapping

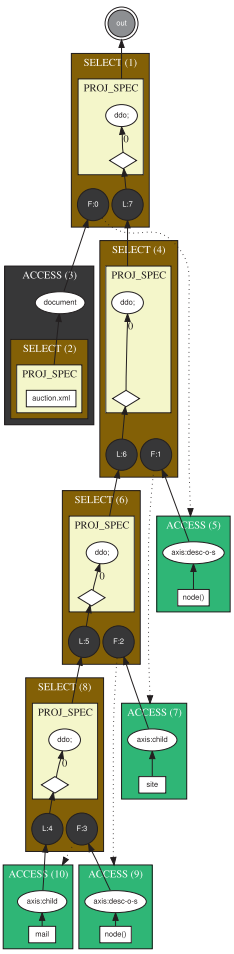



Figure 2: A sample XQGM instance

[1] Note, the XQGM is an extended version of Starburst's well-known *Query Graph Model (QGM)* [PHH92] which we made to measure for the XQuery language.

Figure 3: The XPlain GUI

rules according to our needs. Thus, we can play the role of a query optimizer and immediately see the impact of different optimization strategies even at runtime.

Whenever an action is performed by the transformation engine, a textual representation of the resulting XQGM graph—a so-called *dot graph*—is generated reflecting all changes performed. By doing so, we get a complete history of all transformations applied to the initial XQGM graph as well as a graphical representation of the final QEP.

The XPlain tool—implemented using Java 1.6—provides a sophisticated Swing-based GUI and connects to the XTC server as a client using Java RMI. It receives the query result, statistics on each phase of the query evaluation process, and all dot plans generated. Using the *GraphViz* visualization software [EGKW03]—a powerful framework for layouting huge graphs—all dot plans are converted into Scalable Vector Graphic (SVG) instances which are rendered in the XPlain GUI using the *Apache Batik SVG Toolkit*[2].

Figure 3 shows the XPlain GUI. At the left-hand side, you can see a list of all documents currently stored on the server (top-most box), the path synopsis—a kind of dynamic schema allowing to create XPath path expressions just by clicking on the node names (box in the middle), and meta data on currently available indexes for each document (bottom line). The main panel displays a rendered XQGM graph corresponding to the query entered in the text box atop of it. At the top-most right side, you can select a query from predefined query sets[3]. Furthermore, the right side shows the history of all dot plans generated during query evaluation, which can be rendered by just selecting the corresponding

[2] `http://xmlgraphics.apache.org/batik`

[3] For example, Figure 3 shows the query graph for query Q7 of the well-known XMark benchmark queries [SWK+02].

item. Moreover, by using the up-and-down buttons, you can linearly track each modification of the XQGM graph from beginning to the end. Finally, the menu bar provides three major menus (simplification, restructuring, and transformation) allowing to select all rules to be applied during query evaluation. Figure 3 shows the complete transformation menu. If there is more than one pattern finding a match in the graph, we can assign a priority to each rule, which may be used to give preferences over alternative ones. Because there are several dependencies between rules within and across the simplification, restructuring, and transformation rule sets, we provide predefined rule sets to choose from and support creating custom rule sets by experienced users.

## 4 Demonstration Setup

During the demonstration session, we come up with a predefined set of XMark benchmark queries [SWK+02] and provide different-sized XMark documents to run these queries on. Furthermore, we furnish different rule sets allowing to visually compare the impact of varying query evaluation strategies: Using the *node-at-a-time* configuration, we can explore how a query is evaluated according to XQuery's formal semantics. On the other hand, using different *set-at-a-time* configurations, we illustrate how exclusive or combined use of structural joins, holistic twig joins, and different index access operators can boost query execution tremendously.

## References


[EGKW03] J. Ellson, E.R. Gansner, E. Koutsofios, and S.C. Northand G. Woodhull. Graphviz and Dynagraph—Static and Dynamic Graph Drawing Tools. In M. Junger and P. Mutzel, editors, *Graph Drawing Software*, pages 127–148. Springer, 2003.

[Gra93] Goetz Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[HH07] Michael Haustein and Theo Härder. An Efficient Infrastructure for Native Transactional XML Processing. *Data & Knowledge Engineering*, 61(3):500–523, 2007.

[Mat07] Christian Mathis. Extending a Tuple-Based XPath Algebra to Enhance Evaluation Flexibility. *Informatik – Forschung und Entwicklung*, 21(3–4):147–164, 2007.

[PHH92] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *Proc. SIGMOD Conference*, pages 39–48, 1992.

[RTG07] Jan Rittinger, Jens Teubner, and Torsten Grust. Pathfinder: A Relational Query Optimizer Explores XQuery Terrain. In *Proc. BTW Conference*, pages 617–620, 2007.

[SWK+02] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A Benchmark for XML Data Management. In *Proc. VLDB Conference*, pages 974–985, 2002.

