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 978-3-88579-216-1

This volume contains papers from the workshops of the Software Engineering 2008 conference, held in Munich February 18 to 22, 2008. The topics covered in the papers range from development processes, automotive software engineering, software product lines, requirement engineering, knowledge sharing in software development to model driven architecture, safety critical systems and software testing.

Walid Maalej, Bernd Bruegge (Hrsg.): Software Engineering 2008

122

GI-Edition

Lecture Notes in Informatics

Walid Maalej,
Bernd Bruegge (Hrsg.)

Software Engineering 2008 - Workshopband

Fachtagung des GI-Fachbereichs Softwaretechnik

Proceedings

Walid Maalej, Bernd Bruegge (Hrsg.)

Software Engineering 2008
-
Workshopband

Fachtagung des GI-Fachbereichs Softwaretechnik

18. - 22.02.2008
in München

Gesellschaft für Informatik e.V. (GI)

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

Volume P-122

ISBN 978-3-88579-216-1
ISSN 1617-5468

Volume Editors
Walid Maalej
Fakultät für Informatik
Technische Universität München
85748 Garching, Germany
E-Mail: maalejw@in.tum.de
Bernd Bruegge
Fakultät für Informatik
Technische Universität München
85748 Garching, Germany
E-Mail: bruegge@in.tum.de

Series Editorial Board
Heinrich C. Mayr, Universität Klagenfurt, Austria (Chairman, mayr@ifit.uni-klu.ac.at)
Jörg Becker, Universität Münster, Germany
Hinrich Bonin, Leuphana-Universität Lüneburg, Germany
Dieter Fellner, Technische Universität Darmstadt, Germany
Ulrich Flegel, SAP Research, Germany
Johann-Christoph Freytag, Humboldt-Universität Berlin, Germany
Ulrich Furbach, Universität Koblenz, Germany
Michael Koch, TU München, Germany
Axel Lehmann, Universität der Bundeswehr München, Germany
Peter Liggesmeyer, TU Kaiserslautern und Fraunhofer IESE, Germany
Ernst W. Mayr, Technische Universität München, Germany
Heinrich Müller, Universität Dortmund, Germany
Sigrid Schubert, Universität Siegen, Germany
Martin Warnke, Leuphana-Universität Lüneburg, Germany

Dissertations
Dorothea Wagner, Universität Karlsruhe, Germany
Seminars
Reinhard Wilhelm, Universität des Saarlandes, Germany

© Gesellschaft für Informatik, Bonn 2008
printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Der Fachbereich 2 „Softwaretechnik“ der Gesellschaft für Informatik (GI) veranstaltete die jährliche Konferenzserie „Software Engineering“ vom 18. bis 22. Februar 2008 an der Technischen Universität München. Diese alljährlich stattfindende Tagung, bei der sich Fachleute aus Wissenschaft und Industrie treffen, um über aktuelle Themen aus dem Gebiet der Softwaretechnik zu berichten und zu diskutieren, ist mittlerweile eine wichtige Plattform für deutschsprachige Informatiker und Software-Ingenieure geworden.

Die SE 2008 gliederte sich in mehrere Teile. In den Workshops wurden neue Ansätze und Techniken aus dem Bereich der Softwaretechnik diskutiert und vorgestellt. In den Tutorien wurden aktuelle Themen vertieft vermittelt. Die eingeladenen Vorträge im Industrie-Forum adressierten vor allem aktuelle Probleme aus der Software-Industrie. Das Ziel dabei war, den Dialog zwischen Praktikern und Theoretikern zu fördern, um neue Ideen und Lösungswege zu identifizieren. In der wissenschaftlichen Tagung wurden eingereichte und nach einer umfangreichen Begutachtung angenommene Beiträge vorgestellt.

Mit Freude haben wir das große Interesse an den Workshops vermerkt. Workshops sind oft die Lieferanten und die Quelle frischer Ideen. Die eingereichten Vorschläge zu den verschiedensten Schwerpunkten durchliefen ebenfalls eine Begutachtung. Nur die akzeptierten Beiträge wurden bei den Workshops vorgestellt. Nach der Durchführung der Workshops haben wir die Autoren gebeten, ihre Beiträge aufgrund der anregenden Diskussionen während der Workshops noch einmal zu überarbeiten.

Mit dieser Lösung folgen wir einem neuen Weg. Wir ermöglichen die Veröffentlichung von neuen Ideen, die sich bereits einer Begutachtung und intensiven Diskussionen gestellt haben. Aus diesem Grund legen wir die revidierten Workshop-Beiträge in diesem separaten Tagungsband nach der Konferenz vor.

Mein Dank geht an Thomas Groß, der den Vorsitz für die Workshops im Programmkomitee innehatte. Mein besonders herzlicher Dank gilt Walid Maalej, der die Koordination der Workshops und die Zusammenstellung der Beiträge für diesen Band übernommen hat.

Bernd Bruegge
Tagungsleitung der SE 2008

München, im März 2008

Tagungsleitung
Prof. Bernd Bruegge, Technische Universität München

Tagungsorganisation
Lilith Al-Jadiri, Jörn David, Jonas Helming, Korbinian Herrmann, Maximilian Kögel, Walid Maalej, Monika Markl, Maximilian Reiß, Helma Schneider, Uta Weber.
(alle Technische Universität München)

Programmkomitee
Bernd Brügge; Technische Universität München (Vorsitz)
Uwe Aßmann; Technische Universität Dresden
Colin Atkinson; Universität Mannheim
Klaus Beetz; Siemens AG München
Andreas Braun; T-Systems
Ruth Breu; Universität Innsbruck
Manfred Broy; Technische Universität München
Oliver Creighton; Siemens AG
Armin B. Cremers; Rheinische Friedrich-Wilhelms-Universität Bonn
Jürgen Ebert; Universität Koblenz-Landau
Gregor Engels; Universität Paderborn
Harald Gall; Universität Zürich
Martin Glinz; Universität Zürich
Michael Goedicke; Universität Duisburg-Essen
Thomas Groß; ETH Zürich (Vorsitz Workshops)
Volker Gruhn; Universität Leipzig
Wilhelm Hasselbring; Carl-von-Ossietzky-Universität Oldenburg
Heinrich Hußmann; Ludwig-Maximilians-Universität München (Vorsitz Tutorien)
Matthias Jarke; RWTH Aachen
Udo Kelter; Universität Siegen
Eva-Maria Kern; Universität der Bundeswehr, München
Hemut Krcmar; Technische Universität München
Claus Lewerentz; Brandenburgische Technische Universität Cottbus
Horst Lichter; RWTH Aachen
Peter Liggesmeyer; Universität Kaiserslautern
Florian Matthes ; Technische Universität München
Peter Müller; Microsoft
Manfred Nagl; RWTH Aachen
Barbara Paech; Universität Heidelberg
Klaus Pohl; Universität Duisburg-Essen
Gustav Pomberger; Johannes-Kepler-Universität Linz
Jörg Raasch; HAW Hamburg
Joachim Reichel; Wacker IT München

Ralf Reussner; Universität Karlsruhe
Francesca Saglietti; Friedrich-Alexander-Universität Erlangen-Nürnberg
Wilhelm Schäfer; Universität Paderborn
Andy Schürr; Technische Universität Darmstadt
Hans-Werner Six; Fernuniversität Hagen
Gregor Snelting; Universität Passau
Walter F. Tichy; Universität Karlsruhe
Markus Voß ; sd&m AG
Mario Winter; Fachhochschule Köln
Martin Wirsing; Ludwig-Maximilians-Universität München
Andreas Zeller; Universität des Saarlandes
Wolf Zimmermann; Universität Halle
Heinz Züllighoven; Universität Hamburg
Albert Zündorf; Universität Kassel

Offizieller Veranstalter
Fachbereich Softwaretechnik der Gesellschaft für Informatik (GI)

Schirmherr
Staatsminister Erwin Huber, Staatsministerium für Wirtschaft, Infrastruktur, Verkehr und Technologie

Unterstützt wird die Tagung zudem von:
Schweizer Informatik Gesellschaft (SI)
Oestereichische Computer Gesellschaft (OCG)

Platinsponsor

Goldsponsoren

Silbersponsoren

Sachsponsoren

Inhaltsverzeichnis

Agile Knowledge Sharing for Distributed Software Teams

Workshop Agile Knowledge Sharing for Distributed Software Teams 19
Björn Decker, Hans-Jörg Happel, Walid Maalej

Collaborative Development of Knowledge Bases in Distributed Requirements Elicitation 22
Steffen Lohmann, Thomas Riechert, Sören Auer, Jürgen Ziegler

Making Wikis productive as the glue code of project information (Project WAVES) 29
Tim Romberg, Peter Szulman

Enabling Agile Offshoring with the Dual-Shore Model 35
Joachim Sauer

From Research to Practice: How do TXT e-solutions Plan to Deploy Innovations in Sharing Development Knowledge 43
Alessandra Bagnato, Walid Maalej

A Tool with which to Recommend Knowledge Sources in Virtual Communities 51
Juan Pablo Soto, Aurora Vizcaíno, Javier Portillo-Rodríguez, Mario Piattini

Navigation Recommendation On Knowledge Artifacts 59
Joern David

Video-based Information Sharing in Distributed Teams 67
Maximilian Reiss

Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Developement 72
Patrick Mukherjee , Aleksandra Kovacevic , Andy Schuerr

Peer-to-Peer Issue Tracking System: Challenges and Solutions 77
Vijay Srinivas Agneeswaran, Rammohan Narendula, Karl Aberer

Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis

Workshop Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis 85
Bernhard Hohlfeld, Stefan Kowalewski, Michael Reinfrank, Gerhard Wirrer

Konzeption eines Austauschformates für AUTOSAR Diagramme 88
Vladimir Stantchev, Dawid Kostrzycki

Defining a Timing Model for AUTOSAR – Status and Challenges 93
Kai Richter

The legacy ECU software problem - approach and research challenges 98
Thomas Heinz, Jörn Schneider

BRIEF: Systematisches Prozess-Briefing und –Debriefing. Ein bei der Erstellung komplexer Automobilsoftware praxiserprobtes Entwicklungsvorgehen, das Elemente klassischer und agiler Modelle kombiniert 103
Hannes Omasreiter, Claudia Schlumpberger, Lorenz Widmann

Vom modellbasierten Architekturentwurf über die Simulation verteilter Systeme zum Serienprojekt 108
Rainer Otterbach, Oliver Niggemann

Model-Based Design for IEC 61508: Towards Translation Validation of Generated Code 113
Mirko Conrad

Effizienzsteigerung im Systemtest von Steuergeräten Eine empirische Untersuchung 118
Matthias Wiemann, Jörg Gericke

Integration von Prozessen und Werkzeugen im agilen Zeitalter

Workshop zur Integration von Prozessen und Werkzeugen im agilen Zeitalter 125
Walid Maalej, Oliver Creighton, Ernst Pohn

Integrating Software Lifecycle Models into a uniform Software Engineering Model 128
Jonas Helming

Integration des V-Modell®XT im Visual Studio Team Foundation Server – Erfahrungen aus dem Projekt CollabXT 135
Marco Kuhrmann

Integrationsnotwendigkeit im agilen Zeitalter 141
Jennifer Schiller

Das Link-System: Von Weblinks zu universellen, sensitiven Hyperlinks 149
Dirk Fischer

TIME - Tracking Intra- and Inter-Model Evolution 157
Maximilian Kögel

Change Propagation with the Change Notification Bus 165
Tobias Rodenbach, Lena Wiese

Modellgetriebene Softwarearchitektur – Evolution, Integration und Migration

Workshop Modellgetriebene Softwarearchitektur – Evolution, Integration und Migration (MSEIM 2008) 173
Michael Goedicke, Maritta Heisel, Sascha Hunold, Stefan Kühne, Matthias Riebisch, Niels Streekmann

Using collaborative business process models for the configuration of distributed workflow environments 178
Dirk Werth, Jörg Ziemann, Philipp Walter, Peter Loos

MDA und KI: Domänenspezifische Modellierung und Umsetzung wissensintensiver Prozesse 184
Steffen Kruse, Malte Zilinski, Hilke Garbe, Claus Möbus

Systematische Berücksichtigung von Abhängigkeitsbeziehungen bei Architekturentscheidungen 191
Sven Wohlfarth, Matthias Riebisch

Grafische Validierungsregeln am Beispiel von EPKs 198
Sven Feja, Daniel Fötsch, Sebastian Stein

Towards Evaluating Maintainability Within Model-Driven Environments 205
Thomas Goldschmidt, Jens Kübler

UPPAAL-Modelle als ausführbare Spezifikation in Java 212
Michael Goedicke, Moritz Balz, Michael Striewe

Modellgetriebene Integration von Altsystemen 219
Dieter Hildebrandt, Michael Gründler, Heiner Feislachen

Inkrementelle Transformation einer monolithischen Geschäftssoftware 226
Sascha Hunold , Matthias Korch , Björn Krellner, Thomas Rauber, Thomas Reichel, Gudula Rünger

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation PIK2008 Workshop-Einleitung und Überblick 235
Andreas Birk, Ludger Fiege, Klaus Schmid

Relating Product Line Context to Requirements Engineering Processes Using Design Rationale 240
Samuel Fricker, Reinhard Stoiber

Varianten in der Automobilelektronikentwicklung Herausforderungen und Lösungsansätze 252
Martin Becker, Matthias Weber, Thomas Wierczoch

Multiple software product lines in automotive software development 261
Uwe Beher, Günter Bönisch, Mike Heidrich

The Sales Scenario: A Model-Driven Software Product Line 273
Henrik Lochmann, Birgit Grammel

Modellierung von Variabilität in der Geschäftsanalyse – eine industrielle Fallstudie 285
Thomas von der Maßen, Andreas Wübbeke

Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme

Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme: Bedeutung modellbasierter und formaler Ansätze für effiziente Entwicklung und Zertifizierung 299
Hardi Hungar, Erwin Reyzl

Die Zertifizierung des Modellierungswerkzeugs ASCET nach der IEC 6 15 08 303
Peter Dencker, Tilman Glötzner

Unified Modeling Style and Unified Process Architecture: Model-Driven Design and Model-Based Verification key to safety and cost-reduction 307
Ralf Fachet

Safety Critical Java erleichtert die Zertifizierung sicherheitskritischer Echtzeit-Anwendungen 311
Thomas Henties

Erfolgsfaktoren bei der Entwicklung der OBCE-WW 316
Thomas Buchberger

Modellsicht auf die Logik eines generischen Stellwerkssystems 319
Serhat Cakaloglu, Hans-Jürgen Nollau, Murat Sahingöz

Secure Coding als Bindeglied zwischen formalen Ansätzen um Safety in Infrastruktur-kritischen Systemen zu gewährleisten 323
Wolfgang Köppl, Gordon Rohrmair

Nachweis von Sicherheitseigenschaften in modellbasierten Entwurfsprozessen 329
Thomas Peikenkamp

Zertifizierung einer Sicherungskomponente mittels durchgängig formaler Modellierung 335
Udo Frese, Daniel Hausmann, Christoph Lüth, Holger Taubig, Dennis Walter

Positionsstatement: Zertifizierungsargumentationen mit mathematischer Präzision 339
Hardi Hungar

Positionsstatement: Über die Rolle von formalem Beweis und Entwicklung im Zertifizierungsprozeß 341
Christoph Lüth

Schwachpunkte und Grenzen gegenwärtiger Methoden 343
Theodor Tempelmeier

Ergebnisse des Workshops „Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme“ 345
Hardi Hungar, Erwin Reyzl

Testmethoden für Software - Von der Forschung in die Praxis

2. Workshop Testmethoden für Software - Von der Forschung in die Praxis 353
Fevzi Belli, Christof J. Budnik, Andreas Spillner

Modellbasierter Test mit FitNesse – Ein Erfahrungsbericht aus der Praxis 355
Dehla Sokenou

Plattformübergreifende funktionale Tests durch GUI-Abstraktion 359
Benno Markiewicz, Holger Flemig, Karsten Weicker

Vergleich einer graphenbasierten Methode mit der Klassifikationsbaummethode für die Testfallermittlung anhand einer automotiven Fallstudie 367
Friedrich Beidinger, Axel Hollmann, Markus Kleinselbeck, Wolf Ritschel

A TTCN-3-based Web Service Test Framework 375
Edith Werner, Jens Grabowski, Stefan Troschütz, Benjamin Zeiss

Die Einführung eines wieder verwendbaren Testframeworks in der Sozialversicherung 383
Armin Beer

Test Case Structuring and Execution Control in an Integration Framework for Heterogeneous Automatic Software Tests 387
Andreas Ganser, Holger Schackmann, Horst Lichter, Heinz-Josef Schlebusch

Ein Framework für die Testautomatisierung bei Fahrer-Assistenz-Systemen 395
Vladimir Entin, Klaus Meyer-Wegener

Using UML Environment Models for Test Case Generation 399
Maritta Heisel, Denis Hatebur, Thomas Santen, Dirk Seifert

Glassboxtest zur Testsuite-Optimierung 407
Rainer Schmidberger

Automatic Generation of Optimized Integration Test Data by Genetic Algorithms 415
Florin Pinte, Francesca Saglietti, Norbert Oster

Automatic Test Case Generation from UML Models and OCL Expressions 423
Stephan Weißleder, Dehla Sokenou

A Domain Specific Language for Uncovering Abstract Protocols and Testing Message Scenarios 427
Tim Reichert, Dominikus Herzberg

Path Constraint Construction with Lookahead 431
Ralf Gerlich

Simplifying the Description and Application of Tests 435
Daniel Brenner, Colin Atkinson

Statistical Testing within the Scope of the System Validation of a RIS/PACS System 439
Stefan Holpp, Helmut Goetz, Winfried Dulz

Static Bug Detection Through Analysis of Inconsistent Clones 443
Elmar Juergens, Benjamin Hummel, Florian Deissenboeck, Martin Feilkas

Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung

2. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung 449
Joerg Doerr, Peter Liggesmeyer

Assessing Relations between Non-Functional Requirements 451
Michael C. Jaeger, Anne Hoffmann

Qualitätsrisiken und deren Abhängigkeiten 457
Andrea Herrmann

Erfassung, Strukturierung und Überprüfung von Qualitätsanforderungen durch aktivitätenbasierte Qualitätsmodelle 464
Stefan Wagner, Florian Deissenboeck, Sebastian Winter

Über die Auswirkungen nicht-funktionaler Anforderungen beim Entwurf eines Rahmenwerkes zur Anbindung von Smart Items an ERP-Systeme 470
Rosemaria Giesecke, Florin Spanachi

Agile Knowledge Sharing for Distributed Software Teams

(AKNOWLEDGE' 08)

Inhaltsverzeichnis

Workshop Agile Knowledge Sharing for Distributed Software Teams 19
Björn Decker, Hans-Jörg Happel, Walid Maalej

Collaborative Development of Knowledge Bases in Distributed Requirements Elicitation 22
Steffen Lohmann, Thomas Riechert, Sören Auer, Jürgen Ziegler

Making Wikis productive as the glue code of project information (Project WAVES) 29
Tim Romberg, Peter Szulman

Enabling Agile Offshoring with the Dual-Shore Model 35
Joachim Sauer

From Research to Practice: How do TXT e-solutions Plan to Deploy Innovations in Sharing Development Knowledge 43
Alessandra Bagnato, Walid Maalej

A Tool with which to Recommend Knowledge Sources in Virtual Communities 51
Juan Pablo Soto, Aurora Vizcaíno, Javier Portillo-Rodríguez, Mario Piattini

Navigation Recommendation On Knowledge Artifacts 59
Joern David

Video-based Information Sharing in Distributed Teams 67
Maximilian Reiss

Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Developement 72
Patrick Mukherjee , Aleksandra Kovacevic , Andy Schuerr

Peer-to-Peer Issue Tracking System: Challenges and Solutions 77
Vijay Srinivas Agneeswaran, Rammohan Narendula, Karl Aberer

Workshop
Agile Knowledge Sharing for Distributed Software Teams

Björn Decker[1], Hans-Jörg Happel[2], Walid Maalej[3]

[1] empolis GmbH
Europaallee 10. D-67657 Kaiserslautern
Bjoern.Decker@empolis.com

[2] FZI Forschungszentrum Informatik
Haid-und Neu-Str. 10-14, D-76131 Karlsruhe
happel@fzi.de

[3] Technische Universität München
Boltzmannstr. 3, D-85748 Garching
maalejw@in.tum.de

1 Motivation

The nature of software engineering projects is rather complex. They involve people with different backgrounds, working in different phases and activities. Software engineering is a field where constant technology change takes place, rendering the work of involved people highly dynamic, in the sense that discovering solutions for new problems belongs to the daily tasks. Knowledge in software engineering is diverse and its proportions are immense and continuously expanding. Important needs that drive the use of knowledge management are capturing and sharing process and product knowledge, acquiring knowledge about the application domain and new technologies, as well as knowing who knows what.