[WMH08] Andreas M. Weiner, Christian Mathis, and Theo Härder. Rules for Query Rewrite in Native XML Databases. In *Proc. EDBT DataX Workshop*, pages 21–26, 2008.

# ATE: Workload-oriented DB2 tuning in action

David Wiese[1], Gennadi Rabinovitch[1], Michael Reichert[2], Stephan Arenswald[2]

[1] Friedrich-Schiller-University Jena, Germany
*{david.wiese, gennadi.rabinovitch}@cs.uni-jena.de*

[2] IBM Deutschland Research & Development GmbH, Boeblingen, Germany
*{rei, arens}@de.ibm.com*

## 1 Introduction

Databases are growing rapidly in scale and complexity. High performance, availability and further service level agreements need to be satisfied under any circumstances to please customers. In order to tune the DBMSs within their complex environments highly skilled database administrators are required. Unfortunately, they are becoming rarer and more and more expensive. Improving performance analysis and moving towards automation of problem resolution requires a more intuitive and flexible source of decision making.

This demonstration points out the importance of best-practices knowledge for autonomic database tuning and addresses the idea of formalizing and storing this knowledge for the autonomic management process in order to minimize user intervention and enable the system to (re)act autonomously. For this purpose, we propose an architecture for autonomic database tuning of IBM* DB2* UDB for Linux*, UNIX* and Windows* and demonstrate our system's tuning performance under changing workload.

## 2 ATE Architecture

The architecture of our Autonomic Tuning Expert (ATE) [WRRA08] implements a component-based MAPE loop [IBM05] and is based on widely accepted and influential

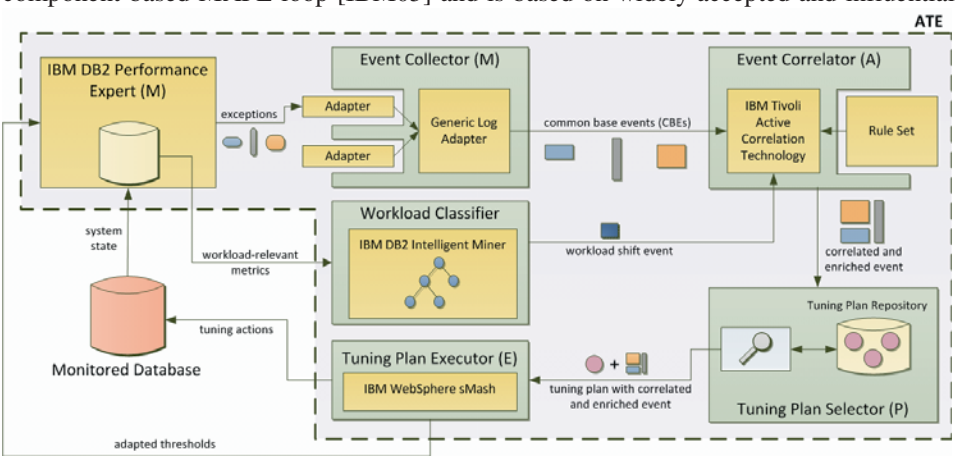
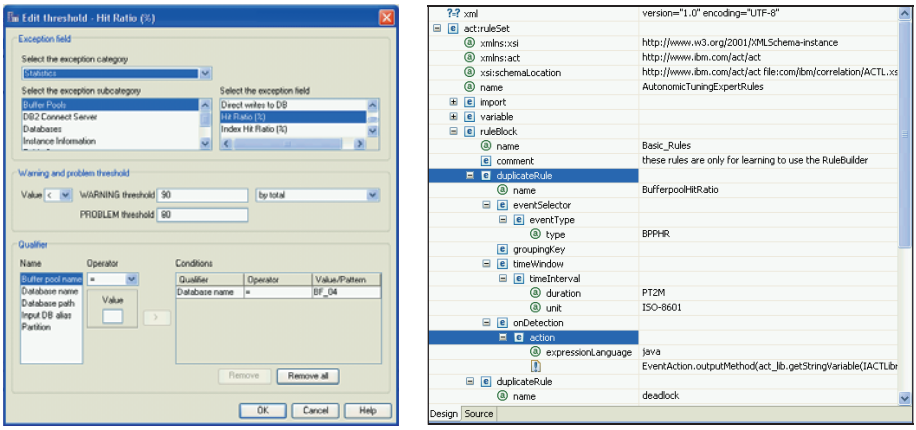



Figure 1: Architectural Blueprint of ATE




a) PE threshold definition b) ACT complex event definition

Figure 2: Problem formalization

technologies and industry-proven products like Eclipse, Generic Log Adapter (GLA), IBM DB2 Performance Expert (PE), Tivoli* Active Correlation Technology (ACT), and Common Base Events (CBE), as illustrated in Figure 1.

PE's periodic exception processing feature [CBM06] regularly provides information about all pre-defined exception situations in XML format (atomic events). GLA obtains these atomic events, transforms them into the CBE format and finally sends them to the Event Correlator. The Event Correlator deploys ACT [BG05] for filtering and correlating events and enables the framework to determine the context of the problems indicated by atomic events. Every time the ACT engine detects a complex correlated event, a rule response is created. This response consists of the original CBE containing information – like the name of the database, names of the affected objects, etc. – and additional meta information – like the recognized workload on the monitored database system – that is appended to the CBE. The Tuning Plan Selector is now responsible for browsing through the Tuning Plan Repository and retrieving a tuning plan that best resolves the problem identified by the Event Correlator. After a proper tuning plan is determined, the Tuning Plan Executor is invoked. The problem context representing CBE is passed along to the Tuning Plan Executor ensuring that the knowledge about the problem itself is available for tuning plan execution as well. The Tuning Plan Executor is implemented by the help of a light-weight workflow engine integrated in IBM WebSphere* sMash (WSM) application server [IBM08]. It enables the execution of user-defined database tuning workflows.

As tuning of a database system heavily depends on the current workload type, a general workload classification framework has been implemented and integrated into ATE. The classification framework, based on IBM DB2 Intelligent Miner* [IBM06], is used to capture workload characteristics and to establish a system and tuning independent database workload classification model. At run-time this model can be used to determine the current workload on the monitored database.

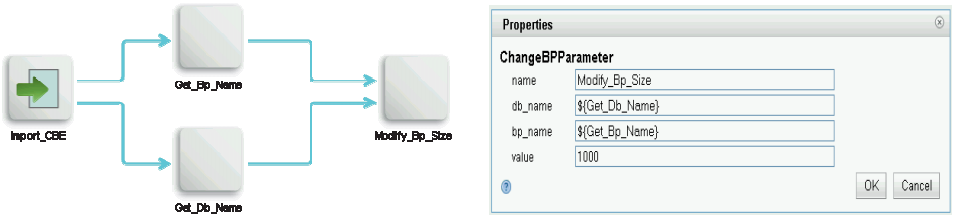


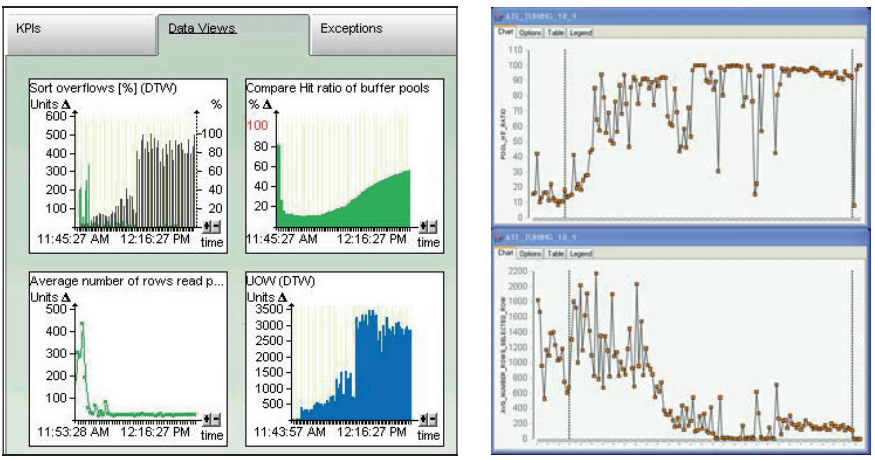

Figure 3: Websphere sMash workflow editor

## 3 Demonstration Setup

At the demonstration stand we show how ATE automatically reacts to user-defined problems with user-defined tuning plans by re-configuring the environment under control considering the current workload.

Initially, problem situations need to be encapsulated into atomic and complex events by defining PE thresholds and ACT correlation patterns (Figure 2). Furthermore, tuning plans for resolving these tuning problems need to be implemented and integrated into the system as well. WSM's visual workflow editor allows users to create new or adapt existing tuning workflows by combining pre-defined database tuning steps using control structures (Figure 3). In addition, upper and lower boundaries for numeric configuration parameters depending on the amount of physical memory available can be specified. They are captured in a simple resource restriction model that helps to avoid resource overallocations.

ATE's Workload Classifier enables workload-oriented problem detection and resolution. At the moment, it can distinguish TPC-C-like [TPCC07] and TPC-H-like [TPCH08] workload types. However, further workload classes could be integrated with little




a) PE system overview b) ATE KPI Visualizer tool

Figure 4: Tools for the evaluation of ATE

overhead. For evaluating the prototype's effectiveness, an arbitrary sequence of both workload types can be used.

Users can verify if iterative tuning has the desired effect on relevant Key Performance Indicators (KPI) either at run-time by using PE performance data real-time visualization or afterwards by means of our ATE KPI Visualizer tool that additionally can display ATE-internal metadata such as start or stop of ATE, event occurrences and corresponding tuning plan executions (Figure 4). Furthermore, pre-defined scripts can be used to obtain current parameter values and object allocations.