In order to address these issues, software engineering practitioners have adopted various forms of organizational practices, collaboration technologies and knowledge management systems such as lessons learned databases. However, it turns out that those often centrally organized and heavyweight solutions are difficult to apply and maintain in distributed development settings with frequently changing collaborators from different organizations [RM06]. Accordingly, such dynamic distributed development settings – as also found in the Open Source communities – raise new challenges for collaboration and knowledge sharing for software teams:

- Distributed development hinders knowledge sharing because of reduced communication bandwidth. Despite progress in communication technology, distributed teams have less context information and encounter more awareness difficulties. Research has shown that distributed teams are less efficient than collocated ones [HM03, SB98].

- Agile methodologies embrace tight collaboration and informal communication, manifesting in practices such as frequent "Scrum" meetings or pair programming. These practices dismiss "unnecessary" formal overhead like extensive documentation.

Traditional centralized knowledge management solutions fail to address these challenges. They require large upfront configuration and investment as well as stable, long-term environments. However, in dynamic software projects, developers avoid spending extra effort for following formal and extensive knowledge management policies. Knowledge Management methods and tools have to adapt themselves with the current development practices.

2 Topics

Despite of the practical relevance, current research around these topics is fragmented. Various disciplines and communities, such as the Artificial Intelligence, Knowledge Management and Software Engineering ones, have overlapping interests. However proposed approaches are mostly undertaken from only one perspective. Moreover, different research methods such as empirical, process-oriented or tool-oriented ones, lead to unrelated, partial and sometimes even contradictory results.

The workshop "Agile Knowledge Sharing for Distributed Software Teams" [AK07] explores enablers of new, agile approaches for coordination and knowledge sharing in distributed teams. Major topics addressed at the workshop are:

- Lightweight and unobtrusive tools for knowledge capture and sharing in distributed teams.

- Methods for supporting knowledge sharing in agile and/or distributed development teams.

- Developer observation tools and frameworks for context awareness.

- Tools for intelligent developer assistance, such as search and recommendation tools.

- Empirical studies of developers' collaboration and information behaviour.

- Scientific analysis of the relation between agile processes and knowledge management (e.g. witch activities in agile processes enable knowledge capture and sharing).

In the workshop, we aim at bringing together practitioners and researchers from various disciplines and backgrounds. Thereby, one major goal is to establish a community around the crosscutting topic of Agile Knowledge Sharing in Distributed Software Teams.

References

[AK07] Aknowledge Plattform http://www.aknowledge.org

[HM03] Herbsleb, James D.; Mockus, Audris: Formulation and preliminary test of an empirical theory of coordination in software engineering. In: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, ACM Press, 2003, pp. 138-137.

[SB98] Seaman, Carolyn B.; Basili, Victor R.: Communication and Organi-zation: an Empirical Study of Discussion in Inspection Meetings. In: IEEE Transactions on Software Engineering 24 (1998), No. 7, pp. 559-572.

[RM06] Ramesh, B., Cao, L., Mohan, K., and Xu, P. 2006. Can distributed software development be agile?. Commun. ACM 49, 10 (Oct. 2006), 41-46.

Collaborative Development of Knowledge Bases in Distributed Requirements Elicitation

Steffen Lohmann[1], Thomas Riechert[2], Sören Auer[2], Jürgen Ziegler[1]

[1]University of Duisburg-Essen
Department of Informatics and Applied Cognitive Science
Lotharstrasse 65, 47057 Duisburg, Germany
{lohmann, ziegler}@interactivesystems.info

[2]University of Leipzig
Department of Computer Science
Johannisgasse 26, 04009 Leipzig, Germany
{riechert, auer}@informatik.uni-leipzig.de

Abstract: One of the main challenges in distributed software development is the elicitation and management of knowledge regarding system requirements. Due to spatial distribution of involved parties, many limitations concerning interaction, communication, and conceptualization have to be faced. The *SoftWiki* project aims to provide an agile, wiki-based methodology to overcome these limitations in part. This paper introduces the *SoftWiki* approach and presents some of the tools that are developed to support knowledge sharing in distributed requirements engineering.

1 Motivation

Knowledge-intensive activities in distributed software development such as requirements, configuration, and change management are not sufficiently covered by existing knowledge management solutions thus far [KT05]. Particularly in settings where knowledge is spread over diverse participants, established methodologies and tools mostly fail to adequately support collaboration. However, a consolidated conceptualization of the domain in focus and a shared understanding of the planned software product are crucial to the success of the project [HL01].

The aim of the cooperative research project *SoftWiki*[1] is to support these activities, in particular with respect to system requirements. Potentially very large and spatially distributed stakeholders shall be enabled to collect, semantically enrich, classify and aggregate knowledge regarding requirements. Attempts from the domains of software engineering, knowledge management, and the semantic web are combined in a comprehensive framework. In the following, we describe preliminary results of the project, particularly concerning the developed solutions to support agile knowledge sharing.

[1] Research project funded by the German Federal Ministry of Education and Research, see: http://softwiki.de

2 Collaborative Development of a Requirements Knowledge Base

Figure 1 illustrates the overall SoftWiki architecture. The interaction between system users and developers is at the heart of the approach. Users are enabled to directly participate in the development of knowledge bases for system requirements, whereas developers support the users and moderate the collaborative activities. The effort and formal overhead for stating or modifying some piece of knowledge is minimized due to the adoption of the wiki paradigm [LC01] and its underlying philosophy of 'making it easy to fix mistakes, rather than making it hard to make them'. Stakeholders can collaboratively externalize their knowledge in natural language without any formal restrictions. Additional semantics are added by linking the contents with ontological classes and instances.

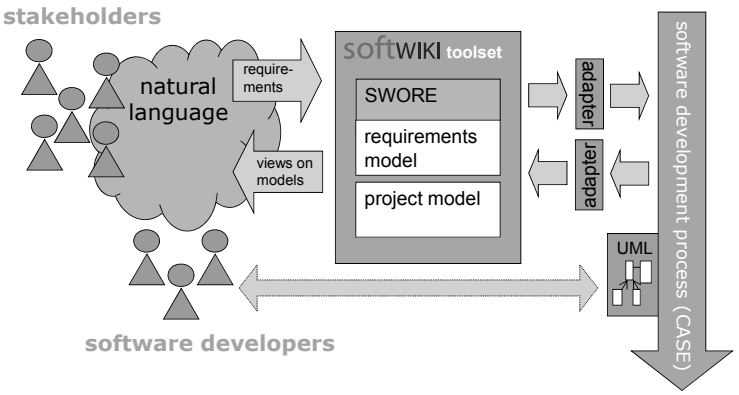


Figure 1: General SoftWiki architecture

The SoftWiki knowledge repository consists of different modeling perspectives. Essentially, these are the requirements and the project model, with the latter containing knowledge about the domain in focus, the product itself, and its context. In order to facilitate participation of various stakeholder groups, the knowledge repository can be accessed in several ways. Further knowledge sources for requirements such as relevant documents or system descriptions can be included via appropriate adaptors.

It is not the goal of the SoftWiki project to replace established CASE tools with a wiki-based approach. It rather aims to support agile collaboration in early stages of distributed requirements engineering that are not sufficiently covered by available tools thus far. Hence, the collaboratively created knowledge base can be made available to further development activities via specific adaptors, at best resulting in a 'knowledge flow' that constantly accompanies the software development cycle.

The underlying semantic structure is provided by the *SoftWiki Ontology for Requirements Engineering* (*SWORE*) that has been developed in accordance with established practices of the requirements engineering community (see [Ri07] and Figure 2 for an illustration of the SWORE core). It provides an ontological schema for

instances of the types *goal*, *scenario* and *requirement*. These instances have relations of the type *defined by* to *stakeholders* and are interconnected by the relations *details* and *detailed by*. In addition, SWORE offers the opportunity to link requirements with domain concepts or parts of already existing system models by using relations of the type *refers to* (see [RL07]).

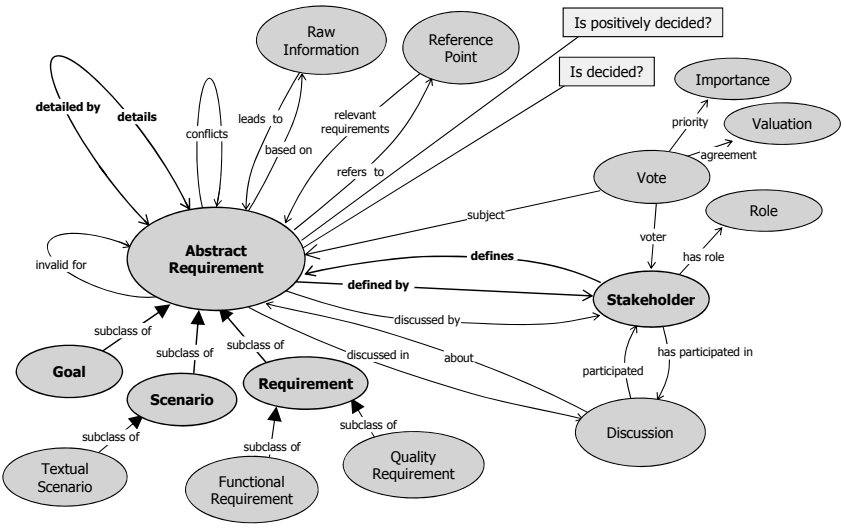


Figure 2: SoftWiki Ontology for Requirements Engineering

3 Applying a Semantic Wiki to Requirements Engineering

Within the SoftWiki project we adopt and extend the semantic wiki *OntoWiki* [ADR06] enabling it to support knowledge management regarding system requirements[2]. As OntoWiki is completely web based, it can be accessed from any location without installation simply by using a web browser.

Figure 2 presents a screenshot of OntoWiki as it is currently used to manage requirements. The user interface is divided into three areas. The left area provides features for selection of knowledge bases and ontological classes. In this case, the knowledge base of an e-government use case ("http://req.lecos-gmbh.de/fabius/") has been selected that corresponds to the SWORE ontology schema ("http://ns.softwiki.de/req/"). Accordingly, the corresponding class structure is given showing in brackets the number of instances for each class in this knowledge base. Classes without instances are hidden from the user. For instance, the presented use case does not have any instances of the type *goal* or *scenario*. The middle area provides several views on the instances and

[2] *OntoWiki* is based on the ontology authoring and management framework *Powl*. It is open source and available for download at: http://powl.sf.net. A demo installation including some knowledge bases can be accessed at: http://demo.ontowiki.net/.

attributes as well as inline editing possibilities. In the screenshot, a functional requirement instance related to user roles and access control concerning the e-government application is presented. Additionally, its attributes *description*, *comment* and *label* as well as its relations *defined by*, *detailed by* and *refers to* are displayed. Instances that are related to the selected one are shown in more detail by clicking on the plus icons. Other views can be selected via tabbed navigation, such as a map, a calendar, or a view on the change history. The right area provides additional, context-sensitive features including search, similar instances, and rating options.

Figure 3: A knowledge base for requirements of an e-government use case managed in the *OntoWiki* tool

By using this environment, stakeholders are enabled to create and interlink knowledge concepts, requirements, goals, and scenarios. Furthermore, they can vote, discuss, and comment pieces of content. The integrated versioning and change history features facilitate traceability when collaboratively working on a knowledge base. In order to enrich the requirements with semantics, externally available domain knowledge can easily be referenced, too. One source for structured knowledge might be domain ontologies that are extracted from *Wikipedia*[3] – as it is subject of investigation in the SoftWiki-related *DBpedia* project [AL07].

Although the user interface is designed according to the wiki principle, early user studies showed that it is not immediately understandable and too generic in some regards. Therefore, we are currently working on a redesign of the interface and general appearance of the application to ease interaction and better meet the users' needs in requirements elicitation. Not all users can, however, be reached with a wiki-based approach, no matter how well the user interface is designed. This is particularly true for end-users that are not actively participating in requirements engineering for the most part. Nonetheless, as end-users normally are directly affected by the implementation of the requirements, their feedback can be crucial for the success of the project [Ku05, HL01]. This is one reason why we are currently extending the wiki-based environment in various directions in order to provide additional support for specific requirements elicitation activities and stakeholder groups.

4 Extensions of the Wiki-based Approach

Due to the continuous application of semantic web technology, the wiki-based approach can be seamlessly extended by decentralized communication and participation channels. The left screen in Figure 4 shows a part of a tool called *Softfox* that can be easily integrated into the web browsers of users. It has been realized according to the paradigm of social tagging [Ma06] and allows gathering user feedback along with information regarding the usage context. Users are encouraged to add freely chosen tags to their input that can later assist in analyzing the user feedback and developing a conceptualization of the requirements domain.

Another extension called *SW-Analytics* supports the integration of knowledge that has been elicited decentrally (see right screen in Figure 4). It facilitates multidimensional visual exploration and preselection of relevant contents in accordance with concepts that have been defined in the SWORE ontology. The current implementation considers geospatial and tagging data enabling developers to browse stakeholder contributions based on the location where they have been stated or the tags that have been added. This way, requirements can be explored with respect to a specific purpose – for instance, a developer may want to view only those requirements that have been stated at a specific company site.

[3] http://wikipedia.org

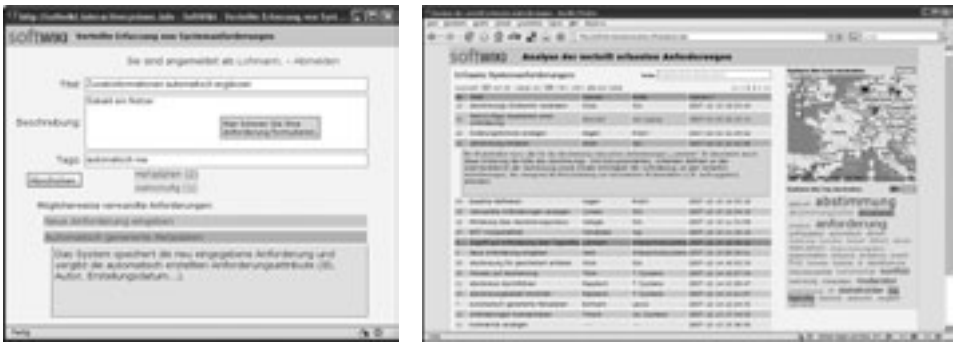


Figure 4: Extensions for decentralized participation and visual exploration

In order to facilitate semantic interoperability with other tools, the collaboratively developed knowledge base can be exported in RDF-format according to the schema of the SWORE ontology. Alternatively, parts of the knowledge base can be accessed via a SPARQL endpoint. Moreover, we are currently working on an adapter that enables export in RIF[4]-format to integrate the SoftWiki approach with established requirements and project management tools (in our case: *IRqA*[5]). However, we do not intend to fully embed the wiki-based environment itself, but rather try to link the collaboratively developed knowledge base with other development activities (see above).

Further extensions that are currently under development include the integration of semi-automatic knowledge extraction and classification mechanisms as well as additional visualizations such as network graphs of the knowledge and requirements structure.

5 Conclusions and Further Work

First experiences with use cases of the project indicate that an agile, wiki-based methodology can provide valuable assistance for distributed requirements engineering. Related work (e.g. [De07, Lo06]) supports this general conclusion. Furthermore, the application of a semantic wiki in combination with an ontological schema seems to foster terminological accuracy and shared understanding among the participants. In addition, it allows for semantic interoperability and automated utilization of the collaboratively developed knowledge base. In order to improve development assistance and meet the needs of various stakeholder groups, we addressed some extensions of the wiki-based approach that we are currently working on.

Furthermore, we consider integrating the SoftWiki approach with on-demand software development. As we experienced with COfundOS[6], a web platform for on-demand development and funding of open source software, sophisticated and usable means to

[4] Requirements Interchange Format: http://www.automotive-his.de/rif/doku.php
[5] http://www.qa-systems.de/html/deutsch/produkte/irqa
[6] http://cofundos.org

handle requirements are crucial in order to make this type of software development possible. Semantic wiki based management of requirements knowledge seems promising in this regard.

Generally, the design of any solution for web based collaboration of stakeholders needs to be easy to use in order to gain high participation. At the same time, it has to be powerful enough to enable structured access as well as efficient analysis and post-processing of requirements knowledge. Furthermore, the solution should be as generic as possible on the one hand and provide sufficient support on the other. Balancing these conflicting demands is a key issue to any approach that aims to assist in distributed requirements engineering.

Literaturverzeichnis

[ADR06] Auer, S.; Dietzold, S.; Riechert, T.: OntoWiki – A Tool for Social, Semantic Collaboration. In: Proceedings of the 5th International Semantic Web Conference, 2006; pp. 736-749.

[AL07] Auer, S.; Lehmann, J.: What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content. In: Proceedings of the 4th European Semantic Web Conference, 2007; pp. 503-517.

[De07] Decker, B.; Ras, E.; Rech, J.; Jaubert, P.; Rieth, M.: Wiki-Based Stakeholder Participation in Requirements Engineering. In: IEEE Software, 24(2), 2007; pp. 28-35.

[HL01] Hofmann, H. F.; Lehner, F.: Requirements Engineering as a Success Factor in Software Projects. In: IEEE Software, 18(4), 2001; pp. 58-66.

[KT05] Komi-Sirviö, S.; Tihinen, M.: Lessons Learned by Participants of Distributed Software Development. In: Knowledge and Process Management, 12(2), 2005; pp. 108-122.

[Ku05] Kujala, S.; Kauppinen, M.; Lehtola, L.; Kojo, T.: The Role of User Involvement in Requirements Quality and Project Success. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, 2005; pp. 75-84.

[LC01] Leuf, B.; Cunningham, W.: The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley, Reading, 2001.

[Lo06] Louridas, P.: Using Wikis in Software Development. In: IEEE Software, 23(3), 2006; S. 88-91.

[Ma06] Marlow, C.; Naaman, M.; Boyd, D.; Davis, M.: HT06, Tagging Paper, Taxonomy, Flickr, Academic Article, To Read. In: Proceedings of the 17th Conference on Hypertext and Hypermedia, 2006; pp. 31-40.

[RL07] Riechert, T.; Lohmann, S.: Mapping Cognitive Models to Social Semantic Spaces – Collaborative Development of Project Ontologies. In: The Social Semantic Web 2007 – Proceedings of the 1st Conference on Social Semantic Web, 2007; pp. 91-98.

[Ri07] Riechert, T.; Lauenroth, K.; Lehmann, J.; Auer, S.: Towards Semantic based Requirements Engineering. In: Proceedings of the 7th International Conference on Knowledge Management, 2007.

Making Wikis productive as the glue code of project information (Project WAVES)

Tim Romberg, Peter Szulman {romberg|szulman}@fzi.de

FZI Research Centre for Information Technologies at the University of Karlsruhe

Abstract. Today's software engineering projects require a knowledge management infrastructure that allows integration of existing information sources (e.g. issue trackers) and the exchange across organizational boundaries. In Project WAVES, we implement a solution based on a semantic, virtual knowledge base and a Rich Wiki Client.

1 Trends and Knowledge Management scenarios in software engineering today

Software engineering today is faced with the following trends:

- A greater proportion of effort is spent on *maintaining, refactoring and reintegrating* existing software, rather than writing it from scratch. This is because hardware platforms have been more stable and scalable in recent years than in the earlier computing history, when the entire application stack had to be created from scratch periodically.
- Typical projects can involve *several programming languages and frameworks*. It is not uncommon, for example, to find an application server component written in Java accessing a library written in C++; the Java component in turn serving both a Web interface written in PHP with some Flash elements and a Microsoft .NET Rich Client. Common Internet standards such as SOAP or XML make it possible to plug together best-of-breed frameworks for each task, rather than having to choose one.
- Projects are *distributed both geographically and over several organizations*. A full offshoring of all programming activities to a low-cost country is only one possibility of many.
- The *growing library of Open Source software* provides opportunities to jump-start development on new products by integrating or extending existing code, or to reach a large audience of existing users with add-on products and services.