## References


[BG05] A. Biazetti and K. Gajda. Achieving complex event processing with Active Correlation Technology. November 2005.

[CBM+06] W.J. Chen, U. Baumbach, M. Miskimen, et al. DB2 Performance Expert for Multiplatforms V2.2, March 2006.

[IBM05] IBM Corp. An Architectural Blueprint for Autonomic Computing. Third Edition. White Paper, IBM Research Lab. June 2005.

[IBM06] IBM Corp. DB2 Data Warehouse Edition Version 9.1.1 - Intelligent Miner Modeling: Administration and Programming Guide, April 2006.

[IBM08] IBM Corp. WebSphere sMash Documentation. http://publib.boulder.ibm.com/infocenter/wsmashin/v1r0/index.jsp. Last accessed: November 17, 2008.

[TPCC07] TPC Benchmark* C. Standard Specification. Revision 5.9. Transaction Processing Performance Council, June 2007.

[TPCH08] TPC Benchmark* H. Standard Specification Revision 2.7.0. Transaction Processing Performance Council, September 2008.

[WRRA08] D. Wiese, G. Rabinovitch, M. Reichert and S. Arenswald. Autonomic Tuning Expert - A framework for best-practice oriented autonomic database tuning. In Proceedings of Centre for Advanced Studies on Collaborative Research (CASCON 2008). Ontario, Canada, October 2008.


## *Trademarks


IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. A current list of other IBM trademarks is available on the Web at: http://www.ibm.com/legal/copy-trade.shtml

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

# Value Demonstration of Embedded Analytics for Front Office Applications

Erik Nijkamp, Martin Oberhofer, Albert Maier

IBM Deutschland Research & Development GmbH
Schönaicherstrasse 220
71032 Böblingen
{nijkampe, martino, amaier}@de.ibm.com

**Abstract:** Users of front office applications such as call center or customer support applications make millions and millions of decisions each day without analytical support. For example, if a support employee gets a new support ticket and needs to decide how much time should be used for problem resolution and which measures should be taken, this is done without analytical insight. As a result, companies cannot optimize their front office departments because analytical insight derived in Business Intelligence (BI) Systems is not available to users of these applications. Our demo shows how to improve a Customer Relationship Management (CRM) System [Lin01] by embedding analytics in an "in context" and "on demand" fashion without requiring any BI System skills. "In context" means that only analytics relevant for decision making on the current UI screen is made available. "On demand" means that the user has the information accessible in "mouse-over" events, i.e. the user decides when to consume which portion of the analytical information. This avoids being flooded with information not needed. The underlying implementation uses UIMA [GS04] to determine the context. Real-time lookup services for the delivery of the analytic insight are dynamically bound to the application UI. In the demo we will show the system at work and explain the architecture, the underlying technologies, and the algorithms used for the embedded analytics. The system has been built in the context of a bachelor thesis.

## 1 System Architecture

The general architecture is shown in Figure 1. The UI of an application, here a CRM System (4), is shown in a Web browser (1) on which a specific plugin for embedded analytics has been deployed. A user executes a complaint resolution business process through executing several complaint UI screens (3). The embedded analytics components (5) are deployed in a backend application server. They are based on UIMA, specific annotators (6), and lookup services. A lookup service is a direct call through an ESB (9) to backend systems such as Master Data Management (MDM) systems (13) [DHM+8], Data Warehouse systems (12), HR systems (11), or other external systems (10). A lookup service can also be a trigger causing a complete analytical workflow to be executed on a process orchestration component (8), i.e. a series of fine granular services is invoked to compute an overall analytical result which is returned to the browser (1).



Trademarks: IBM logos are trademarks or registered trademarks of the International Business

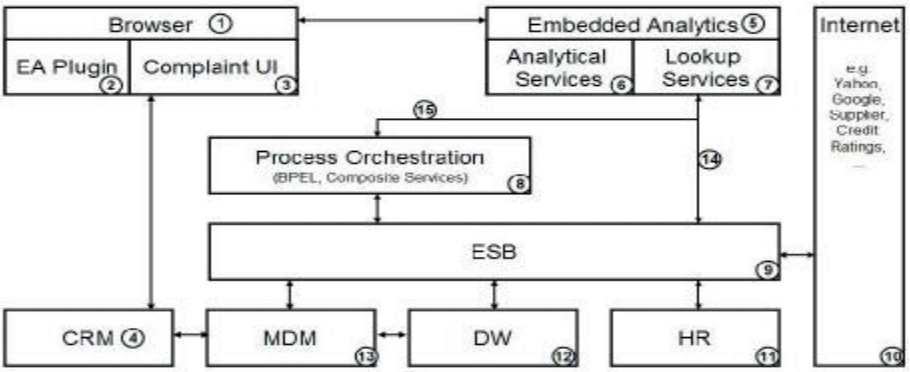



Figure 1: High Level Architecture

A more detailed view with a focus on the CRM system is shown in Figure 2. Analytical workflows in the context of a Business Process Execution Language (BPEL) [WS-BPEL] based orchestration component consume for example several SAP CRM web services to compute the overall customer segmentation result dynamically. Advantages of this workflow abstraction layer between the UI and the backend systems include:

- backend systems providing raw analytical data and the embedded analytics infrastructure are decoupled allowing changes when business priorities shift
- business analysts can graphically design the analytical workflows without a need for a developer and inject relevant analytic insight into front office applications
- front office application users can consume the graphical representation of the analytical insight in context without deep BI skills leveraging operational BI [Imh06]

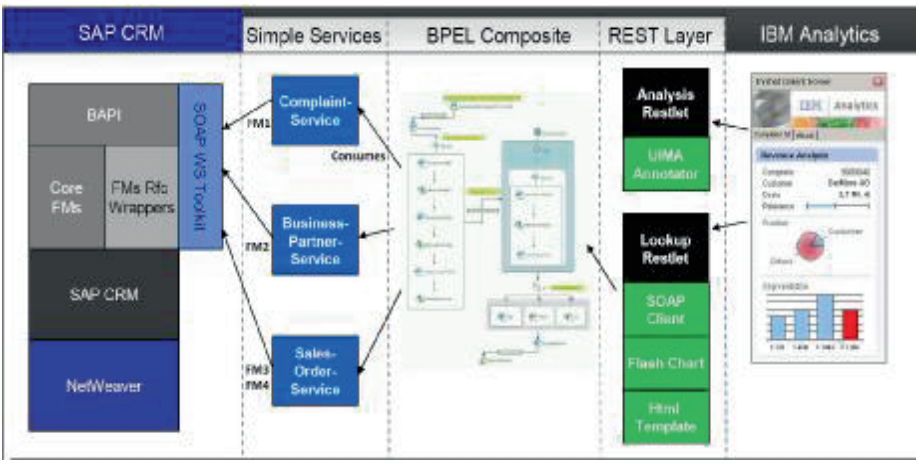



Figure 2: Components Involved in a CRM Complaint Process

The results of the analytical workflows are transformed into graphics using HTML Template and Flash Charting APIs and rendered by the browser in a sidebar. The visualization components are called through a REST API.

## 2 Demo Scenario

The demo follows the steps as shown in Figure 3.

- Step 1: The user opens a complaint in the CRM Web UI. The text shown in the UI is processed by an IBM analytics component based on UIMA.
- Step 2: If certain elements are identified by UIMA, the corresponding elements in the text are highlighted when the requested page is rendered by the browser.
- Step 3: If the user triggers the lookup of certain analytical information through a "mouse over"-event, analytical workflows and lookup services are executed.
- Step 4: With a HTML rendering component the result is prepared to be shown in a panel of the browser on the left hand side of the CRM Web UI main frame.

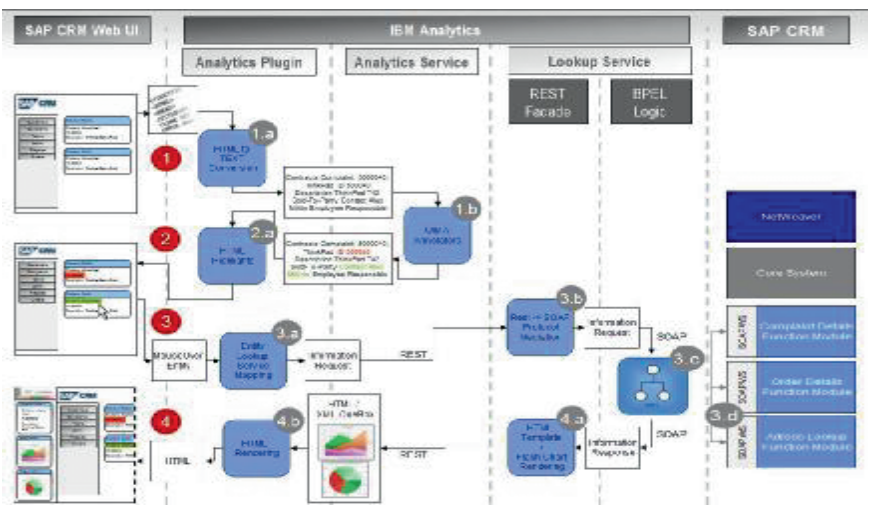



Figure 3: Component Interactions

In the demo the user will be able to connect to an SAP Netweaver CRM system using the Web UI and work with complaints. Depending which "mouse over"-event is triggered, the user will see in the browser sidebar on the left hand side various context sensitive, analytical results optimizing the decision making while processing the complaint. We will show how the system works, and explain the architecture and the underlying technologies.