This leads to Knowledge Management scenarios which cross organizational borders and require integration of existing knowledge, such as:

- A small interface design consultancy is hired, together with a number of other specialized service providers, into a software development project at a large telecommunications company. The question is how to set up a knowledge repository for this project where partners are provided selective access to each other's existing knowledge (such as a style guide), and it is ensured that any contributions by a

contractor's employees is still accessible to at least this contractor after the project ends.

- A company (e.g. Apple) publishes a software component (e.g. Safari WebKit) as Open Source. While the Open Source project has a public site providing version management, issue tracking, continuous integration, etc., the company still needs to manage private information with respect to this component (e.g. issues which would reveal coming features of its competitive offerings). The relationships of this information with the public information need to be kept in sync.

2 The role of Wikis

It is a frequent point of contention how *Knowledge* Management is different from *Information* Management, where the one starts and the other ends. No matter where one draws the line, the members of a software engineering team (encompassing programmers and testers as well as designers, business analysts and customer contacts), clearly need access to both knowledge and information, for example:

- the concepts and methods that define their respective qualification (such as the concepts of object-oriented programming or lambda calculus for a programmer, the concept of perspective for a graphic designer);
- knowing how to use the specific tools and raw material in their job (programming languages, libraries and environments, graphics software etc.);
- background knowledge of the problem domain their activities address (e.g. new legislation that will have to be implemented by the software product, users' background);
- specific infrastructure and processes in their work environment (how to log onto the staging server, whom to notify when changes are made);
- and finally, knowledge concerning the very project at hand – what the responsibility of certain modules are, why certain design decisions were made, which limitations and workarounds exist etc.

Today, mature tools and integrated Application Lifecycle Management (ALM) suites exist for managing standard processes and most aspects of the current project information. In many of the other areas, wherever a team needs to capture evolving knowledge, Wikis have established themselves as a versatile tool for managing knowledge whose structure is not well defined beforehand (as opposed to project information that can be modelled in databases). In fact, the first Wiki (Ward Cunningham's WikiWiki) was created to share software engineering patterns – an example for a semi-structured type of knowledge.

3 Integration, sharing and distribution requirements

However, today's Wiki engines constitute an island of their own with respect to existing information sources and the outside world. The goal of the Waves project is to address the mentioned challenges of Knowledge Management across organizational borders and integrating existing information sources. While similar challenges exist in many problem domains, we consider Software Engineering to be especially relevant, since it involves electronic artefacts which to a certain degree are vehicles of knowledge themselves, and since the dependencies between projects across organizations are long-term and require ongoing collaboration.

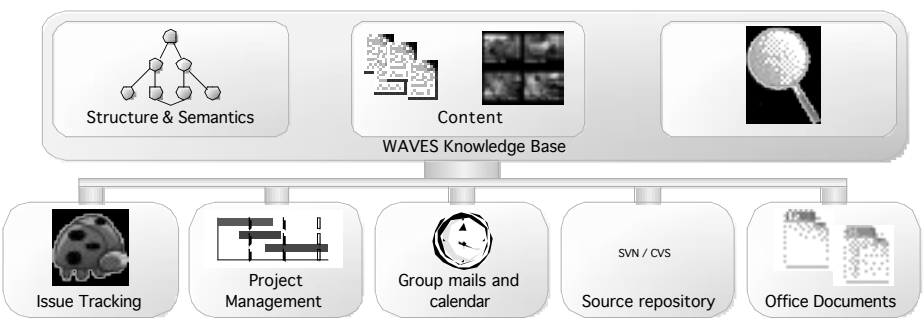
In this paper, we want to especially address the integration, sharing and distribution requirements, such as:

- being able to easily reference any existing information item (such as a bug or a release) when authoring Wiki pages;
- annotating any existing information item (e.g. using a Wiki page);
- integrating different visibility spaces (public, company-wide, team-wide, personal), by
  - being able to capture private (less visible) comments on public (more visible) knowledge resources;
  - being able to easily publish currently private (less visible) knowledge into a public (more visible) space while keeping a maximum of outgoing and incoming references intact.

4 Solution architecture

Our solution architecture consists of an integrated virtual knowledge base which allows querying knowledge (integrated search) and authoring knowledge (Rich Wiki client).

4.1 The integrated virtual knowledge base



Waves' virtual knowledge base integrates the various existing information sources, as depicted in the diagram, through a flexible adapter interface. The sources then become accessible through the client interface of the knowledge base (so-called SWECR – semantic web content repository, [Vö07]). Authored knowledge (Wiki pages including ad-hoc form data) is stored directly in the knowledge base. The knowledge base is implemented by combining a semantic metadata store (currently Sesame) with a binary store (for content) and a full-text index (Lucene). Both replication and virtual access is supported: Replication is usually chosen for data which is highly relevant for search criteria, low volume, and stable. Virtual access is preferred for seldom accessed, high volume, and volatile data.

4.2 Rich Wiki client

Waves provides a Rich Wiki client based on Java Swing. This enables a user-friendly, productive WYSIWYG interface. The client automatically analyzes the currently entered text to suggest hyperlinks to existing resources and Wiki pages (autocomplete). The user can also explicitly ask for link suggestions in a given context and analyze text pasted into the Wiki page from another application.

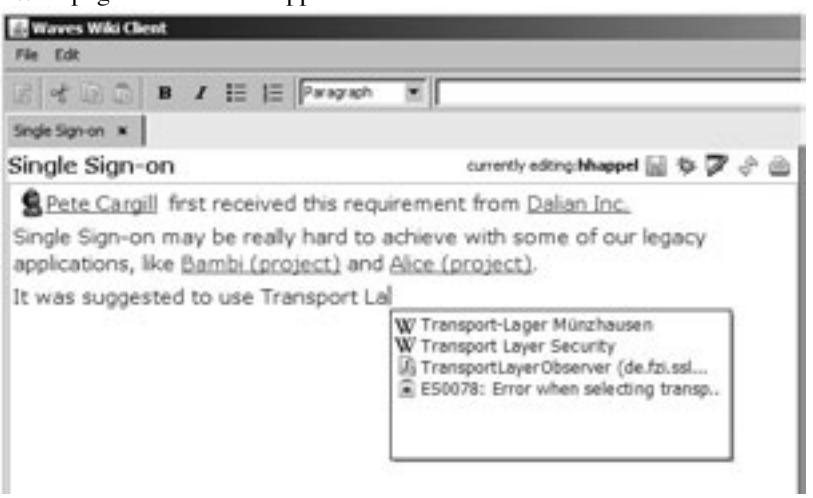


Figure 1. Automatic link suggestion in the Rich Wiki client

4.3 Sharing across organizational spaces

Sharing across organizational spaces involves the physical distribution of content and metadata on the one hand, and their conceptual organization in different visibility spaces.

4.3.1 Physical distribution

The SWECR interface (client interface of the virtual knowledge base) implements concepts borrowed from distributed relational databases, and thereby allows chaining together virtual knowledge bases and stream query responses through them, to form a

comprehensive virtual knowledge base as seen by the user. For writing (transactions), there is always one authoritative source. For reading, local server instances can replicate more remote instances to provide more throughput and reliability. A replica of any remote content that has been locally annotated is also kept for future reference, since the external resource may become unavailable. With this distribution model we hope to solve the Knowledge Management scenarios in our introduction: When an organization or individual joins a new project, they can explicitly define certain areas within their knowledge base which are allowed to be access and replicated by their partners.

4.3.2 Conceptual organization

Content and metadata is organized into discrete visibility spaces. Larger spaces are created by the administrators of a server instance. But there is also the possibility to create ad-hoc spaces for small groups by users themselves. When a piece of knowledge is published, it is actually transferred into the more visible space (which may involve a physical transfer to a different server instance). The transfer process then involves updating the incoming and outgoing references, and issuing warnings about any inconsistencies arising from this process.

5 Evaluation plans

The project takes an iterative approach, with major releases about every 6 months which are evaluated by 4 industry partners. The first release concentrated on the integration of the most important information sources (about a dozen applications in total, including several issue trackers and "legacy" Wikis, as well as file servers, code repositories etc.), and their joint querying. Evaluation of the second release, which includes the Wiki, is currently underway.

6 Related work

One of the earliest Wiki engines addressing specific software engineering activities and structured information types was SnipSnap, developed at Fraunhofer FIRST.[1] For example, SnipSnap allows collaborative editing of UML diagrams within a Wiki page using special syntax. The RISE project and the subsequent SOP project at Fraunhofer IESE have elicited the potential of combining Wikis and ontologies for the purposes of distributed requirements engineering ([DRR05], [DRR07]): Using the same Wiki principles, users can not only collect requirements, but also redefine the concepts and relationships used in the requirements model. This flexible and powerful model is especially appropriate when diverse stakeholders are involved. The SOP project is implemented on top of the fairly popular Semantic Mediawiki extension ([KVV06]). Structured information is freely mixed with free-form text.

[1] http://snipsnap.org

The use of collaborative ontologies for requirements engineering is also the focus of the OntoWiki ([ADR06]) and SoftWiki ([AF06]) projects at the University of Leipzig, with a stronger accent on structured information types and browsing and editing of RDF data. Similarly to the Waves project, SoftWiki tries to address usability and productivity issues with an intuitive interface, and provides Social Networking functions such as ratings and comments.

The Waves project is conceptually compatible and complementary with these efforts, as it shares the vision of a semantic collaborative knowledge base and its access through Wiki principles. However, it focuses more on the integration of existing information sources into this knowledge base through a service-oriented architecture, and therefore its Wiki serves as a complement rather than an alternative to commercial, structured requirements engineering tools and ALM suites. It also takes a more radical, rich client approach with respect to usability and uniquely addresses challenges of knowledge sharing across organizational spaces.

7 Conclusion

The proposed technology enables the scenarios of integrating various existing information sources, and of sharing knowledge across organizational boundaries. At the current implementation stage, it is possible to search through a great variety of relevant sources in Software Engineering, and author knowledge using the described Rich Wiki Client. The distribution architecture will be implemented in future releases.

References

[ADR06] Auer, S.; Dietzold, S.; Riechert, T.: OntoWiki – a tool for social, semantic collaboration. In Cruz et al. (Eds.): ISWC 2006. LNCS 4273 pp. 935--942. Springer

[AF06] Auer, S.; Fähnrich, K.-P.: SoftWiki – Agiles Requirements-Engineering für Softwareprojekte mit einer großen Anzahl verteilter Stakeholder. Statuskonferenz Forschungsoffensive „Software Engineering 2006“, 26.-28. Juni 2006, Leipzig. http://www.informatik.uni-leipzig.de/~auer/publication/softwiki.pdf (7.2.2008). Project web page: http://softwiki.de

[DRR05] Decker, B.; Ras, E.; Rech, J.; Klein, B.; Hoecht, C.: Self-organized reuse of software engineering knowledge supported by semantic wikis. In: Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE), ISWC 2005

[DRR07] Decker, B.; Ras, E.; Rech, J.; Jaubert, P.; Rieth, M.: Wiki-based stakeholder participation in requirements engineering. IEEE Software March/April 2007. SOP project web page: http://www.sop-world.org

[KVV06] Krötzsch, M.; Vrandecic, D.; Völkel, M.: Semantic MediaWiki. In Cruz et al. (Eds.): ISWC 2006. LNCS 4273 pp. 935--942. Springer. http://korrekt.org/papers/KroetzschVrandecicVoelkel_ISWC2006.pdf (7.2.2008). Project web page: http://semantic-mediawiki.org

[Vö07] Völkel, M.: A semantic web content model and repository, Proceedings of the 3rd International Conference on Semantic Technologies, SEP 2007.

Enabling Agile Offshoring with the Dual-Shore Model

Joachim Sauer
C1 WPS GmbH
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
js@c1-wps.de

Abstract: Conventional offshoring with requirements engineering onshore and implementation offshore is not very agile. Problems often arise in the transfer of requirements and knowledge and in the quality of the developed system. Agile offshoring may be a solution for these problems. But its application involves risks. In this paper common challenges of collaboration in agile offshoring projects are presented. Based on these and reflections on dimensions of distribution and cultural differences, three cooperation models for offshoring are analyzed from a knowledge sharing perspective. Thereby we show how dual-shore offshoring with onshore and offshore development offers a stable organizational foundation for agile offshoring.

1 Characteristics and challenges of agile offshoring

The offshoring of application development is an important part of global software development. But many offshoring projects suffer from problems resulting from a reduced communication bandwidth and not very agile development with long feedback cycles. This ultimately leads to inappropriate applications and failed projects [CT05].

The application of agile methods in offshoring projects promises to improve this situation. Agile offshoring (AO) means the application of agile methods and practices in offshore software development projects. Agile offshoring thus is a special case of agile global software development (AGSD).

Agile methods strive for improving flexibility and quality. Requirements are not considered to be carved in stone but may change and lead to adjustments in the developed system. Open communication, incremental releases, frequent feedback and simple, clear processes are core values of most agile methods. Many advantages of agile methods are especially desirable in distributed projects, e. g. early discovery of shortcomings and defects, intense involvement of the later users and better insight into the project progress. Well-known approaches for agile offshoring include Distributed Extreme Programming (DXP) [KJCL02] and Distributed Scrum [SVBP07]. They carry over the fundamentals of frequently used agile methods to distributed projects.

Unfortunately, real agility seems hard to realize in offshoring projects. A main problem lies in the distribution and diversity of the project members that hinders the adoption of interdisciplinary teams and informal communication. We recently carried out an investigation to assess further common challenges of agile offshoring. For this purpose, experience

reports that were published on major agile conferences were analyzed. The depicted challenges were categorized using open coding – a practice from grounded theory [Pan96]. First results of this study have been published in [Sau06].

Here we concentrate on the challenges that are most important for the collaboration of distributed teams. The following issues were identified:

Team spirit: Personal contact amongst project members is reduced in consequence of the geographical and temporal distribution. This makes it difficult to cultivate a common team spirit.

Common vision: The same reasons also hamper the establishment of a common vision – a set of shared values and principles that facilitates the cooperation.

Dealing with diverse cultures: Cultural differences show up on different levels: organizational structures and working habits, country-specific characteristics, and individual behaviors. They can lead to misunderstandings and frustration if they are not dealt with in proper form.

Language problems: Another frequent source of misunderstandings is the language. Although all participants of the experience reports' projects spoke English as common language, lacking language skills, different pronunciation, missing body language and different meanings of some words were problematic.

Informal communication: Communication during coffee breaks or informal meetings is important for a pleasant atmosphere and mutual friendship and trust. This is hard to accomplish in distributed teams.

Cooperation and coordination: All experience reports name cooperation and coordination of activities more or less as main problem areas of distributed teamwork. Issues that could be resolved easily in a colocated environment can take much longer in global software development.

Knowledge transfer: Knowledge distribution is critical in global teams. Common knowledge has to be explicitly build up and maintained. This can be challenging as fluctuation of employees is much higher in offshoring countries. The process of introducing changes and news should be planned carefully.

These issues should be addressed in research and practice to improve the outcome of agile offshoring projects.

2 Our research objective

Research of communication challenges in distributed projects concentrates on two areas. The literature on CSCW (computer supported cooperative work) deals with different

communication needs and the effects of distance [OO00]. Another important area of research are tools for supporting distributed teams. These can be grouped in the categories communication (e. g. video conference systems, e-mail), common information spaces (e. g. wikis), workflow-management (workflow management systems) and workgroup-computing (e. g. calendars, group editors, meeting support systems) [TSMB95].

In this paper we concentrate on organizational structures of distributed teams. Three different cooperation models are compared in search for better communication in agile offshoring projects. "A **cooperation model** is a usage model that describes and regulates a cooperative situation either explicitly or implicitly." [Zül04, p. 342] It mainly determines the distribution of the project members to the various project sites. The analyzed models are direct and indirect offshoring as conventional cooperation models for offshoring and dual-shore offshoring as agile solution for some of their shortcomings.

The discussion is based on the identified challenges of agile offshoring, three case studies of real-world projects (one using action research, two interview-based) [KS07] and reflections on dimensions of distribution and cultural differences in offshoring.

We distinguish between six different dimensions of distribution that were derived from the literature ([Gum06], [ECWP03], [AP99]). **Geographic distribution** is the most discussed one that separates offshoring from other forms of cooperative software development. **Temporal distribution** is typically caused by separated time zones, but also by different work rhythms or part-time work. Outsourcing leads to a strong **organizational distribution** between client and service provider. Dimensions not specific to offshoring are **distribution among stakeholder groups** (team members have different roles and functions, education and knowledge), **project distribution** (the project is split into several subprojects that have to be synchronized), and **personal distribution** (team members working on several projects in parallel).

The other focus of this analysis are **cultural differences**. As offshoring projects are characterized by teams working on different continents, team members have their roots in different cultural backgrounds. Cultural differences can hinder the knowledge transfer between teams, can lead to misunderstandings and problems and prevent the building of trust and a common team spirit [KSW04]. They are not limited to national cultures. In software development projects, people from many backgrounds have to work together. They act in different roles with different believes and goals: mainly the customer (typically split in business and IT department), analysts, programmers, testers, and project managers.

3 Direct and indirect offshoring

The first-generation offshoring model is known as **direct offshoring**. The customer directly interacts with the offshore service provider. This model typically uses a strict separation of tasks: the customer determines the requirements, the service provider does the implementation. This model's roles and communication channels are depicted in figure 1.

The customer's IT and business departments have to agree on the requirements and the specification (1). The project manager (PM) and tester (T) typically belong to the IT de-

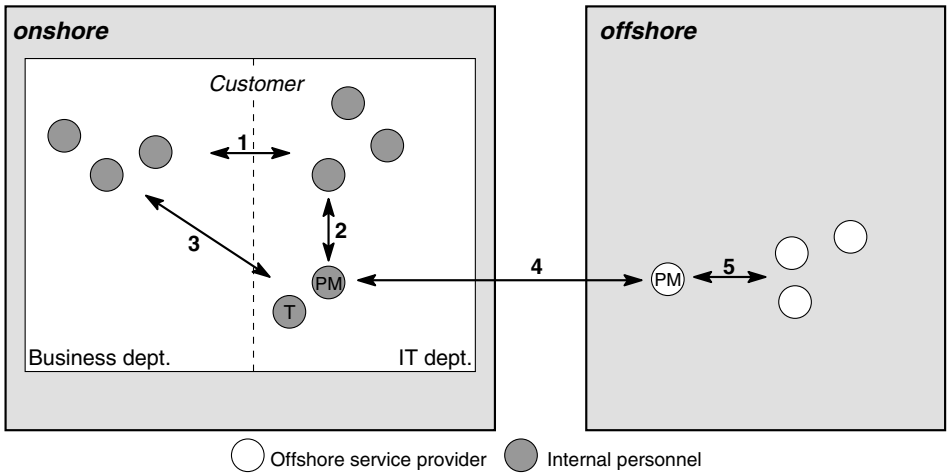


Figure 1: Direct offshoring

partment. They discuss organizational and technological issues with the IT department (2) and business issues with the business department (3). Problems due to different stakeholder groups and work cultures may arise. The most burdened communication channel is the one between the project managers from the customer and the offshore service provider (4). It serves as the only onshore-offshore bridge as well as the interface between requirements and implementation. The project managers have to deal with several challenges in the areas of geographic, temporal and organizational distribution and cultural differences. They have to manage the knowledge transfer progress with few context information and possible awareness difficulties. The software development process at the offshore location is more or less hidden from the customer (5).

The success of this model depends on the correct transfer of the requirements and domain knowledge to the offshore service provider. The geographical and temporal separation as well as cultural differences and language problems can hamper the cooperation between the customer and the offshore service provider severely. The customer has to overcome communication difficulties by himself. If problems arise, only contact persons from the offshore service provider are available.

Indirect offshoring has been developed to avoid problems of direct offshoring. In the following, only new communication channels and important changes in comparison with direct offshoring are described. See figure 2.