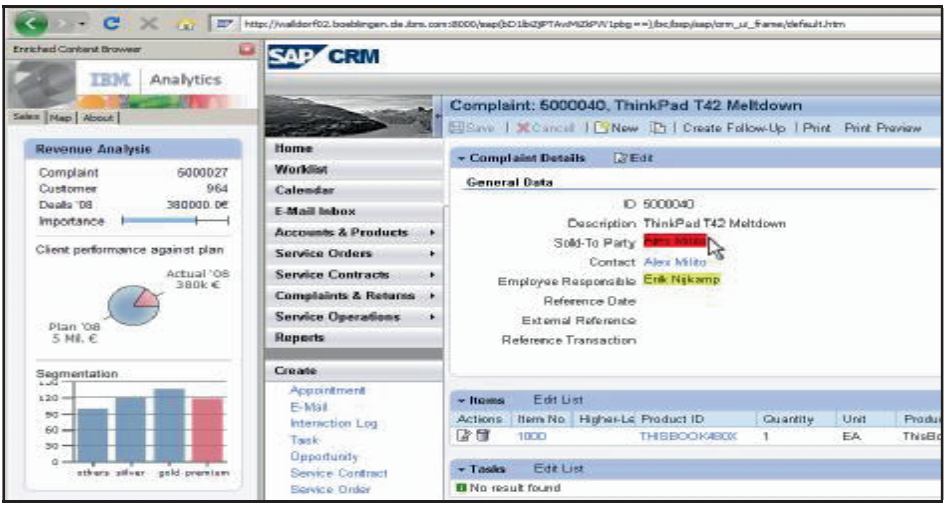



Figure 4: Example Screenshot

The screenshot in Figure 4 shows the outcome of retrieving analytical information, such as revenue and customer relevancy information, for a customer identified by his name.

## References


[DHM+08] Allen Dreibelbis, Eberhard Hechler, Ivan Milman, Martin Oberhofer, Paul van Run, Dan Wolfson. Enterprise Data Management – An SOA Approach to Managing Core Information. IBM Press, 1. Auflage, New Jersey 2008.

[GS04] Thilo Goetz, Oliver Suhre. Design and Implementation of the UIMA Common Analysis System, IBM Systems Journal 43, no.3, 2004, S. 490-515.

[Imh06] Claudia Imhoff. Operational business intelligence, http://www.teradata.com/tdmo/v06n03/Viewpoints/EnterpriseView/OBI.aspx, 9/2006.

[Lin01] Joerg Link. Customer Relationship Management. Springer-Verlag, 1. Auflage, München 2001.

[WS-BPEL] Web Service Business Process Execution Language Standard Specification. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 17.12.2008

# Support 2.0: An Optimized Product Support System Exploiting Master Data, Data Warehousing and Web 2.0 Technologies

Martin Oberhofer, Albert Maier

IBM Deutschland Research & Development GmbH
Schönaicherstrasse 220
71032 Böblingen
{martino, amaier}@de.ibm.com

**Abstract:** The proposed system integrates traditional and Web 2.0 based product support systems and uses master data management, data warehousing and text analytics functionality to send problem records to the person most capable to solve it, let it be an in-house technical support engineer or another customer in a Peer-to-Peer system. The demo system has been implemented by 3 students in the context of an Extreme Blue Project [EBP].

## 1 Introduction

There are several fundamentally different approaches to product support. The most common approach is to have a dedicated support staff and a multi-tier support process. A first tier support is usually staffed with people who are not very technical; the last tier is typically staffed with very skilled technical engineers – often members of product development. There might be service level agreements to obey, e.g. problem resolution and maximum response times might be guaranteed. We will hence-forth call this the "traditional support". Other popular approaches are online self support (e.g. via FAQs) and Peer-to-Peer support (e.g. via forums or wikis). We will hence-forth call this "online support". With such an "online support" there is no guarantee for a customer that his problem will be solved in a certain time, if at all. On the other hand, other customers who already solved this problem could provide solutions very quickly. This can increase customer satisfaction and is cost effective at the same time. Many companies today have both infrastructures – however, they are not integrated.

We are proposing a new product support system that combines the best of traditional and online support approaches. The basic idea is to have a single entry point for problem records and to implement a distribution algorithm that decides where to route a record (e.g. to a forum versus a customer relationship management system), when to escalate it (from a peer-to-peer system to the traditional support system), and where to publish problem resolutions. Assessments of the value of proposed problem resolutions and an incentive system for contributors are also part of the system.