Instead of directly interacting with the offshore service provider, the customer employs a local external partner who has experience with software development in offshoring. Its personnel works at the onshore location with the customer's IT and business department (2, 3). The external project manager takes over the important part of communicating with the offshore service provider (5), reducing the need of direct interaction between the customer

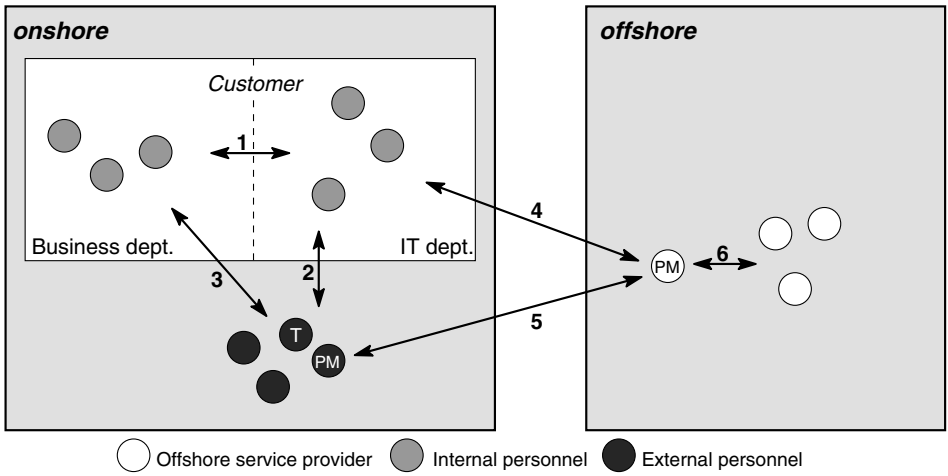


Figure 2: Indirect offshoring

and the offshore service provider (4). This set-up does not change the distribution and cultural challenges and possible overload of the onshore-offshore channel.

The success of this model depends mainly on management, language, cultural and overall skills of the external personnel. If they manage to effectively mediate between the customer and the offshore service provider, many communication problems that would otherwise affect the project can be resolved.

4 Dual-shore offshoring with onshore and offshore development

Dual-shore offshoring is based on the insight that the transfer of requirements and knowledge over considerable distances between people with different cultural backgrounds is very hard. With dual-shore offshoring, employees of the offshore service provider alternate between working offshore and onshore. Optionally, onshore personnel works for a while offshore. This exchange is not limited to developers. It also makes sense to let onshore domain experts and business analysts pass on their knowledge directly at the offshore location.

In our understanding of dual-shore offshoring, development teams work onshore and offshore, coordinating their work and shifting tasks dynamically. Onshore and offshore employees work in mixed teams in close cooperation, directly sharing business and technological knowledge. See figure 3.

With this model, many issues can be resolved without involving the other shore (7, 8). The important onshore-offshore channel is relieved as communication takes place between or with the support of members of the same company, eliminating organizational and cul-

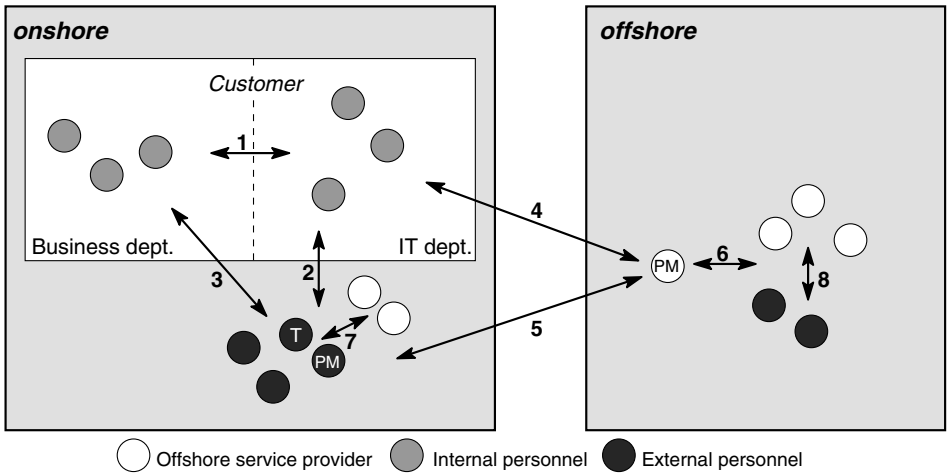


Figure 3: Dual-shore offshoring with onshore and offshore development

tural differences (5). Furthermore, business requirements can be translated onshore into tasks easier understood by developers. Communication can take place from developer to developer, focusing on technological issues.

By now, several companies apply the dual-shore model in agile offshoring projects, including Zensar Technologies [Win05], NIIT Technologies [Ste05] and Valtech (Duo-Shore, TwoShore) [MS03].

5 Comparison and conclusion

The direct and indirect offshoring cooperation models should only be employed in small or technology-focused projects. In these projects, the requirements can be unambiguously specified and they as well as the environments remain stable. The offshore service provider should be chosen carefully. Under these circumstances, these conventional forms of offshoring offer the potential of considerable cost reduction due to extensive use of offshore resources. Indirect offshoring is especially advisable if the customer has no prior experience with offshoring. The risk can be further reduced at the expense of higher costs if the external partner is willing to take on part of the responsibility.

Dual-shore offshoring is well suited for agile offshoring projects for the development of complex application software. This comes mainly due to two factors:

1. The relief of the onshore-offshore channel through mixed teams that cooperate directly.

2. Onshore and offshore development that overcomes the separation of requirements engineering and implementation.

Our empirical, qualitative studies have shown that dual-shore offshoring offers many advantages in the areas of communication, coordination and cooperation: easier team building and mutual understanding, better requirements and knowledge transfer, less misunderstandings, more frequent feedback and better controllability for the project manager. On the other hand, dual-shore offshoring with onshore and offshore development imposes the need of global software development with distributed teams. This can introduce new risks and increase the efforts needed for personnel, coordination, travels and working abroad. The advantages and disadvantages of the cooperation models have to be carefully considered for every single project with its specific setting and requirements.

While cooperation models offer instructions for the basic set-up of the project and the cooperation between the teams, they have to be complemented by a detailed concept for everyday communication and work division. Future research work in this area should focus on finding such a suitable concept for dual-shore offshoring to improve agility and outcome of offshoring projects.

References

[AP99] Josef Altmann and Gustav Pomberger. Kooperative Softwareentwicklung: Konzepte, Modell und Werkzeuge. In August-Wilhelm Scheer and Markus Nüttgens, editors, *Electronic Business Engineering: 4. Internationale Tagung Wirtschaftsinformatik 1999*, pages 643–664. Physica-Verlag, Heidelberg, 1999.

[CT05] Erran Carmel and Paul Tjia. *Offshoring Information Technology – Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, 2005.

[ECWP03] J. A. Espinosa, J. N. Cummings, J. M. Wilson, and B. M. Pearce. Team boundary issues across multiple global firms. *Journal of Management Information Systems*, 19(4):157–190, 2003.

[Gum06] Dorina C. Gumm. Distribution Dimensions in Software Development Projects: A Taxonomy. *IEEE Software*, 23(5):45–51, 2006.

[KJCL02] Michael Kircher, Prashant Jain, Angelo Corsaro, and David L. Levine. Distributed Extreme Programming. In Michele Marchesi, Giancarlo Succi, Don Wells, Laurie Williams, and James D. Wells, editors, *Extreme Programming Perspectives*, chapter 44, pages 553–568. Pearson Education, 2002.

[KS07] Andreas Kornstädt and Joachim Sauer. Mastering Dual-Shore Development – The Tools & Materials Approach Adapted to Agile Offshoring. In Bertrand Meyer and Mathai Joseph, editors, *Software Engineering Approaches for Offshore and Outsourced Development, First International Conference, SEAFOOD 2007, Zurich, Switzerland, February 5–6, 2007. Revised Papers*, volume 4716 of *Lecture Notes in Computer Science*, pages 83–95. Springer, 2007.

[KSW04] S. Krishna, Sundeep Sahay, and Geoff Walsham. Managing cross-cultural issues in global software outsourcing. *Commun. ACM*, 47(4):62–66, 2004.

[MS03] Brad Murphy and Tim Snyder. TwoShore – The Valtech Agile Global Delivery Model, Part 1 – for the Project / Program Director. Whitepaper, Valtech Technologies, 2003.

[OO00] G. M. Olson and J. S. Olson. Distance Matters. *Human-Computer Interaction*, 15(2&3):139–178, 2000.

[Pan96] Naresh R. Pandit. The Creation of Theory: A Recent Application of the Grounded Theory Method. *The Qualitative Report*, 2(4), December 1996.

[Sau06] Joachim Sauer. Agile Practices in Offshore Outsourcing – An Analysis of Published Experiences. In *Proceedings of the 29th Information Systems Research Seminar in Scandinavia, IRIS 29 – Paradigms Politics Paradoxes, August12–15, Helsingoer, Denmark*, 2006.

[Ste05] Rolf Stephan. Kommunkation und Wissenstransfer – Schlüsselfaktoren für erfolgreiche Offshore Projekte. In Heinz-Josef Hermes and Gerd Schwarz, editors, *Outsourcing. Chancen und Risiken, Erfolgsfaktoren, rechtssichere Umsetzung*, chapter 11, pages 213–234. Haufe, 2005.

[SVBP07] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed Scrum: Agile Project Management with Outsourced Development Teams. *hicss*, 0:274a, 2007.

[TSMB95] Stephanie Teufel, Christian Sauter, Thomas Mühlherr, and Kurt Bauknecht. *Computerunterstützung für die Gruppenarbeit*. Addison-Wesley, Bonn, 1995.

[Win05] Martin Winkler. Mit dem Dual Shore Delivery Model Sprach- und Kulturbarrieren bei IT-Offshoreprojekten überwinden. In Reiner Clement, Andreas Gadatsch, Martin Kütz, and Jens Juszczak, editors, *IT-Controlling in Forschung und Praxis – Tagungsband zur 2. Fachtagung IT-Controlling, Sankt Augustin, 21. und 22.02.2005*, number 13, pages 127–136, 2005.

[Zül04] Heinz Züllighoven. *Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach*. dpunkt.verlag. Copublication with Morgan-Kaufmann, 2004.

From Research to Practice: How Does TXT e-solutions Plan to Deploy Innovations in Sharing Development Knowledge

Alessandra Bagnato[1], Walid Maalej[2]

[1] TXT e-solutions
Salita San Barborino 23r Genoa, Italy, 16100
alessandra.bagnato@txt.it

[2] Technische Universität München
Bolzmannstr. 3, 85748 München, Germany
maalejw@in.tum.de

Abstract: The research project TEAM aims at developing lightweight knowledge sharing mechanisms, optimized for distributed software teams. This paper describes how the software development company TXT e-solutions plans to evaluate and deploy innovative results from this project. First, we introduce the as-is situation for collaboration and knowledge sharing inside the company. Then, we discuss expectations as well as evaluation environments for TEAM and similar research initiatives.

1 Introduction

Working in distributed teams rapidly gains increasing importance for professional software development [Ca99, MR05]. These teams face not only the emergence of communication bottlenecks due to far-distance collaboration, but also a growing need for extensive communication and knowledge sharing. For example, nowadays reuse happens at a higher level of abstraction and complexity than earlier, as the "construction by configuration" approach continues to emerge [S308]. Correct component integration, effective work with powerful frameworks or successful usage of design patterns, all require significant background and experience about used artifacts.

Moreover, the globalization of software development enables the formation of new alliances and more effective software creation value chains, resulting in shorter turn-around time for knowledge. New versions are released more frequent and the experience with old versions ages rapidly. The research project TEAM [TE08] aims at developing lightweight knowledge sharing mechanisms optimized for distributed teams to deal with such agile environments. Key enablers are semantic annotations of development artifacts, search and recommendation features as well as automatic capture and sharing of developers' contexts.

This paper discusses an evaluation plan for the TEAM research innovations. In a real life development environment inside the company TXT e-solutions [TXT08], a pilot project has been selected with teams located in three Italian cities. We introduce the as-is situation and discuss expectations from the TEAM system as well as the evaluation framework.

2 The Context

TXT e-solutions is a software development company, listed on the Italian Stock Exchange since July 2000. Its revenues in 2006 were € 56,3 million. TXT e-solutions is based in Milan with offices in Turin, Vicenza, Perugia, Bari, Roma and Genoa. The company owns subsidiaries in Paris, Lyon, London, Chemnitz, Halle, Haarlem, Barcelona and New York (Figure 1). Most development activities take place in the Italian offices. The company employs 500 persons, 250 of whom are software developers.

Figure 1: Distributed offices of TXT e-solutions

TXT e-solutions operates in three main areas:

- In Demand & Supply Chain Management: the company offers products and consulting services based on the TXTPERFORM suite. Target markets are fashion, retail, aerospace, defense and automotive.
- In Multi-channel Content Management: the company provides solutions based on the TXT Polymedia product suite that supports various media formats, e.g. TV and SMS. Products target the media, telecommunication, industry and finance markets.
- In Business Intelligence and Business Process Modeling: the company offers specialized solutions dedicated to the primary needs of industry and finance.

At TXT e-solutions, the software development process is often distributed since the competencies are distributed among several offices. For a distributed project, there is a unique manager and one technical team leader for every office involved in the project. Technical team leader coordinates a team of 2-10 developers.

2.1 The Pilot Project

The usefulness of the TEAM conceptual models and knowledge sharing approaches for TXT e-solutions will be evaluated in the framework of a pilot project, where three distributed teams will develop a software application. Each team includes four developers working in three different locations in Italy: Genoa, Milan Bari. Genoa team presents a high expertise in developing graphical user interfaces, whereas Bari team is highly skilled in security matters and algorithms. Most customers are based in Milan.

The pilot project aims at creating a graphical application to assist the selection of persons that should be included in a team. It will comprise the following three main components:

- *Matching Engine*: This core component finds employees' profile that fits best to the required skills. The engine will be developed in Milan and Bari.
- *Ontology Tree Displayer*: This component displays the employee's skill-tree, allowing the easy selection of different nodes. It will be developed in Milan and Genoa.
- *Self Skills Identifier*: This component manages the automatic selection of own skills. It takes input from Ontology Tree Displayer (selected nodes) and displays them to the user, allowing the selection of related skill levels. This component will be developed in the Milan and Bari with support from Genoa developers in graphical issues.

Table 1 introduces libraries already deployed in past projects and planed to be reused in the pilot project. Figure 2 summarizes the main project organization, the components to be developed and the libraries to be reused.

Table 1: Libraries to be reused in the pilot project

LIBRARY	**DESCRIPTION**
JFREECHART 1.0.6 [JF08]	*JFreeChart* supports the creation of several charts types such as pie charts (2D and 3D), bar charts, line charts, time series charts, high-low-open-close charts, Gantt charts etc. *JFreeChart* can be used in applications, applets, servlets and JSP.
Google Widgets/Gadgets [Wi08]	*Web Widget* is a portable chunk of code that can be installed and executed within any separate HTML-based web page by an end-user without requiring additional compilation. Google supplies its own developer SDK to create widgets for its platforms.

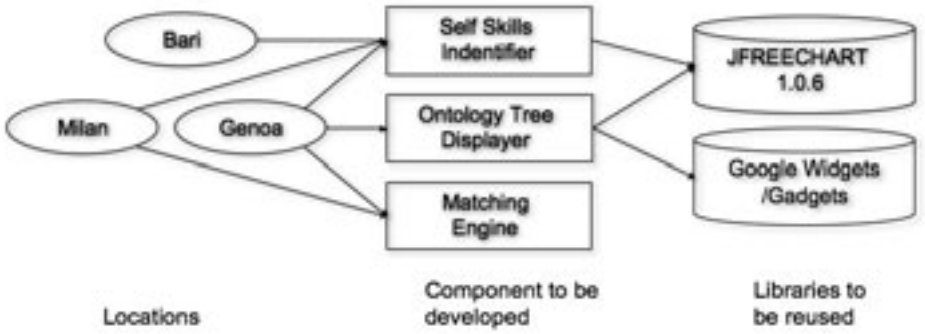


Figure 2: Organization, system architecture and libraries to be reused in the TEAM pilot project.

3 Current Practices of Collaboration and Knowledge Sharing

The various offices of TXT e-solutions make extensive use of a Virtual Private Network (VPN), enabling employees to share data, without realizing that the computer, with which they are working, is located in offices kilometers away. All employees including developers may view content shared in various company computers by means of shared folders. In distributed teams, developers tend to share folders on their own computers with the members involved in the project with temporary and non-temporary data. Source code and documentation such as installation manuals, user guides, system requirements, and component descriptions are kept and maintained centrally by means of SVN repositories.

In addition, TXT developers working in a distributed team have computers equipped with headphones to facilitate skype calls [SK08]. They use to have almost daily skype calls with team members from other offices, and exchange emails with more technical details. They also make use of the yugma tool [YU08] to share desktop content during the calls. Multi-users conferences via skype are seldom held due to the experienced connection difficulties. On a weekly basis, a phone conference is organized with 4-5 participants. However calls are not sufficient in order to communicate and discuss complex models that are often used in software development. Although, screen sharing helps communicating the discussion context, offline knowledge articulations are required to clarify and structure ideas. Bugzilla [BU08] is rarely used to internally report bugs. Developers often prefer mail exchange and direct phone calls for collaboration and organize their email folders as it better suits their needs.

Developed components are kept in a separate machine that is accessible from all offices via VPN or via SVN. Details on involved developers, the technical leader and the manager are always included together with the component, however in a non-systematic way. For instance, such information can be included in the source code or in a component description document. The phone number and the contact details of all the involved people are to be found in the intranet-based address book. If developers have specific question on reusing the component, the manager is contacted. In turn, the manager forwards the request to the technical leader or the most competent developer, who has the knowledge and expertise to answer the question.

If they need assistance, developers involved in the project send emails to a special address *dummyProject@dummyDomain*. A component guru replies to messages sent to this address. This workaround enables the separation of personal e-mails and e-mails related to the component, in this case *JFreeChart*. Documents describing special issues about the component, which were found by the guru, are not communicated to the team. Knowledge discovered about a particular component that should be reused is often kept on the mind of the component guru. This is due to the limited time of the guru and the reduction of information overload for developers. Experience showed that even written documents do not guarantee that all needed information is provided. The guru might be able to answer a question taking into account more than one concrete situation he experienced during development.

4 Knowledge Sharing Approach and Expected Benefits

TEAM project aims to develop an open-source software system, seamlessly integrated in a software development environment for enabling decentralized, personalized and context-aware knowledge sharing. This will be achieved through:

- Lightweight annotation of development artifacts as well as knowledge manipulation features directly integrated in the software development environment.
- Capturing of the user's behavior in the desktop environment and automatically triggering the knowledge manipulation.
- Analysis of interaction history and elicitation of a user profiles.
- Context-aware and ontology-based proximity search for relevant knowledge items.
- Semantic recommendation, through a proactive knowledge delivery depending on the actual working and personal context of the user.
- Ontology-based efficient structuring and persistent storage of acquired knowledge, as well as reasoning about its completeness and consistency.
- P2P-based decentralized communication.

By using the TEAM system TXT e-solutions strive for improving knowledge sharing among its distributed software development teams. The TEAM system should make development activities more efficient and foster the collaboration among the developers. Developers should get recommendations and suggestions from the TEAM system, which save them time from email communication with other developers or with components gurus.

Using TEAM, solutions to problems with potentially reusable library features should be shared only if other developers require this.

Known bugs as well as ways to solve them are often kept solely in the mind of the developers that discovered them. Usually, this knowledge is not made explicit anywhere. With TEAM it should be possible to keep track of developers that are expert in certain features or certain parts of the library. Recommendations should be similar to normal developers' brain. In situations where specific knowledge is required, developers remember: a) who was the person involved in a particular project; b) why he had that particular issue with the library to be reused; and c) how he solved this issue.

The TEAM system should gather information from existent repositories, in this case the mail archive. The TEAM system should help involved developers with recommendations without a need for consulting the guru. This will speed up the development process since the guru will not be a bottleneck any more. Developers are not blocked if the guru is, e.g. on holiday, ill or away for work.

Features similar to semantic recommendation or history analysis have not been tried in TXT software development environment yet. The developer's perception of this new means of assistance is crucial, since the acceptance of the new system might influence the whole development process. The acceptance of the TEAM system cannot be guaranteed due to the monitoring functionality. Thus, the implemented privacy protection mechanism as well as the perceived added value for the developers should be clearly convincing. For example, developers should have control over the monitoring functionality and just switch it off if desired.

The developers will use the TEAM system to facilitate knowledge exchange in reusing software artifacts. An example recommendation might be: 'this interface of the component is the one you need to solve your problem'. It is intended to better understand what are the minimum requirements that a pilot project should give to the TEAM system, which is crucial for its further rollout. Important functionality that can be only evaluated in such concrete situation includes: Does the TEAM system learn automatically? How much extra effort do developers require for running the TEAM system? What functionality and recommendation will be provided to development teams?

5 Evaluation Criteria for Lightweight Knowledge Sharing

Table 2 shows the evaluation criteria as well as techniques that will be used to evaluate the perceived usefulness of TEAM functionalities within the pilot project. Meeting the success indicators described below will mean that developers working with the TEAM system perform better than working without it.

Table 2: Evaluation criteria for deploying the TEAM system in the pilot project

Evaluation criteria	**Technique to be used for measuring the performance indicator**	**Target value**
Quality of knowledge models (i.e. TEAM Ontologies)	Adequacy of these models as perceived by developers in the pilot project.	> 75% of pilot project developers
	Assessment of ontology quality according to [Go04]: Consistency, Completeness, Conciseness, Expandability and Sensitiveness.	
Usefulness of elicited context	Perceived retrieval quality (controlled experiments with users, with and without context usage)	> 90% pilot project developers acceptance
Quality of semantic search	Efficiency of retrieval algorithm (runtime efficiency)	Answer time < 5 seconds
	Perceived usefulness of full search functionality (user questionnaires)	> 90% pilot project developers acceptance
Quality of semantic recommendation	Efficiency of recommendation algorithm (runtime efficiency)	Answer time < 8 seconds
	Perceived usefulness of recommendations (user questionnaires)	> 80% accepted recommendations
	Distraction factor (user questionnaires)	< 10% of notifications rejected by user
Usefulness of history analysis	Perceived retrieval quality (controlled experiments with users, with and without history usage)	> 80% user acceptance
Usability	User satisfaction with GUI concepts and design for annotation and knowledge manipulation	> 90% user acceptance

6 Conclusion

In this paper we described the as-is situation for collaboration and knowledge sharing in a distributed software development company. In particular, reuse-intensive project problems and needs with regard to current practices, primarily file sharing, were derived. The research project TEAM aims at coping with these problems using semantic search and recommendation technologies and context awareness, i.e. observing and analysing users' interactions. Based on a concrete evaluation framework, we described developers' as well as company's expectations from the TEAM approach of capturing and sharing component reuse knowledge, in a distributed setting. Next steps include conducting the pilot project and assessing the benefits as well as the consequences of the TEAM approach on the development activities.

Acknowledgement

We thank Maria Legal for her valuable comments on consistency and wording. This work was partly supported by the European Commission (IST-35111-TEAM).

References

[BU08] Bugzilla Homepage http://www.bugzilla.org/
[Ca99] Carmel, E. 1999, Global Software Teams: Collaborating Across Borders and Time
[Go04] A. Gómez-Pérez: Ontology Evaluation. In: R. Studer & S. Staab (eds.): Handbook on Ontologies, pp. 251-274. Springer-Verlag, 2004
[JF08] JFreeChart Homepage: http://sourceforge.net/project/showfiles.php?group_id=15494
[MR05] Minevich M. and F.-J. Richter, Global Outsourcing Report 2005, March 2005
[SK08] Skype Homepage http://www.skype.com/
[S308] The S3 initiative, A European Vision for Software, Services and System ftp://ftp.cordis.lu/pub/ist/docs/directorate_d/st-ds/visdoc.pdf
Zones, Prentice-Hall, Upper Saddle River, N.J., 1999
[TE08] TEAM Project Homepage http://www.team-project.eu
[TXT] TXT Group Homepage http://www.txtgroup.com/
[Wi08] Google Desktop API: http://desktop.google.com/dev/index.html
[YU08] A Skype collaboration tool http://www.yugma.com/

A Tool with which to Recommend Knowledge Sources in Virtual Communities

Juan Pablo Soto, Aurora Vizcaíno, Javier Portillo-Rodríguez, Mario Piattini

University of Castilla–La Mancha
Alarcos Research Group – Institute of Information Technologies & Systems
Department of Information Technologies & Systems – Escuela Superior de Informática
Ciudad Real, Spain
JuanPablo.Soto@inf-cr.uclm.es
Aurora.Vizcaino@uclm.es, Javier.Portillo@uclm.es, Mario.Piattini@uclm.es

Abstract: This paper describes a multi-agent architecture with which to recommend knowledge sources in virtual communities. The tool presented in this work may be particularly useful to software organizations for two main reasons: it can assist them in the management, generation, acquisition, exchange, protection, distribution, and utilization of knowledge generated during the development of their projects, and it can also help to increment the collaboration of their members by, for example, supporting the sharing and use of knowledge between them. The applicability of the tool is illustrated with a scenario in order to show its features.

1 Introduction

Knowledge Management (KM) provides methods and techniques that can help organizations to increment the collaboration of their members by, for example, supporting the sharing of knowledge between them. Documented examples of benefits that may be attained from managing knowledge effectively include: reduced time-to-market; reduced development costs; innovative uses of existing products; revolutionary product ideas; and reduced employee turnover [Sk03]. An essential ingredient of knowledge sharing in organizations is that of virtual communities, by which we mean groups of people whose members may or may not meet one another face to face and may exchange word and ideas through the use of computer networks [GBK04]. This concept has become more and more popular within the field of the KM where it is mainly used as a KM tool to support the externalization of knowledge, both for reuse and for the purpose of innovation [HW00]. The importance of the concept of a virtual community at an organizational level is parallel to the growth in the interest of management approaches such as organizational learning and knowledge management. Virtual communities are often used in software development groups since software development is a knowledge intensive work in which software engineers must collaborate with other members of the team, and share their knowledge and experience in order to complete their assignments.

We believe that by providing software development groups with tools that facilitate their members' collaboration and sharing of knowledge, the performance of these communities can be incremented. In order to do this, we have designed a recommendation tool with which to identify knowledge sources and provide useful knowledge to virtual community members. The remainder of this work is organized as follows. The next section defines the concept of the virtual community and its advantages in software organizations. In Section 3 the multi-agent architecture used to develop our tool is described. Section 4 illustrates the applicability of the tool. In Section 5 related works are outlined. Finally, in Section 6 conclusions are presented.

2 Virtual communities

Formal and informal networks play an increasing role in business life and are a fundamental part of learning and knowledge exchange. Whilst the social aspect of learning and knowledge sharing plays a key role in these types of networks, the need to network and share knowledge through a chain of interdependent organizations (globalization) means that members of such networks are often not located at the same site or even in the same country. This has led to the emergence of virtual communities through which to keep the members of these networks in touch even without face-to-face meetings. According to Hamman, the sociological term 'community' should be understood as meaning a group of people who share social interaction, and some common ties between themselves and other members of the group, and who share an area for at least some of the time [Ha01].

In contrast to the sociological definitions of 'community' in which place and physical presence are important aspects, it is clear that computer networks allow for communities that stretch well beyond the neighbourhood [WG99]. In summary then, virtual communities are seen as the intersection of social and technical systems [SS01], and, as is recognized by [MAI97], neither technology nor sociality can supplant the need for the other. On the other hand, the use of virtual communities is increasing day by day. Virtual community members may or may not meet one another face to face and may exchange words and ideas through the use of computers networks [GBK04]. Consequently, the association of community members and the enabling electronic medium constitute an interesting infrastructure that provides benefits by supporting interpersonal relationships and companionship, encouraging discussion and knowledge sharing, allowing for quick access to information and enabling collective action such as, for instance, software development [Bu01]. Our research is focused upon professionally-orientated communities (software organizations), which consist of company employees who communicate and share information in order to support their professional tasks.

Software development communities

Many changes have recently taken place in the software development process, and software development is, to a large extent, distributed in that developers and users are geographically dispersed. Multi-site development is a current matter of study and discussion, since global development is becoming a usual style of software production. Offshore and nearshore outsourcing are practices which are therefore increasing considerably. Moreover, when participants are distributed geographically new problems often arise. For example, communication and coordination are more difficult because of differences in culture, timetable, language, etc., [BF04]. Some areas of research exist which attempt to minimize the impact of these problems. One such area is that of communities. These groups in software organizations provide the necessary infrastructure through which to exchange information between software engineers. The adequate management of these communities provides the necessary information to respond to certain questions that must be answered, such as: What kinds of problems could be solved in the community? What is the knowledge involved in the activities performed by the software developing groups? How do they share that knowledge? and who are the expert employees in topics related to the community? For these reasons, we consider it important to implement tools which give support to virtual communities in software organizations with the goal of recommending knowledge sources (which may be documents or ever community members). Before describing the proposed tool in detail, we shall first describe the architecture used to develop it.

3 Multi-agent architecture

The goal of this work is to provide an infrastructure for KM in virtual communities. In order to carry this out, we first designed a multi-agent architecture in which software agents try to emulate the behaviour of the employees in communities. This architecture provides the capacity to:

- Assist employees in identifying trustworthy entities and provide the confidence necessary to foster the usage of information and knowledge of recommendation systems. In order to do this, we have designed a reputation formula based on real world social properties (expertise, previous experience, intuition and position) of trust in communities [SVP07]. The main goal of this formula (which will be explained in detail in Section 4) is to rate the level of confidence in an information source or in a provider of knowledge. This formula could also help companies to detect those employees with more knowledge about a topic (expert detection).
- Give artificial agents the ability to reason about the trustworthiness of the agents or of a knowledge source in virtual communities.
- Encourage knowledge exchange between the community members.

Taking these facts into account, we designed a multi-agent architecture which is composed of two levels (see Figure 1): the reactive and the deliberative-social level.

The reactive level is considered by other authors as a typical level that a multi-agent system must have [UHN98]. A deliberative level is often also considered as a typical level but a social level is not frequently considered in an explicit way, despite the fact that these systems (multi-agent systems) are composed of several individuals, interactions between them and plans constructed by them. The social level is only considered in those systems that try to simulate social behaviour or those that represent a more generic architecture prepared to represent this or other behaviour. In our case the deliberative and the social level are not separate levels because, after developing several prototypes, we realised that in our domain both are narrowly related.

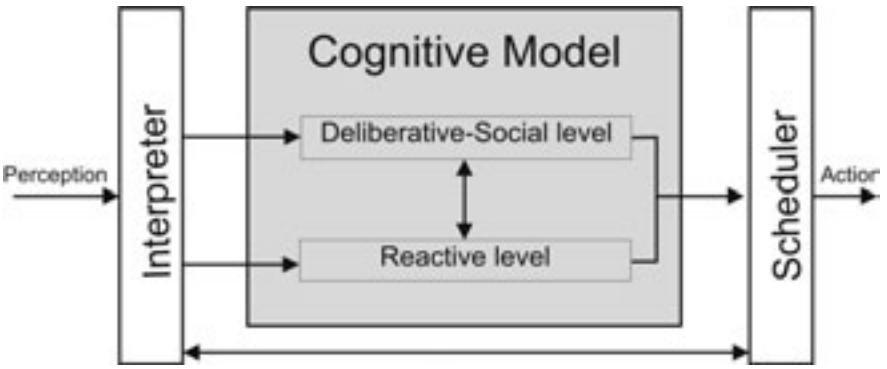


Figure 1: General architecture

Reactive level: In this level we represent the agent's capacity to perceive changes in its environment and to respond to these changes at the precise moment at which they happen. It is in this level when an agent will execute the request of another agent without any type of reasoning.

Deliberative-Social level: In this level the agent's behaviour is based on goals, that is, the agent has several defined goals and it attempts to achieve these goals by scheduling plans. Due to the fact that we are trying to represent human behaviour in a virtual community, it is necessary to bear in mind that this human behaviour must benefit the whole community. Therefore, the agent has to deliberate about its individual goals but it must also act by taking community goals and the community's profit into account. It is for this reason that we have considered both the social and deliberative aspects. The former attempts to achieve individual goals and the latter is more focused upon achieving social goals (community goals). In this level the agent obtains information about the environment and, by taking into account its interests and intuitions, it decides which plan is best suited to the achievement of its goals.

Two further important components of our architecture are an *Interpreter* and a *Scheduler*. The former is used to perceive the changes that take place and to decide which level must take the initiative depending on the event that the agent perceives. The scheduler indicates how the actions should be scheduled and executed. Due to space constrains the modules that form each of the levels have been omitted.

4 Recommendation tool

In order to test our proposal we have developed a prototype system into which people can introduce documents and where these documents can also be consulted by other people. The goal of this prototype is to allow software agents to help software development groups to discover information that may be useful to them, thus decreasing the overload of information that employees often have and strengthening the use of knowledge bases in software organizations. In addition, we attempt to avoid the situation of software engineers storing valueless information in the knowledge base.

One feature of this system is that when a person searches for knowledge in a community, and after having used the knowledge obtained, that person has to evaluate the knowledge in order to indicate whether:

- The knowledge was useful.
- How it was related to the topic of the search (for instance a lot, not too much, not at all).

To design this prototype we have implemented a *User Agent* and a *Manager Agent*. The former is used to represent each person that may consult or introduce knowledge in a knowledge base. Therefore, the *User Agent* can assume three types of behaviour or roles similar to the tasks that a person may carry out in a knowledge base. The User Agent plays one role or another depending upon whether the person that it represents carries out one of the following actions:

- The person contributes new knowledge to the communities in which s/he is registered.
- The person uses knowledge previously stored in the community.
- The person helps other users to achieve their goals, for instance by giving an evaluation of certain knowledge.

The second type of agent within a community is called the *Manager Agent* (which must manage and control its community). In the following paragraphs we shall explain how the prototype works.

Firstly, when a user wants to join a community in which no member knows anything about him/her, the reputation value assigned to the user in the new community is calculated on the basis of the reputation assigned from others communities in which the user is or was a member. For instance, a User Agent called *j*, will ask the managers of each community in which s/he was previously a member to consult each agent that knows him/her with the goal of calculating the average value of his/her reputation (R_{Aj}). This is calculated as:

$$R_{Aj} = (\sum_{j=1}^{n} R_{xj})/n$$

where *n* is the number agents who know j and $\mathbf{R}_{sj}$ is the value of reputation of j in the eyes of s. In the case of being known in several communities the average of the values $\mathbf{R}_{Aj}$ will be calculated. The User Agent j then presents this reputation value (similar to when a person presents his/her curriculum vitae when s/he wishes to join a company) to the Manager Agent of the community to which it is "applying". This reputation value permits an initial reputation value to be assigned, taking into account the previous experiences and relations with others agents, thus generating a flow and exchange of information between the agents. This mechanism is similar to the "word-of-mouth" propagation of information for a human [AH00].

In addition, $\mathbf{R}_{sj}$ value is computed as follows:

$$R_{sj} = w_e * E_j + w_p * P_j + w_i * I_{sj} + (\sum_{j=1}^{n} QC_{sj})/n$$

where R_{sj} denotes the reputation value that $agent_s$ has in $agent_j$ (each agent in the community has an opinion about each of the other agent members of the community with which it has interacted). E_j is the value of expertise which is calculated according to the degree of experience that a person has in a domain and which is given by the company. P_j is the value assigned to a person's position. This position is defined by the enterprise's organizational diagram. Therefore, a value that determines the hierarchic level within the organization can be assigned to each level of the diagram. I_{sj} denotes the intuition value that $agent_s$ has in $agent_j$ which is calculated by comparing each of the users' profiles.

In addition, previous experience should also be calculated. We suppose that when an agent *s* consults information from another agent *j*, the agent *s* should evaluate how useful this information is. This value is called QC_{sj} (Quality of *j*'s Contribution). To attain the average value of an agent's contribution, we calculate the sum of all the values assigned to their contributions and we divide it between their total. In the expression *n* represents the total number of evaluated contributions. Finally, w_e, w_p and w_i are weights with which the Reputation value can be adjusted to the needs of the organizations. For instance, if an enterprise considers that all its employees have the same category, then w_p= 0. The same could occur when the organization does not take its employees' intuitions or expertise into account. In this way, an agent can obtain a value related to the reputation of another agent and decide to what degree it is going to consider the information obtained from this agent.

Finally, when a person searches for a document relating to a topic his/her User Agent consults the Manager Agent about which documents are related to its search. Then, the Manager agent answers with a list of documents. The User Agent sorts this list according to the reputation value of the authors, which is to say that the contributions with the best reputations for this Agent are listed first. On the other hand, when the user does not know the contributor then the User Agent consults the Manager Agent about which members of the community know the contributors. Therefore, the User Agent can

consult the opinions that other agents have about these contributors, thus taking advantage of other agents' experience. To do this the Manager consults its interaction table and responds with a list of the members who know the User Agent This User Agent then contacts each of them. If nobody knows the contributors then the information is listed, taking their expertise and positions into account. In this way the User Agent can detect how worthy a document is and recommend it, thus saving employees' time, since they do not need to review all the documents related to a topic but only those considered most relevant by the members of the community or by the person him/herself according to previous experience with the document or its authors.

5 Related work

This research can be compared with other proposals that use agents and trust in knowledge exchange. For instance, in [AH00], the authors propose a model that allows agents to decide which agents' opinions they trust more and propose a protocol based on recommendations. This model is based on a reputation or word-of-mouth mechanism. The main problem with this approach is that every agent must keep rather complex data structures that represent a kind of global knowledge about the whole network.

Barber and Kim present a multi-agent belief revision algorithm based on belief networks [BK04]. In their model the agent is able to evaluate incoming information, to generate a consistent knowledge base, and to avoid fraudulent information from unreliable or deceptive information sources or agents. This work has a similar goal to ours. However, the means of attaining it are different. In Barber and Kim's case they define reputation as a probability measure, since the information source is assigned a reputation value of between 0 and 1. Moreover, every time a source sends knowledge, the source should indicate the certainty factor that the source has of that knowledge. In our case, the focus is very different since it is the receiver who evaluates the relevance of a piece of knowledge rather than the provider as in Barber and Kim's proposal.

6 Conclusions

The main contribution of this paper is the design of a recommendation tool with which to identify and recommend knowledge sources in virtual communities. The advantages of this contribution are that:

- It improves the management of knowledge in software organizations. This implies several advantages for organizations since it encourages groups of employees to exchange information. It is therefore expected that a greater flow of communication will exist between them which will consequently produce an increase in their knowledge.
- It gives information about the location of information, best practices and expertise.

All these situations provide organizations with a better control of their knowledge sources which will have more trustworthy knowledge and it is consequently expected that employees will feel more willing to use it.

Acknowledgements

This work is partially supported by the MECENAS (PBI06-0024), ESFINGE (TIN2006-15175-C05-05) and MELISA project (PAC08-0142-3315), Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, in Spain and CONACYT (México) under grant of the scholarship 206147 provided to the first author.

References

[AH00] Abdul-Rahman, A., Hailes, S.: Supporting Trust in Virtual Communities. Proceedings of the 33rd Hawaii International Conference on Systems Sciences (HICSS'00), Vol. 6, 2000.

[BK04] Barber, K., Kim, J.: Belief Revision Process Based on Trust: Simulation Experiments. 4th Workshop on Deception, Fraud and Trust in Agent Societies, Montreal Canada, 2004; pp. 1-12.

[BF04] Boland, D., Fitzgerald, B.: Transitioning from a Co-Located to a Globally-Distributed Software Development Team: A Case Study at Analog Devices Inc. 3rd. International Workshop on Global Software Development, Edinburgh, Scotland, 2004; pp. 4-7.

[Bu01] Butler, B.: Membership Size, Communication Activity and Sustainability: A Resource-based Model of Online Social Structures. Information Systems Research, Vol. 12, No. 4, 2001**;** pp. 346-362.