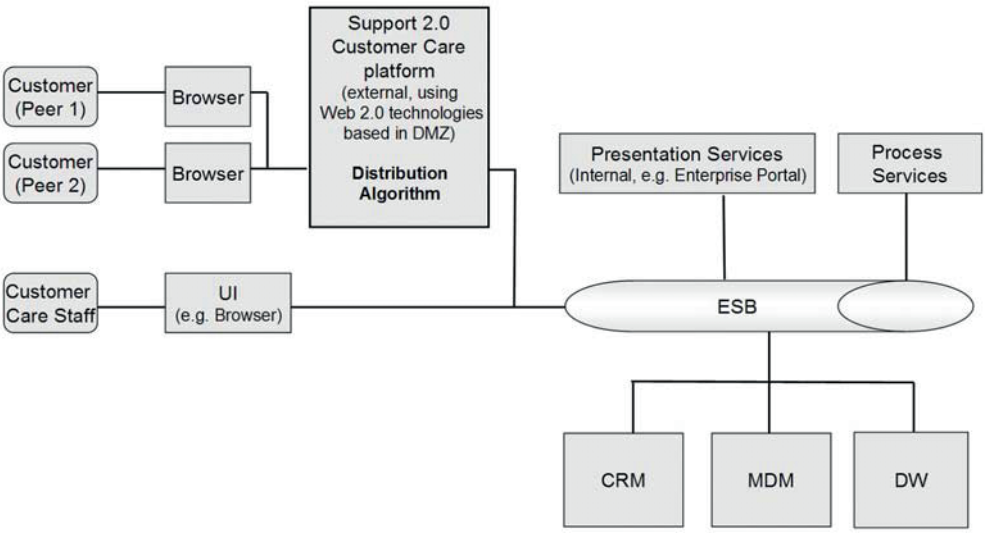



Figure 1: System Architecture

This distribution algorithm is using services from various information management systems. This includes

- a data warehouse for getting information such as customer profitability (this can influence at which point in time a support engineer will start working at a customer problem record and how much time/money he will be able to spend)
- a master data management system that guarantees high quality customer and product data (e.g. it prevents customer duplicates) and keeps information such as customer privacy and preferences settings
- a text search system for federated search over all systems storing problem record solutions (forums, wikis, blogs, customer relationship management system), and
- a text analytics system for matching key words in the problem message to themes (a classification that allows peers and support engineers to specify what kind of problem records they are interested in and able to help with)

The main advantages of such a system above the current state of the art are manifold including the potential for significant cost savings (without compromising service level agreements), faster average response times and higher average solution quality, and an increased customer satisfaction and loyalty.

## 2 System Architecture and Demo Scenario

Figure 1 shows the overall architecture of the system. There are three critical systems:

- A Customer Relationship Management (CRM) System [Lin01]: This system is used for customer care processes like complaint support processing whenever a customer reports a problem. This system represents the traditional support.

- A Master Data Management (MDM) System [DHM+8]: This system is used to manage customer and product master data with high quality efficiently. It provides data to the CRM System and the Support 2.0 Customer Care platform. It furthermore manages the peer status, and privacy and preference settings for all participants of the Support 2.0 Customer Care platform.

- The Support 2.0 Customer Care platform: This is an online platform used by end customers as part of our new method supporting new interaction paradigms in customer support. It includes the distribution algorithm, a forum infrastructure and Web 2.0 capabilities (blogs, wikis, etc.), an infrastructure to rate peers and offer incentives, a search engine for integrated search across all content sources (forums, CRM system, …), and a monitoring component for checking problem expiry thresholds used for guaranteeing service level agreements.

Figure 2 is sketching the problem record distribution algorithm. Rectangles with round corners represent the problem message respectively information added to it, diamonds represent processing steps, rectangles with corners represent specific input information for the respective processing step.