[GBK04]Geib, M., Braun, C., Kolbe, L., Brenner, W.: Measuring the Utilization of Collaboration Technology for Knowledge Development and Exchange in Virtual Communities. Proceedings of the 37th Annual Hawaii International Conference on Systems Sciences (HICSS), Vol. 1, 2004.

[Ha01] Hamman, R.: Computer Networks Linking Networks Communities. Werry, C., and Mowbray, M. (Eds.), Online Communities, Prentice Hall, 2001; pp. 72-92.

[HW00] Huysman, M., Wit, D.: Knowledge Sharing in Practice. Dordrecht, 2000.

[MAI97] Mynatt, A., Addler, A., Ito, M., O'Day, V.: Design for Network Communities. Online Proceedings of CHI (www.acm.org(sigs/chi97/proceedings), 1997.

[Sk03] Skyrme, D.: Knowledge Management: Making Sense of an Oxymoron. Management Insight, Vol. 22, 2003.

[SVP07] Soto, J. P., Vizcaíno, A., Portillo, J., Piattini, M.: Applying Trust, Reputation and Intuition Aspects to Support Virtual Communities of Practice. 11th International Conference on Knowledge-Based and Intelligence Information & Engineering Systems (KES), LNCS 4693, pp. 353-360, 2007.

[SS01] Stanoevska-Slabeva, K., Schmid, B. (2001). "A typology of online communities and community supporting platforms." Proceedings of the 34th Hawaii International conference on System Sciences (HICSS) **7**: 7010.

[UHN98] Ushida, H., Hirayama, Y., Nakajima, H.: Emotion Model for Life like Agent and its Evaluation. Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI'98 / IAAI'98), Madison, Wisconsin, USA, 1998: pp. 8-37.

[WG99] Wellman, B., Gulia, M.: Virtual Communities as Communities: Net Surfers Don't Ride Alone. Smith, M.A., and Kollock, P. (Eds.), Communities in Cyberspace. Routledge, London, 1999.

Navigation Recommendation On Knowledge Artifacts

Jörn David
Institute of Computer Science I1, Technical University Munich
Boltzmannstrasse 3, 85748 Garching, Germany
david@in.tum.de

Abstract: The central problem addressed by this interdisciplinary paper is to provide navigation recommendation on knowledge artifacts represented as nodes of a graph. Actual user navigations on those graphs are captured, which reflect the browsing behavior and interests of many users. The observed navigations are exploited to predict a sequence of artifacts that are relevant to the user. Existing approaches to navigation recommendation are mostly based on symbolic methods like *Association Rule Mining* (ARM), which cannot incorporate the semantics of the entities to be recommended, since their content is not considered at all. Furthermore agile software development principles entail several implications on the desired machine learning support.

We suggest a connectionist approach based on a *Recurrent Neural Network* (RNN) in conjunction with rich content representation to provide navigation recommendation. Our ensemble of machine learning methods is also able to generalize onto unseen artifacts – meaning those that have not been present in the training set. Advanced text analysis methods such as *Latent Semantic Indexing* (LSI) are employed to consider the textual attributes of these artifacts. The proposed technique combines the benefits from both *content-based* and *collaborative filtering* techniques, since the navigation preferences of many users are taken into account. The observed navigation histories are fed into the neural network, which exploits the possibly heterogeneous content of the presented artifacts in order to predict a sequence of related artifacts. We applied our recommendation technique to a publicly available database of web page navigations based on representative log files. The evaluation showed that every second artifact was properly predicted by our machine learning approach using rich content representation.

1 Introduction

As subarea of knowledge management, knowledge sharing is an activity through which information, skills or expertise are exchanged among the members of an organization [NMR+03]. This paper focuses especially on the provision of skills and expertise in form of *control knowledge* [UK95] that becomes manifest in user navigations. Navigation recommendation is a form of user assistance that assists the user with the search for relevant artifacts or the exploration of an artifact space. Normally certain types of users prefer or require certain types of artifacts like documents, manual pages, system models, multimedia items etc., but dislike or ignore other ones. The users exhibit individual behaviors when traversing the navigation space, driven by their different information needs and intended tasks. The central problem of this paper is to learn the users' information needs from the visited artifacts by *learning by example*. Users navigate in an associative way by perform-

ing sequences of navigation operations to solve a knowledge-driven problem, for example trying to fix a bug by browsing through a software documentation. The goal of navigation recommendation is to predict a *sequence of relevant artifacts* based on the observation of the hitherto user navigation on the knowledge graph, which is detailed in the next section.

A lot of research has been done in the field of navigation recommendation that resulted amongst others in *content-based* and *collaborative filtering* [LSY03] techniques. As opposed to our rich content approach, a symbolic realization of collaborative filtering based on *Association Rule Mining* was given by Fu et al., who tried to exploit the tacit knowledge incorporated in the navigation history [BFH00]. Fu proposes an information recommendation system called *SurfLen*, which suggests interesting web pages to users. Association rule mining as underlying data mining technique is used to discover *frequent 2-itemsets* containing web page URLs like $\{\{p, p_1\}, \{p, p_2\}, \ldots, \{p, p_n\}\}$. When the user is reading page p, then the associated pages $\{p_1, \ldots, p_n\}$ are recommended. So the items stand for URLs here and itemsets are ranked by an intersection measure $rank(U_i, S_j) = |U_i \cap S_j|$ between users' browsing histories $U_i = \{p_1, \ldots, p_m\}$ and mined frequent itemsets $S_j = \{p'_1, \ldots, p'_m\}$. The best k itemsets according to the $rank$-measure are recommended, when the current browsing history is matching one of the stored ones. Such a symbolic approach cannot incorporate the semantics of the entities to be recommended, since the content of the web pages is not considered by the URL-based representation and thus no similarity measure is imposed on these nodes. This deficiency can be solved by considering a *rich representation* of artifacts instead of reducing them to meaningless identifiers.

2 Machine Learning Based on Rich Contents

The software life cycle is a complex and changing process. Modern software engineering methodologies like *Agile Model-Driven Development* that have overcome rigid development processes like the *Waterfall Model* require tool support that takes into account agile paradigms. Machine learning tools must address the basic principles of *agile methods* as far as recommendation functionality is concerned. The following agile principles have direct implications for navigation recommendation on rich knowledge bases.

- **Dealing with Changing and Incomplete Knowledge.** Change is not abnormal but omnipresent and requires continuous retraining of the recommender system, which is more feasible in case of learning from example as opposed to maintain static user profiles. Automatic retraining even leads to a self-adapting recommender system that "follows" the agile development process. To deal with incomplete and uncertain knowledge, the machine learning algorithm must be robust, which is an inherent property of connectionist machine learning methods.

- **Content is More Important Than Representation.** [Amb06] This claim is taken into account by the *multi-representation* of domain objects defined beneath. This technique allows to exploit various types of attributes such as text, categorical, metric, etc. and serves for independence from the respective content representation.

- **Everyone Can Learn From Everyone Else.** [Amb06] This paradigm is especially supported by navigation recommendation, since a project participant can learn from other ones by walking on their paths through the knowledge base, while being proactively guided by recommendations telling how to proceed. Normally a beginner learns from a domain expert, but also an expert can behave like a beginner when faced with unknown sub domains or new processes.

Since in the general case no behavioral rules are available that reveal the browsing behavior of users on a certain knowledge base in form of explicit descriptions, learning from examples is the only way to incorporate knowledge about the users and the application domain. Interests and expertise of the users often become manifest in their *navigation behavior* on the respective knowledge base. The users implicitly reveal their interests and furthermore produce new associations by actually using the system. An example of a tree-shaped knowledge base is depicted by figure 1. Our recommendation technique implicitly

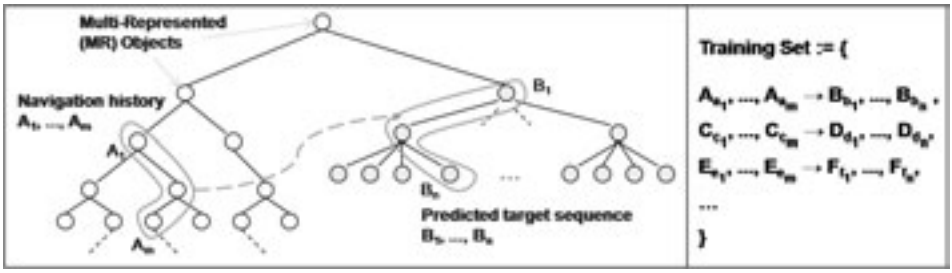


Figure 1: Exemplary user navigation consisting of history and predicted target sequence on a tree-shaped knowledge base. A set of those navigations makes up a *training set*.

addresses the aspect of *collaboration* and knowledge sharing by learning from the navigation behavior of many users, which may be distributed across several teams working on the same project. In a multi-user environment, the navigation histories can be contextualized by the respective user, such that the same sequence of visited objects leads to different recommendations for different users. This can be achieved by encoding the user in the multi-representation of each visited artifact.

Multi-represented objects as constituents of the navigation sequences depicted by figure 1 capture several different aspects of the same domain object. These aspects are called *features* in the field of knowledge discovery, which is equivalent to the denominations *attribute* or *dimension*. A recent example is the encapsulation of all biometric features of a person like voice pattern, image and finger print in one multi-represented object.

Definition 2.1 ***(Multi-Represented Object)*** *A multi-represented (MR) object is an element of a multi-dimensional feature space: $o = (r_1, \ldots, r_n) \in F_1 \times \ldots \times F_n$, where F_i is a feature that can be weighted by a factor $w_i \in [0, 1]$ additionally. Missing values r_i should principally be allowed in the object representation, since modern machine learning algorithms are able to deal with incomplete data. The allowed feature types are:* ***Unstructured text****,* ***Metric*** *(distance measure),* ***Ordinal*** *(ordered) and* ***Categorical*** *(unordered).*

Table 1: Multi-representation of objects by different features. One table line defines the schema of one object, which can have a different number of features, depending on the object type. The available feature types are: *Text*, *Metric*, *Boolean* (a subtype of *Categorical*), *Ordinal* and *Categorical*. *KPI* means *Key Performance Indicator*.

MR-Object Type	**Feat$_1$[Type]**	**Feat$_2$[Type]**	**Feat$_3$[Type]**	...
Requirement	Name[Txt]	Description[Txt]	...	
Issue	Name[Txt]	Description[Txt]	IsResolved[Bool]	Prio[Ordinal]
Picture	Name[Txt]	Caption[Txt]	Histogram[Metric[]]	
Web page	URL[Txt]	Head[Txt]	Body[Txt]	

Each feature type like *categorical* or *metric* requires its own treatment with respect to its possible value range and the appropriate scaling function to be applied during data preprocessing. A concrete instance of a multi-represented object according to the schema of table 1 above would be an *issue ("Issue_abc01", "Solving the bug in the ODBC database connection", false, 8)*. The configuration of the used feature set can even change within the same discourse domain, for example to represent text documents on the one hand and more formal use case elements on the other hand. Each MR-object that contains attributes standing for unstructured text has to be transformed into a machine-processable representation. The textual content of software artifacts is considered by a text mining approach, which provides a rich semantical representation of artifacts and imposes a similarity measure upon them. We apply an advanced text mining method called *Latent Semantic Indexing (LSI)* for computing a low-redundancy numerical representation (vector space model). Thereby a matrix $M_{i,j}$ of keyword frequencies per text unit (e.g. sentences, paragraphs, sections, documents) is spanned. The rows denote the frequency of occurrence for term i in text unit j, which represents the textual content of an artifact. *Stemming* and *stop-word removal* are realized by the *Apache Lucene* indexing and search framework (http://lucene.apache.org).

Definition 2.2 ***(Sequence Prediction)*** *Sequence prediction is a mapping $P: V^* \rightarrow V^*$ from history sequences to target sequences of nodes, where V is the set of all nodes. The nodes of the graph-based knowledge based depicted by figure 1 stand for arbitrary multi-represented objects. The represented object features F_i are also allowed to be continuous $F_i \in \mathbb{R},\ i = 1, \ldots, d$, for example when predicting the remaining time to complete a new software feature indicated by burn down charts.*

Application to Software Engineering Building *Predictive Software Models (PSMs)* by methods from artificial intelligence such as *classification and regression trees (CART)*, neural networks or case-based reasoning is already a big trend in software engineering. There are a lot of research contributions that propose the prediction of the *project success* [Smi07], the *project costs* [She05], the *project duration* and so on.

The machine learning system presented in this article aims at providing recommendations on a document level rather than on source code, especially not on the fine-grained level of operations. For this purpose, the work of Zimmermann et al. [ZWDZ04] might be more appropriate, who applied non-rich but symbolic *association rule mining* to analyze code changes that frequently occur together and thus is able to give recommendations upon

source code fragments. A similar tool that recommends example code from third-party libraries and APIs is *Strathcona* [HWCM05], which operates on a relational database.

Our approach is beneficial when used on rich entities such as issues, requirement documents or communication artifacts consisting of unstructured text or which are even multi-represented by several attributes. As an example, during *controlling* of a software project the responsible project participant might want to know which artifacts have typically been changed together. When a requirement is changed during analysis (Requirements Analysis Document (RAD)) or an issue is attached, the affected use case documents and diagrams also have to be adapted. Figure 2 shows an exemplary landscape of requirements, issues and use case artifacts that are often changed together. The controller can quickly browse

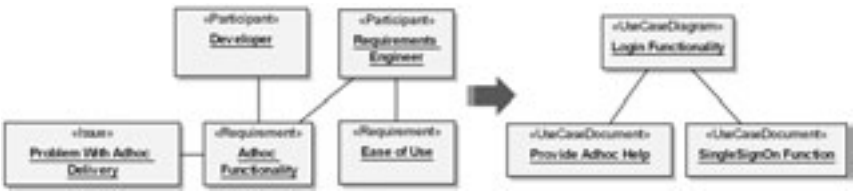


Figure 2: UML object diagram showing different types of development artifacts concerning a guided login functionality, which are frequently changed together. These analysis objects can be interpreted as nodes of a project graph in terms of figure 1. In this scenario, the left partition of objects is sighted by the project controller and the right partition is recommended despite the absence of a hard-wired link between them (missing inter-model link). The project participants are modeled as objects, since the system also exploits their profiles for recommending related artifacts.

through the changed requirements, while the system proactively recommends the relevant use case documents upon content-based similarity *and* the navigations performed by the participants from the analysis team.

3 The Recurrent Neural Network

The *Modular Recurrent Neural Network* (MRNN) is employed as enabling technology in this paper, thus its detailed design is out of scope here. The main purpose of the MRNN is to learn navigation sequences on a knowledge graph and to predict the most appropriate *target sequence* according to the domain-specific user interests. Main benefits of neural networks are their inherent context-representation as well as high generalization capability, flexibility and robustness. The neural prediction is computed on the basis of the recent *navigation history* $\vec{x}_{t-k}, \ldots, \vec{x}_t$ according to the observed user behavior. Figure 3 illustrates the topology and the information flow of the employed recurrent neural network. $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ represents the associated sequence of target nodes, which are predicted to correlate strongest with the observed navigation history. An important characteristics of the recurrent network design is its inherent temporal memory provided by the *hidden state layer* $\vec{s}_{t-k}, \ldots, \vec{s}_{t+m-1}$. Due to the *temporally unfolded network topology*, the MRNN can learn the *sequential structure* of a set of node sequences. Thus the navigation behavior

of many users is explicitly modeled in the hidden state layer. One input vector $\vec{x}_t$ or output vector $\vec{y}_{t+1}$ corresponds to one node of the knowledge graph.

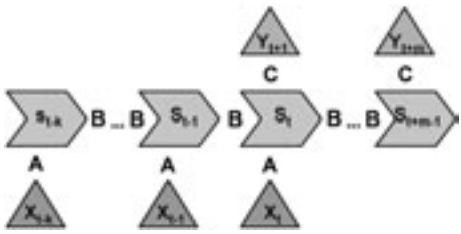


Figure 3: *Schematic topology of the proposed modular recurrent network MRNN. The block arrows indicate the internal state transition $\vec{s}_t \rightarrow \vec{s}_{t+1}$. A, B and C are weight matrices. $\vec{x}_t$ is the external input vector at time t, $\vec{y}_{t+1}$ is the correspondingly predicted output vector. The depicted block arrow direction shows the forward propagation phase.*

In order to process variably long node sequences $\vec{x}_{t-k}, \ldots, \vec{x}_t$, $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ as part of the same training set, we developed a modular neural network. Its technical design is defined by the following recurrent model describing the forward propagation phase.

- A is a ${}^h\mathbb{R}^{d_1}$, B is a ${}^h\mathbb{R}^h$ and C is a ${}^{d_2}\mathbb{R}^h$ matrix. d_1 is the dimensionality of the input- and d_2 is the dimensionality of the output feature space.
- $h = dim(\vec{s}_i)$, (i = t-$k, \ldots, t$+m) is the dimensionality of the state layer. h is independent from d_1 and d_2 and was set to $h = 20$ (experimentally determined) to provide sufficient network resources.

$$\vec{s}_t \quad = \quad f(B\vec{s}_{t-1} + A\vec{x}_t), \qquad \vec{o}_{t+1} = f(C\vec{s}_t) \tag{1}$$

$$\vec{o}_{t+i} \quad \overset{training}{\longrightarrow} \quad \vec{y}_{t+i}, \quad i = 1, \ldots, m \tag{2}$$

As introduced above, $\vec{s}_t \in S$ stands for the internal state at the discrete time step t. The crucial recurrent equation 1 combines an external input $\vec{x}_t$ with the previous state $\vec{s}_{t-1}$ to the subsequent state $\vec{s}_t$, which indirectly depends on all foregoing external inputs $\vec{x}_{t-k}, \ldots, \vec{x}_{t-1}$ *and* internal states $\vec{s}_{t-k}, \ldots, \vec{s}_{t-1}$. In case of supervised network training, the target symbols $\vec{y}_{t+1}, \ldots, \vec{y}_{t+m}$ are known. Here, the activation function is chosen as sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$. During one training epoch, the network adapts at runtime to the individual structure of the respective node sequence with its history and target part. In the operational application of the trained network, two functions are provided:

1. *Recognition and Recovery*: When presenting a sequence of artifacts to the MRNN that equals the *history part* of a learned navigation sequence, the subsequently visited artifacts can be *recovered* and *recommended*.
2. *Generalization*: A virtually unknown sequence of software artifacts is fed into the MRNN, which is then able to predict the most likely related artifacts.

The MRNN operates like a *dynamic system* by explicitly modeling a state layer, which also enables the systematic processing of structured knowledge, which is often used in the knowledge and software engineering domain.

There are still a few *limitations* of the connectionist approach: 1. The length of the navigation history has to be chosen as parameter in order to know when to compute a prediction. Alternatively several recommendations can be computed for several histories of different lengths. 2. Relational training patterns cannot be processed directly, these have to be transformed to functional mappings of history and target sequences.

4 Application to Real-World Navigation Data

We applied our recommendation technique to a knowledge base of web page visitations, since no actual software engineering datasets have been available that reflect the navigation behavior on project artifacts. Microsoft has logged web access data concerning its portal *www.microsoft.com* by sampling and processing the server log files, resulting in 32,700 user navigations on 329 different web pages [BHK98]. The data reflects the usage of the internet portal by 38,000 anonymous, randomly-selected users. For each user, the set of all visited web pages in a timeframe of one week is stored, which is considered as the user's navigation. The web page access data was split up into an arbitrary training set of 2,000 cases and a test set of 1,000 cases for evaluating the connectionist generalization capability. The MRNN is trained on navigation sequences such as $(1081, 1082, 1040 \mapsto 1001)$ or $(1081 \mapsto 1040, 1001)$, where the integers identify a single web page and their textual titles are represented using *latent semantic indexing*. Navigation sequences are not bound to specific users but are representative for all of them. After training, all node sequences where *recovered successfully* when navigating according to the learned history nodes, such that the corresponding sequence of target nodes was recommended properly.

A successful recommendation is given, if a web page visit out of the exemplary subset $H := \{1081, 1082, 1040\} \subset TC$ of a single test case, e.g. $TC := \{1081, 1082, 1040, 1001, 1018, 1083\}$, is presented and the network predicts one node of the complementary set $T := \{1001, 1018, 1083\}$. In general, one node $u \in H$ is presented and one node $v \in T = TC \setminus H$ of the target subset is expected as correct recommendation, which counts as hit and is summed over all test cases. The system was able to deal with unseen web pages, that means such that had never or only partially been presented to the neural network in the training phase. We trained the network on 2,000 navigation examples, whereupon the target nodes of 1,000 test sets could be inferred with a precision of 50.00%.

5 Conclusion

We have introduced a semantic navigation recommendation based on our *Modular Recurrent Neural Network (MRNN)* in conjunction with advanced text mining capabilities. The MRNN enables learning of variably long node sequences on graph-based and domain-

specific knowledge bases. We modeled the users' navigation behavior as paths through a knowledge base of artifacts and trained the network on the observed navigation sequences in a supervised manner. Altogether we have designed and fully implemented a recommender system that fulfills two main tasks: ***1. Learning and recovering of node sequences on graph-based knowledge bases*** and ***2. Prediction of node sequences according to the users' interests and in consideration of their multiple contents***. The learned user preferences are generalized by the MRNN, which predicts relevant artifacts upon *unseen* navigation histories. Connectionist recommendation combined with latent semantic indexing achieved an accuracy of 50% on a Microsoft test set of 1,000 navigation patterns each consisting of history and target sequence. Every second node that was recommended in the evaluation matched the users' interests. Compared with the probabilistic *n-gram recommendation* based on NASA web server logs by Sun et al. [SCWM02], which achieved a precision of about 37%, our technique shows a significantly higher performance.

References

[Amb06] Scott W. Ambler. *The Object Primer: Agile Model-driven Development with UML 2.0*. Cambridge, 2006.

[BFH00] Jay Budzik, Xiaobin Fu, and Kristian J. Hammond. Mining Navigation History for Recommendation. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 106–112. ACM Press, New York, NY, USA, 2000.

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, 1998.

[HWCM05] Reid Holmes, Robert J. Walker, and Gail C. Murphy. Strathcona example recommendation tool. In *ACM Software Engineering Notes*, volume 30, pages 237 – 240, 2005.

[LSY03] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. In *IEEE Internet Computing*, volume 4, 1, 2003.

[NMR+03] Donald M. Norris, Jon Mason, Robby Robson, Paul Lefrere, and Geoff Collier. A Revolution in Knowledge Sharing. 2003.

[SCWM02] Xiaoming Sun, Zheng Chen, Liu Wenyin, and Wei-Ying Ma. Intention Modeling for Web Navigation. In *Proceedings of the 11th World Wide Web Conference*, 2002.

[She05] M. Shepperd. Evaluating software project prediction systems. In *Software Metrics, 11th IEEE International Symposium*, pages 2 pp. –, 2005.

[Smi07] Darja Smite. Project Outcome Predictions: Risk Barometer Based on Historical Data. In *International Conference on Global Software Engineering (ICGSE07), Munich, Germany*. Riga Information Technology Institute, Latvia, 2007.

[UK95] Alfred Ultsch and Dieter Korus. Integration of Neural Networks with Knowledge-Based Systems. In *Proc. IEEE Int. Conf. Neural Networks, Perth/Australia*, 1995.

[ZWDZ04] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining Version Histories to Guide Software Changes. In *International Conference on Software Engineering (ICSE 2004)*, 2004.

Video-based Information Sharing in Distributed Teams

Maximilian Reiss, TU München, reissm@in.tum.de

Abstract: One downside of using distributed teams in a software engineering project is the extra burden put onto inter-team communication. A common approach to information sharing are written documents. This is very unfortunate from a communication theoretic point of view. Video can transport more information. This paper describes the use of video-based information-sharing without manual post-processing and with as little extra workload as possible. The benefit for distributed teams, the examplary application in a project and future visions and goals are explored.

1 The Problem of Information Sharing

As a current practice, team members write minutes to collect and share information discussed in a meeting. Written documents, like meeting minutes, emails, etc. have the problem that sender and receiver of these documents not necessarily share the same background and mindset when writing (or reading) the document[1]. This problem increases if sender and receiver are from different domains or from different cultural backgrounds with different languages. Even though video minutes might not be a full substitute for written minutes, they can complement these and help shorten the written minutes, since extended rationale for decisions and the course of the decision's discussion can be kept in the video capture and outside of the written minutes. Other usage scenarios using video for information sharing under development here are the use of video for requirement elicitation[COB06], for end-user feedback[Sta07] or meeting capture with a more complicated setup. This paper focuses on capturing meeting information without complicated hardware setups and without manual post-processing.

Interpersonal communication has been an area of research over the last decades. There are many models and theories regarding interpersonal communication. There is the four-side-model of Friedemann Schulz von Thun, for example, stating that there are four sub-messages in each message [SvT81]. The sub-messages are: the factual sub-message, the self-revealing sub-message, the relationship sub-message and the appeal sub-message. A model aiming at the physical side of communication is the Encoding / Decoding Model described by Hall in [Hal73]. His model is based on Shannon's model for communication channels [Sha48]. The basic message is that there is interference in interpersonal communication like ambiguity, lack of attention, limited perception, etc. Albert Mehrabian states in [Meh71] his "7%-38%-55% Rule" which highlights the importance of the nonverbal part of communcation[2].

[1] "What the writer had in mind, what the writer actually wrote, what the reader reads, what the reader thinks the writer ment ..."

[2] It is important to stress that his findings are about communication of feelings and attitudes and not generally

A written document does not cover vocal or visual content and context at all. Given the four sub-messages and the interference in interpersonal communication together with the importance of nonverbal communication, the usefulness of video recording for information transfer seems to be evident.

2 The Idea: One-Klick Video-Based Meeting Minutes

The question following the insight that video is better than just plain text regarding mutual understanding is if it is worth the extra effort. Our idea is to capture team meetings, design meetings, brainstorming sessions and presentations on video, analyze these in post-processing steps and publish the resulting media project-internally. These video artifacts introduce the coverage of vocal and visual content and therefore improve the comprehensibility of these meetings' content. In these video artifacts the positions of interest, e.g., new issues or decisions reached, should be marked to ease the navigation in the meetings' captures. Theses markers (chapter marks) in the video can additionally be linked from external resources, e.g., a webpage listing the important issues or comprehensive written meeting minutes.

An additional benefit is that the post-processing of the recording is automated and therefore does not put an extra time burden on any team-member. Examples for such recordings can be seen in Figure 1.

Figure 1: Example Team Meeting Recording

Our requirements for the video-based information-sharing system are that the whole system is non-intrusive, non-distracting and works with a minimal hardware configuration. And there should be as little manual post-processing as possible. The only manual task that might still be necessary is linking the written minute points to the marked sections in the video content. Meeting attendees are not filmed individually and the technical equipment should be operated without extra staff. It should be a "one-klick" solution.

valid, still stress the importance of nonverbal communication.

3 Technical Background

Based on the ideas of the previous section, we describe in the following section the resulting setup we experiment with. For our initial experimental setups we use simple DV camcorders connected to a computer. Both, audio and video, are captured with theses cameras, there are no extra microphones or other audio recording hardware deployed. These cameras are mounted flexibly, since another benefit of having a camera in a meeting is that additional artifacts - like whiteboard drawings, acting of use-cases, gesture explanations, etc - can be recorded directly. Currently, the teams have a (web-based) user-interface, in which a member of the team needs to select the actual meeting room and the team name. Afterwards, one button is pressed to start the recording of the meeting.

When the meeting is finished, all needed post-processing is done automatically. The recorded content is sent automatically as a new job to the Podcast Producer [pod] solution, which offers a very flexible media computation engine. This workflow based system works with atomary computation tasks which can be distributed over a grid. These tasks can be processed sequentially, or more important, in parallel. Therefore, job processing scales with the number of computational nodes available in the grid.

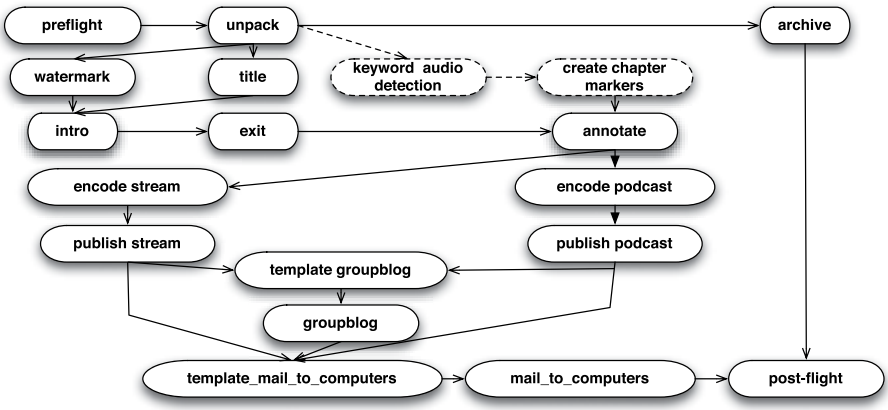


Figure 2: Post-Recording Workflow for Meetings

The workflow in Figure 2 shows a typical post-recording workflow for the recording of team meetings. After the initial unpacking of the material, the video material gets watermarked and parallel to this, a title movie - stating author and title of the movie - is generated. Here it is possible to have several content analyzing tasks in parallel, e.g., an analysis of the audio stream with the goal to capture keywords for setting markers in the final video. Afterwards, different media versions are encoded. After these encoding steps, the streaming movie and the download link to the download version are posted to the team's website and an email is sent to the submitter (recorder) to report the completion of the job.

Apart from the team meeting recordings, we have presentation recordings with the dif-

ference that there are two video sources in a presentation recording. The video of the presenter and the screen capture of the slides presented. Both are merged into one video presentation picture-by-picture. This kind of view has the advantage that both, the presenter's nonverbal communication as well as the slides' content, can be seen.

4 First Observations

The described video-based information-sharing idea is tested in a large scale student project currently running with 50 students in 11 teams at the Chair for Applied Software Engineering of the TU München. These teams are not working full-time and not on a regular schedule. Inter-team communication is done via mailinglists, instant-messaging and each team has access to the team collaboration sites of all other teams. Each team has its meetings recorded and uses the workflow described. A recording is automatically edited, processed, encoded and posted on the team's site. For project-meetings the presentation workflow is used to record the presenter as well as the presentation slides.

The meeting recording feature was adapted quite well and it is considered easy enough to use. There was some resistance to being filmed in the beginning, but it is decreasing since. Apart from the inter-team communication aspect, the meeting records also help in-team for team members who missed a meeting. The feedback for this use-case was very positive and some team members prefer the video over the written meeting minutes or use it in addition to them. The recordings are useful to readers reading minutes, wondering what the writer of these minutes actually meant with that entry. The main challenge is currently to find the important sections in a recording. Ideas how to solve - or at least to reduce - this problem are described in the next section.

5 Next Steps

Based on the initial experimental setup described earlier, we are currently investigating strategies to resolve some of the shortcomings noted in section 4. Currently ongoing work is the question how to introduce markers (bookmarks / chapter marks) in the final video pointing to important moments in the recorded session. Initially, we focus on detecting and marking new action items and issues coming up in a team's discussion. We currently evaluate different strategies. The first strategy is the classic speech recognition approach. With a good speech recognition in place it would be possible to generate written documentation. This can be written minutes with issues and action items, but also a full written transcript. With speach recognition in place, newly identified issues and action items in the video can be marked and can also be directly inserted into a collaboration system (e.g., Sysiphus [Sys]) and from there directly linked to the right mark in the recording. Furthermore, Sysiphus can be used to generate written meeting agendas and written meeting minutes. Therefore, the marked sections in the video can be linked to an entry in the meeting minutes. The second strategy under consideration is the use of image processing

frameworks like OpenCV to identify image patterns of certain gestures. Ideas range from colored signs, which one of the persons in a meeting has to put up in front of the camera, over certain hand gestures in front of the camera indicating a new issue or action item. These extra media analyzing tasks can be added to the earlier described post-recording workflows as tasks parallel to the encoding tasks.

Given the problem that even with markers the sections to watch in the video can be lengthy, it could be really interesting to incorporate a rapid ”skimming” feature like used in the new Apple iMovie. A similar approach of interest is using ”slit-scans” as described by Nunes et al. in [NGCG07]. To transform the information-sharing to knowledge-sharing, semantically analyzing the video is another idea under evaluation. Once these ideas are in place, we plan to conduct usability studies and want to evaluate the use of these video-capturing techniques for distributed teams and distributed inter-cultural teams.

6 Conclusion

Given the early state of our research, we have no definite result yet regarding the usefulness of these team-meeting recordings, but feedback from team members is very positive. With the additional ideas noted before on the way, we think that the usefulness will increase with each new feature added.

References

[COB06] Oliver Creighton, Martin Ott, and Bernd Bruegge. Software Cinema - Video-based Requirements Engineering. In *14th IEEE International Requirements Engineering Conference*, September 2006.

[Hal73] Stuart Hall. Encoding and Decoding in the Television Discourse, 1973.

[Meh71] Albert Mehrabian. *Silent messages*. Belmont, 1971.

[NGCG07] Michael Nunes, Saul Greenberg, Sheelagh Carpendale, and Carl Gutwin. What Did I Miss? Visualizing the Past through Video Traces. In *ECSCW'07: Proceedings of the Tenth European Conference on Computer Supported Cooperative Work*, 2007.

[pod] Podcast Producer. http://www.apple.com/server/macosx/features/podcasts.html.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication, 1948.

[Sta07] Harald Stangl. Integration of multimedia-enable end-user feedback into an iterative software development lifecycle. Diploma thesis, TU München, 2007.

[SvT81] Friedemann Schulz von Thun. *Miteinander reden: Störungen und Klärungen. Psychologie der zwischenmenschlichen Kommunikation.* Rowohlt, Reinbek, 1981.

[Sys] Sysiphus. http://sysiphus.org/.

Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Development

Patrick Mukherjee[1], Aleksandra Kovacevic[2], Andy Schürr[1]

[1]Real-Time Systems Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
mukherjee, schuerr@ES.tu-darmstadt.de

[2]Multimedia Communications Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
sandra@KOM.tu-darmstadt.de

Abstract: In this work, we analyze the potential benefits of taking a peer-to-peer approach to supporting tools for distributed agile software development. Current tools that support software development are client/server based and designed for on-site development. We discuss their drawbacks and show how a peer-to-peer based approach can overcome them. Additionally, we present a peer-to-peer based tool, ASKME, that supports distributed agile software development and focuses on its crucial needs, namely communication and awareness among distributed developer teams.

1 Introduction

The success of agile methods in software engineering made them attractive to a wide range of developers, all focused on getting fast results from the reduced overhead of the classical software development process. Most software projects are developed by distributed teams, and taking into account that agile methods like pair programming and frequent scrum meetings are meant for collocated developer teams, the following question arises: can agile methods be applied in distributed software development? An online survey [ISHGH07] showed that 33% of the observed distributed development projects consisted of teams with less than ten members. Having small-sized teams, like having direct communication between team members, plays a crucial role in the application of agile methods. There is an obvious need to compensate for the inability to meet face-to-face; coming from the field of software engineering, we try to confront this challenge with a tool-oriented approach.

Distributed software development is currently supported by traditional tools, which are created for on-site development. Such tools are based on the client-server communication paradigm. However, the centralized architecture of client/server systems is orthogonal to the natural structure of distributed software development. Developer teams in the same location normally communicate with one another and manipulate the same artifacts. This communication (including message exchange and file transfer) obviously does not need a remote server and should be processed directly between developers.

In this paper we discuss the benefits of using a peer-to-peer communication paradigm for agile software development methods. Therefore, we first identify some key concepts of agile development methods in Section 2 and then, in Section 3, we discuss the problems that the client/server approach brings. In Section 4, we show why peer-to-peer based tools would overcome the previously discussed limitations and in Section 5, we present a peer-to-peer based tool called ASKME that supports agile software development. This tool focuses on the need for developer awareness in distributed teams and for communication that does not use a central server. Finally, we conclude our work in section 6.

2 Properties and Needs of Distributed Agile Software Development

Agile development methods aim to get fast results. To achieve this, the actual implementation phase is emphasized and all other phases (e.g. requirements analysis, design, documentation) are reduced to a minimum time consumption. In contrast to classical development methods like the Rational Unified Process [RUP], agile methods are people-centric rather than process oriented. In fact, the design of the software develops not only during the actual coding, but also during the requirements analysis with the customer. Existing tools that support agile development focus on reducing the duration of non-coding phases. Automatic documentation generators (e.g. JavaDoc) or automatic regression tests (e.g. JUnit) reduce documentation and testing phases, respectively. However, the actual need for documentation is weaker than the need for communication because teams in agile development consist of a few highly competent developers who share their project knowledge directly, i.e., face-to-face [ea01]. Most modern software development is distributed, often even over continents. As such makes the typical agile principle of face-to-face knowledge exchange impossible, a tool to support direct *communication* is needed. Besides basic communication, enabling *awareness* is crucial - because the participants are not able to physically see each other, it is important that their presence in the development environment is visible. As soon as a participant appears online, similar to hallway conversations, other participants are reminded about a deferred conversation, etc.

3 Drawbacks of Client/Server Based Solutions

In the client/server communication paradigm, all data exchanges occur between one dedicated machine (the server) and the machines of the team members (the clients). The server must always be available to offer services while the clients only consume the services. Client/server based support for distributed agile development has numerous drawbacks. First of all, it obviously creates a *communication bottleneck* due to the fact that all data exchanges have to be transmitted through a server, often making communication slow, with poor scalability. If two clients, which might be geographically close to each other, want to exchange information, they cope with *additional latency* introduced by a distant server. Furthermore, *suboptimal resource consumption resources* result from exploiting a

server for all services where client machine resources (such as computation power, memory, bandwidth, and storage space) typically remain unused. If the server fails, the whole system stops working and data could be lost, making the server a *single point of failure*. Finally, the client/server approach has *high set-up and maintenance cost*, which is a particular issue for non-profit projects (e.g. open source projects) as they are normally funded by unreliable donations.

4 Motivation for a Peer-to-Peer Approach

In a peer-to-peer paradigm, each communication participant (peer) offers and consumes services. All communication and data exchange is done directly between intended parties. The *significantly lower costs* of peer-to-peer solutions are the result of *self-organization* and *optimal resource usage* from all participants. Because all users maintain their own machines, there are no additional maintenance costs. If a peer fails, it will be replaced automatically; thus good maintenance is not needed. When a peer comes online he has to be able, in a short time, to overtake services offered by his neighbors; thus the applications must be designed to have a short start-up time. Because each peer is only responsible for a part of a service, the start-up time is a part of the time the combined service would take to start on a single server. [SE05]

In contrast to the client/server approach, messages in peer-to-peer communication have *decreased delay*. Two peers can communicate directly with each other. In the case of client/server communication, if two geographically close clients (e.g. in the same room) want to exchange data, they have to send the data to a distant server (e.g. in another city) and experience unnecessary delay. Only the initialization of the communication requires peer-to-peer routing, which is likely to introduce some delay due to the stretch factor [RS07]. But subsequent messages can be exchanged directly, which is shorter than any three-folded way using the client/server approach[1].

One of the key characteristics of peer-to-peer systems is *robustness*, meaning that a peer failure is considered normal and does not cause a system break. The system would only be harmed if a peer and all its replicating peers (usually five peers altogether) crashed simultaneously, before one were able to replicate its state to a new peer. However, even in this unlikely scenario, only a part of the system would be harmed; the system as a whole would remain functional.

[1] Only in the case that the server were on the routing path would the delay be the same.

5 A Peer-to-Peer based Support Tool for Agile Development: ASKME

As an example how peer-to-peer based applications can improve agile distributed software development, we implemented a tool that aims to improve *awareness* and communication among distributed developer teams; we call this tool **ASKME**, 'Awareness Support Keeping Messages Environment'.

5.1 Functionality

The implemented application is designed as a plug-in for the widely used development environment Eclipse. In a side-panel, a user can add his team mates as contacts by entering their nicknames. The tool allows users to track the status of their colleagues, e.g. wether one is absent or free to respond.