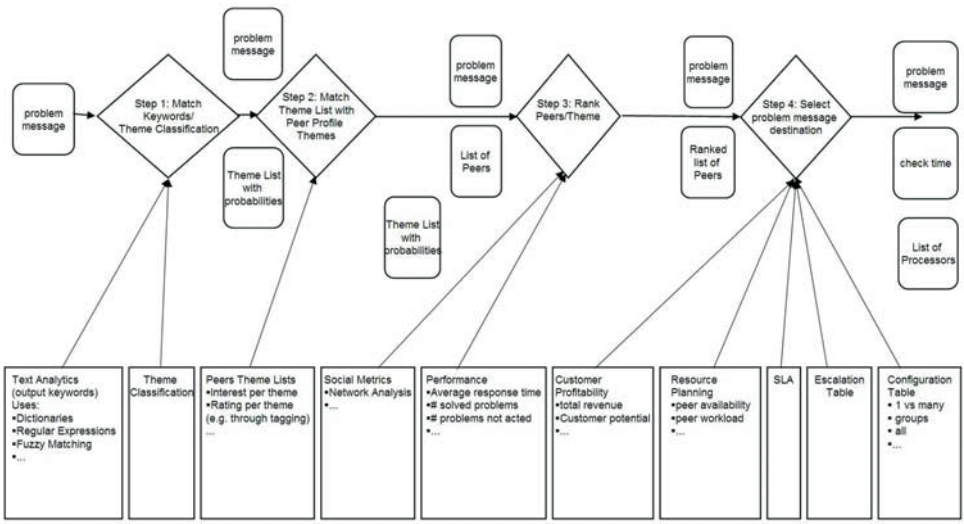



Figure 2: Distribution Algorithm

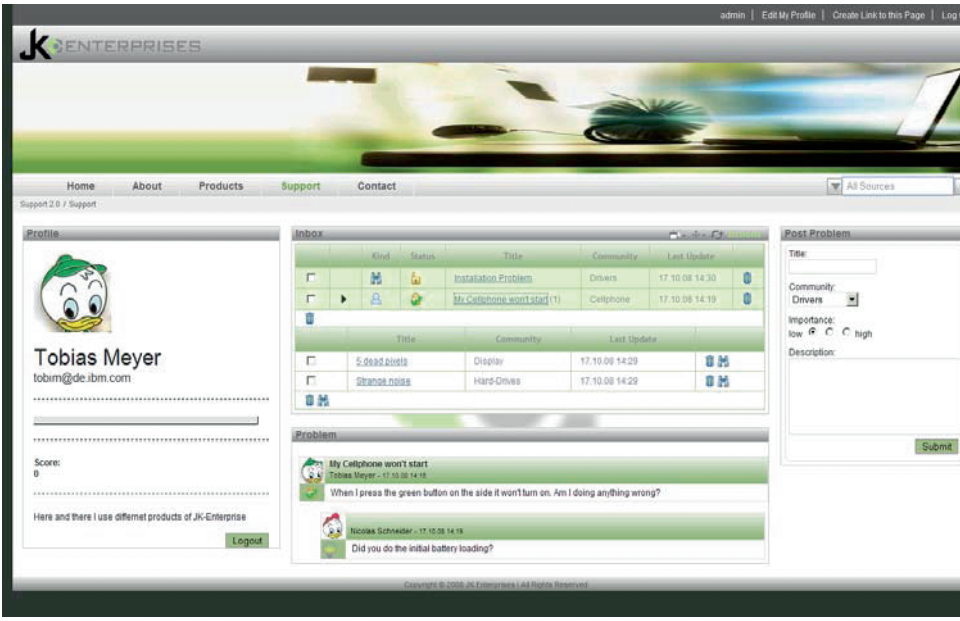



Figure 3: Sample Demo Screen

The algorithm starts when a problem message is received. The text analytics system extracts keywords and checks them against a theme classification. The resulting theme list is matched against peer profiles. Taking into consideration social network metrics and performance metrics, a ranked list of peers is determined. The next step is to determine which peer(s) should get this problem message (peer includes in-house technical support engineers). This decision takes into account input parameters like customer profitability, service level agreements, and resource planning information.

In the demo, the user will be able to use different roles such as problem searcher, peer answering a problem, and technical support engineer to explore the various steps of this new support process. The demo user can therefore post a problem, answer a problem, and rate an answer. The entry point for issuing a problem record is a portal based UI (see figure 3). We will show the system at work, and explain the architecture, the distribution algorithm and the underlying products and technologies.

## References

[DHM+08] Allen Dreibelbis, Eberhard Hechler, Ivan Milman, Martin Oberhofer, Paul van Run, Dan Wolfson. Enterprise Data Management – An SOA Approach to Managing Core Information. IBM Press, 1. Auflage, New Jersey 2008.

[EBP] IBM Deutschland Research & Development GmbH: Extreme Blue. http://www-05.ibm.com/de/entwicklung/extremeblue/, 17.12.2008.

[Lin01] Joerg Link. Customer Relationship Management. Springer-Verlag, 1. Auflage, München 2001.

# DBTT-Tutorial

# Software as a Service: Do It Yourself or Use the Cloud

Dean Jacobs
Chief Development Architect
SAP AG

**Abstract**: In the Software as a Service (SaaS) model, a service provider owns and operates an application that is accessed by many businesses over the Internet. A key benefit of this model is that, by careful engineering, it is possible to leverage economy of scale to reduce total cost of ownership relative to on-premises solutions. This tutorial will describe basic architectures and best practices for implementing enterprise SaaS applications. It will cover both first generation systems, which are based on conventional databases and middleware, as well as second generation systems, which are based on emerging cloud computing platforms. The discussion will include the following topics.

- The Business of SaaS. The tutorial will include a summary of the kinds of SaaS applications that are available today, their relative market shares, and the demographics of their customers. This information is crucial to the design of the SaaS application infrastructure because, by offering less functionality, it is generally easier to increase scalability and lower costs.

- Do It Yourself. The tutorial will outline best practices for implementing SaaS applications on conventional databases and middleware. The challenges of multi-tenancy, including managing resource contention and supporting tenant-level application extensions, will be discussed along with possible solutions. Several cases studies will be presented.

- Use the Cloud. The tutorial will describe emerging cloud computing platforms and the challenges in using them to implement enterprise applications. A primary issue in this regard is that these platforms generally provide little support for transactions and concurrency control. It seems clear that additional capabilities will have to be provided, but it is not clear how weak they can be.

## *GI-Edition Lecture Notes in Informatics*

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Busi-ness 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Erfahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometric and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

The titles can be purchased at:

**Köllen Druck + Verlag GmbH**
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de