The best source of information while having problems with collaborative coding is to exchange information directly, according to the agile principles. That means contacting the developer who modified the code most recently. Exactly that developer gets automatically highlighted once a document is opened, which is stored by an integrated version control system[2]. The user will be made aware of the contact and reminded that he can ask him questions. If the contact is offline a users can still send messages, which will be automatically delivered by other peers as soon as the destination appears online. In existing instant messengers, such a message would arrive only when both communicating parties are online, meaning that the message could be delayed as long as the sender is offline.

5.2 Architecture

To gather context information about participants' presence, our instant messenger application is built as a plug-in for Eclipse. Using methods to access the version control repository from Eclipse, version information can be accessed (e.g. the last author of a document). It is also possible to access other kind of information in the development environment like contextual (e.g. chatting with bob, working on document X) or user-specified (e.g. busy).

For message exchange, ASKME uses the structured, DHT-based peer-to-peer overlay network *pastry* [RD01], namely its implementation *freePastry* [FP]. When a message is sent it gets routed to the contact nearest the destination the sending peer knows, as described in [RD01]. This peer will automatically forward the message until the destination peer is reached. If the destination peer is offline, our algorithm handles the delivery as follows: due to pastry's routing, the last receiver is nearest to the destination; this receiver will replicate the message to his neighbors, which are the next nearest peers to the intended destination. If peers are changing, the message will be replicated via the standard methods in structured peer-to-peer systems, thus ensuring that it will be stored by the peers surrounding the offline destination. If the destination peer finally comes online, the message gets replicated to it. Upon recognizing that the message was addressed to him, the destination peer will remove it from all replicating peers. We have chosen to replicate

[2] We are currently working on a peer-to-peer based version control solution.

offline messages on five neighbors. The message could only get lost if all five were to fail at the same time, which is very unlikely ([RD01]). Fail means that a peer leaves without being able to clean up its connections, that is, to handover stored offline messages to its neighbors and to announce to all contacts that it is going offline. In the case that both communication partners are online, the first message will reach its destination and establish a direct connection by informing both peers about the others IP address. Thus all following messages in a conversation can be exchanged without peer-to-peer-layer routing.

The implemented application could have been built using client/server based communication, but that would have introduced the drawbacks described in section 3, namely the dependency on a single machine, which would make the system less robust, and which would represent a bottleneck, which would reduce the overall communication bandwidth and increase the delay.

6 Conclusion

When applying agile methods to distributed software development using a client/server based approach, numerous issues arise, the main being bottlenecked communication channels, time consuming setup, and maintenance work. In this paper we discussed how peer-to-peer based tools address nearly all of these problems and how they meet the main needs of agile software development - communication and awareness. As an example, we implemented a peer-to-peer based instant messenger as an Eclipse plug-in, with additional features that support agile teams in their communication and aid in building awareness.

References

[ea01] Beck et al. Manifesto for Agile Software Development. `http://agilemanifesto.org/`, 2001.

[FP] FreePastry. `http://www.freepastry.org/FreePastry/`.

[ISHGH07] T. Illes-Seifert, A. Herrmann, M. Geisser, and T. Hildenbrand. The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey. In *Proc. of GREW*, pages 55–66, 2007.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.

[RS07] A. Richa and C. Scheideler. Overlay Networks for Peer-to-Peer Networks. In *Handbook of Approximation Algorithms and Metaheuristics*, chapter 72. May 2007.

[RUP] Rational Unified Process. `http://www-306.ibm.com/software/awdtools/rup/`.

[SE05] Ralf Steinmetz and Klaus Wehrle (Eds.). *Peer-to-Peer Systems and Applications*. Springer, Sep 2005.

Peer-to-Peer Issue Tracking System: Challenges and Solutions

Vijay Srinivas Agneeswaran, Rammohan Narendula, and Karl Aberer,
Distributed Information Systems Lab (LSIR),
Ecole Polytechnique Fédérale de Lausanne,
{*vijaysrinivas.agneeswaran, rammohan.narendula, karl.aberer*}*@epfl.ch*.

Abstract: Efficient and secure knowledge sharing is critical to the success of any software development project. Traditional knowledge storage systems based on a single server are subject to single point failure and compromise. Any powerful adversary can attack and take over the central server, later may manipulate the knowledge maliciously. This was evident in the past during the attacks on sourceforge, linux development clusters. Peer-to-peer storage distribute the storage on large number of peers and thus making such attacks impossible. P2P systems are increasingly adopted for several additional reasons. We demonstrate a secured, privacy preserving p2p storage architecture and demonstrate its applicability for issue tracking systems which are one of the major resources of knowledge storage in software world.

1 Introduction and Motivation

Issue tracking[1] tools such as Jira or Bugzilla have become important for software development, especially with distributed development teams becoming the norm in several organizations. It is interesting to see that some open source software come with their own issue tracking systems, such as Eclipse bugzilla (https://bugs.eclipse.org/bugs/). However, these tools are centralized and are project specific. The software development community needs a decentralized, project independent issue tracking tool that enables the community of users to share and have access to knowledge.

By centralized, we mean that the existing issue tracking systems are managed centrally - with centralized control. Moreover, they are also hosted on centralized servers. However, both these may not be attractive for the open source community. First, they would love to do away with centralized systems for autonomy reasons. Most users would love to preserve the autonomy of their data and to be able to dictate how it must be shared over the network. Secondly, the centralized repositories may face scalability and fault-tolerance issues with increasing number of clients or users and their geographic spread - mainly due to the fact that bug-tracking or issue tracking in general is a data intensive task. Another key motivation for the work is that knowledge management in software engineering must occur across projects - a problem faced by a software developer (say in debugging or in developing a particular plugin for a particular software) in a particular project may be faced

[1] Issue tracking systems can be seen as a generalization of bug reporting systems.

repeatedly by other developers across the community. Hence, the need for a decentralized autonomous issue tracking tool that would enable software engineering knowledge to be stored, accessed by a geographically dispersed community.

A centralized storage for critical knowledge sharing information systems, is subject to single point failure and compromise. A powerful adversary can control the server and can maliciously make the system unavailable to all intended users or manipulate the knowledge, thus making parts of the knowledge system useless. This may lead to release of buggy or inefficient software products which result in huge economic losses to corporates. Earlier there were several such attacks observed on Sorceforge, Debian linux development cluster [deb03]. After detecting the break-in, the administrators were forced to freeze the development for several days [LKMS04]. Similar attacks have also succeeded against Apache [apa01] [Tay04] also.

2 Overview and Challenges

In this position paper, we identify the key challenges of building a Peer-to-Peer (P2P) issue tracking system and outline the initial steps we have taken in this direction. A first question that can be easily posed is why should there be a P2P solution and why a simple decentralized solution is not sufficient. The answer lies in why P2P systems are actually becoming widely used (not for scalability or fault-tolerance) - autonomy and heterogeneity. Users would like to control how their data must be shared (selectively). Different nodes (used by the users) may use different formats to represent the issue tracking data - leading to data heterogeneity. Data heterogeneity has been a key issue in the semantics web research [SBLH06]. The W3C has come with some standards for the same, including the use of Resource Description Framework (RDF) for data representation.

RDF stores such as Jena [McB02] and others allow RDF elements to be stored in relational database and allow queries over the RDF data, in the form of SPARQL [2] or RDQL[3] queries. Many existing RDF stores store RDF data in centralized repositories, making it difficult to provide scalability and fault-tolerance. Moreover, completely decentralized approaches for data management are required, for reasons of freshness and flexibility [SSSH06].

The key challenges we identify in building a P2P issue tracking system are:

- Scalability: The inability of the infrastructure to degrade gracefully when the storage, network or processing load grows has impaired such efforts in the past.
- Semantic heterogeneity: Different sources store data in different formats or by using different schemas.
- Security: Users must be sure that the data is being shared the way they have specified it. Hence, the need for access control policies and their enforcement in a purely decentralized or P2P setting is very important.

[2] http://www.w3.org/TR/rdf-sparql-query/
[3] http://www.w3.org/Submission/RDQL/

These will address the networking challenges of p2p. Efficient data management on top of p2p networking is very critical for a successful high performing issue tracking system. To minimize the number of network requests per an issue, all the components of an issue should be stored on a single peer. The granularity of data management solution is very critical in this case. If the data management solution decides to split all the components of a single issue across several peers, then querying on a single issue needs that many number of network accesses. Query driven data distribution techniques should be designed.

To summarize, a P2P secure RDF store is the platform we need to build the P2P issue tracking system we visualize. GridVine is an effort that has been made in this regard by us. GridVine [ACMHvP04] is a system that addresses both scalability and semantic heterogeneity, achieving data sharing in large scale P2P systems. GridVine facilitates RDF triple based queries that allow users to retrieve structured content from a P2P system. We are currently building an access control framework over GridVine that would enable users to specify how the data must be shared and enforce it in the P2P network.

3 Security and Access Control

Structured p2p networks are characterized by the fact that resources are stored at the end nodes- the owners, and the index of the shared items is stored with in the p2p network in the form of DHT enabling an efficient search. However, the wide adaptability of the p2p data storage systems is hindered by the fact that virtually all the index information and the resources are exposed to every member of the network. For the systems to mature and evolve towards a replacement for traditional client-server data sharing systems, efficient access control mechanisms should be enforced to control the access of the resources by the peers.

The possible points for enforcing the access control policy are the network itself and the end nodes (typically the owners). We argue that any robust p2p information system should employ the enforcement at both the places, which means controlling the access to *both* indexes and resources in a structured p2p information system. Only authorized users should be able to get results for the searches they do based on keywords. These results may include just the extra meta information associated with the resource as part of the index stored in the DHT or the resource itself. Before allowing a user to access the physical resource, the access control policy is checked and request is granted or denied based on the policy statement associated with the user.

The PHera system proposed in [CSMB05] deals with access control in super peer based p2p infrastructures. And the protocol trusts all the super peer nodes to enforce the access control policy. Sub-peers stores their access control policy information with the super peers. This solution can not be applied to the structured p2p networks. We present some research directions and brief overview of the current solution we are developing. We assume the presence of a secured communication in place, which is free from eavesdropping, traffic analysis by unintended participants in the network. Secured Socket Layer (SSL) based p2p infrastructures are already proposed for this purpose [Ram, Kri, Kra, Yea].

We propose that access control on the index can be done by either a *Controlled Queries* approach or a *Controlled Replies* approach. In controlled queries approach, we ensure that only intended users can post queries for the resources. This can be done in several ways and one of them is by obfuscating the *key* used to search in the system. For example, instead of publishing a resource with name *myReport.doc*, the user publishes with a identifier generated using a *secret key*. So a user who does not have access to this key can not derive mappings between the keywords he is interested in and the actual keys used in the system. A group of users decide the key out-of-band for this purpose. So in such a system, ability to query the system with the right key word itself authorizes the user.

In controlling replies approach, the philosophy is to reply to queries that originate only from the intended users. Access control policy information is stored with the index too and is checked for each request for taking a decision on the request. To enforce this technique, we propose the concept of *Trusted Sub Overlay (TSO)*. Each TSO is an overlay on top of existing overlay consisting of only trusted nodes for enforcing the access control function. All the resources for which the index is stored in a TSO are identified with a `(TSO_ID,resource_ID)` pair. The resources for which the index is stored in the normal overlay are referred with a single identifier. In this model, a node plays two roles one as member of the overlay and member of TSO, hence it is responsible for its identifier space as part of the overlay and that of the TSO. Each TSO has its own complete identifier space.

To search in the public overlay, a search query with just the corresponding resource id is issued and the responsible peer for that id responds. To search in the TSO, a search query with TSOID and resource ID is issued, which is forwarded to one of the nodes in the corresponding TSOID, which in turn forwards the query to the responsible peer for that ID in that TSO. This can be be enabled in many ways. All the peers in the TSO inserts (TSO_ID, node_ID) pair into the public overlay [ABC+02].

4 Conclusion

In this paper, we highlighted the need for a secured distributed storage based on p2p technologies for knowledge storage. We discussed the need for handling heterogeneity in knowledge representation. We then presented the methodologies to make such systems secure and privacy preserved.

Literatur

[ABC+02] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer und Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, Seiten 1–14, New York, NY, USA, 2002. ACM.

[ACMHvP04] K. Aberer, P. Cudre-Mauroux, M. Hauswirth und T. van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *International Semantic Web Conference (ISWC)*, Jgg. 3298 of *LNCS*, Seiten 107–121, 2004.

[apa01] Apache.org compromise report. *http://www.apache.org/info/20010519-hack.html*, May, 2001.

[CSMB05] Bruno Crispo, Swaminathan Sivasubramanian, Pietro Mazzoleni und Elisa Bertino. P-Hera: Scalable fine-grained access control for P2P infrastructures. In *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, Seiten 585–591, Washington, DC, USA, 2005. IEEE Computer Society.

[deb03] Debian investigation report after server compromise. *http://www.debian.org/News/2003/20031202*, December, 2003.

[Kra] Steve Krasinsky. Project JXTA Overview. *http://www.mnlab.cs.depaul.edu/seminar/spr2004/JXTAOverview.pdf*.

[Kri] Navaneeth Krishnan. JXTA and Security. *Technial Report, Sun Micro Systems, http://java.sun.com/developer/Books/networking/jxta/jxtap2pch08.pdf*.

[LKMS04] Jinyuan Li, Maxwell Krohn, David Mazières und Dennis Shasha. Secure untrusted data repository (SUNDR). In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Seiten 9–9, Berkeley, CA, USA, 2004. USENIX Association.

[McB02] Brian McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

[Ram] Narendula Rammohan. A Note on Secure P-Grid. *Technical Report, LSIR-REPORT-2008-005, Swiss Federal Institute of Technology (EPFL)*.

[SBLH06] Nigel Shadbolt, Tim Berners-Lee und Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[SSSH06] Staab, Steffen, Stuckenschmidt und editors. Heiner. Semantic Web and Peer-to-Peer. *Springer-Verlag, Berlin Heidelberg*, 2006.

[Tay04] Owen Taylor. Intrusion on www.gnome.org. *http://mail.gnome.org/archives/gnome-announce-list/2004-March/msg00114.html*, March, 2004.

[Yea] Bill Yeager. P2P Security and JXTA. *Technical Report, Sun Labs, http://www.ida.liu.se/conferences/p2p/p2p2003/presentations/Yeager.pdf*.

Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis

Inhaltsverzeichnis

Workshop Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis 85
Bernhard Hohlfeld, Stefan Kowalewski, Michael Reinfrank, Gerhard Wirrer

Konzeption eines Austauschformates für AUTOSAR Diagramme 88
Vladimir Stantchev, Dawid Kostrzycki

Defining a Timing Model for AUTOSAR – Status and Challenges 93
Kai Richter

The legacy ECU software problem - approach and research challenges 98
Thomas Heinz, Jörn Schneider

BRIEF: Systematisches Prozess-Briefing und –Debriefing. Ein bei der Erstellung komplexer Automobilsoftware praxiserprobtes Entwicklungsvorgehen, das Elemente klassischer und agiler Modelle kombiniert 103
Hannes Omasreiter, Claudia Schlumpberger, Lorenz Widmann

Vom modellbasierten Architekturentwurf über die Simulation verteilter Systeme zum Serienprojekt 108
Rainer Otterbach, Oliver Niggemann

Model-Based Design for IEC 61508: Towards Translation Validation of Generated Code 113
Mirko Conrad

Effizienzsteigerung im Systemtest von Steuergeräten Eine empirische Untersuchung 118
Matthias Wiemann, Jörg Gericke

Workshop Automotive Software Engineering:

Forschung, Lehre, Industrielle Praxis

Veranstalter: Fachgruppe Automotive Software Engineering (ASE)
http://www.gi-ev.de/fachbereiche/softwaretechnik/ase/

Organisation:
Bernhard Hohlfeld, Stefan Kowalewski, Michael Reinfrank, Gerhard Wirrer

bernhard.hohlfeld@daimler.com
kowalewski@informatik.rwth-aachen.de
Mreinfrank@aol.com
gerhard.wirrer@siemens.com

Abstract: Bei modernen Fahrzeugen wird die Funktionalität zunehmend durch Software bereitgestellt. Das Spektrum reicht von der Motorsteuerung bis hin zum Allradantrieb und schon bald zu X-by-Wire-Systemkomponenten. Die zugrundeliegende Rechnerarchitektur ist ein verteiltes System, das je nach Fahrzeugtyp aus 20 – 80 Steuergeräteknoten besteht. Die Knoten sind mit bis zu vier verschiedenen Bussystemen verbunden. Der Programmcode umfasst mehrere hunderttausend bis zu mehreren Millionen Zeilen. Über zwei Drittel aller Innovationen im Automobil sind schon heute softwarebasiert. Ein Anstieg der Softwareentwicklungskosten an den gesamten Entwicklungskosten von derzeit ca. 4% auf über 10% wird prognostiziert. Ein Automobil bündelt so auf ca. 5mx2m viele Fragestellungen der Informatik, insbesondere der Entwicklung komplexer und zuverlässiger Softwaresysteme.

1 Ziele

Der von der Fachgruppe Automotive Software Engineering (ASE) veranstaltete Workshop steht unter dem Motto **"Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis"** und bietet ein Diskussionsforum für Kolleginnen und Kollegen, die mit dem Thema Automotive Software Engineering befasst sind und sich insbesondere für die Wechselwirkungen zwischen Forschung, Lehre und industrieller Praxis interessieren. Zentrale Fragestellungen für den Workshop sind u.a.:

- Was muss die Automobilindustrie – Hersteller und Zulieferer – tun, um für Informatiker als Arbeitgeber interessant zu sein?
- Welche Voraussetzungen müssen Informatiker mitbringen, um für die Automobilindustrie als Mitarbeiter interessant zu sein?
- Welche interessanten Fragestellungen bietet die Automobilindustrie der Wissenschaft?
- Welche praxistauglichen Lösungen bietet die Wissenschaft der Automobilindustrie?

Übergeordnetes Ziel der Fachgruppe Automotive Software Engineering ist die weitere Stärkung der „Automotive Software Community".

2 Schwerpunkte

Die Schwerpunkte des Workshops liegen auf Automotive Software, insbesondere den Themenkreisen Forschung, Lehre und industrielle Praxis sowie deren Wechselwirkung. Hierbei sind sowohl methodische Grundlagen und deren werkzeugunterstützte Anwendung als auch Praxiserfahrungen oder Standardisierungsaktivitäten von Interesse.

Die Schwerpunkte beinhalten unter anderem:

- Requirements Engineering
- Modellbasierte Entwicklung
- Methoden zur technischen Qualitätssicherung (Systematisches Testen, Codeanalysen, …)
- Varianten- und Konfigurationsmanagement
- Standards (OSEK, AUTOSAR, …)
- Ausbildung Automotive-SW-Engineering

3 Workshophistorie

Workshops zum Thema "Automotive Software Engineering fanden auf den GI-Jahrestagungen 2003 – 2007 statt:

27. September 2007 in Bremen
Organisation: Dr. Michael Reinfrank, Siemens VDO Automotive

5. Oktober 2006 in Dresden
Organisation: Dr. Michael Daginnus, Volkswagen AG

21. September 2005 in Bonn
Organisation: Dr. Thomas Kropf, Robert Bosch GmbH

23. September 2004 in Ulm
Organisation: Dr. Bernhard Hohlfeld, Daimler AG

30. September 2003 in Frankfurt
Organisation: Dr. Alexandre Saad, BMW Group

4 Programmkomitee

Leitung
Bernhard Hohlfeld, Daimler AG
Stefan Kowalewski, RWTH Aachen
Michael Reinfrank
Gerhard Wirrer, Siemens VDO Automotive AG
Mitglieder
Hans-Jürgen Belz, Hella KGaA Hueck Co.
Jürgen Bortolazzi, Porsche AG
Manfred Broy, TU München
Mirko Conrad, The Mathworks, Inc.
Michael Daginnus; Volkswagen AG
Werner Damm, Universität Oldenburg
Bernd Frielingsdorf, Ford Werke GmbH
Michaela Huhn, TU Braunschweig
Stefan Jähnichen,TU Berlin und FhG FIRST
Hubert Keller, FZ Karlsruhe
Thorsten Kölzow, Audi AG
Rainer Koschke, Universität Bremen
Thomas Kropf, Robert Bosch GmbH
Hans-Jürgen Kugler, Kugler-Maag
Stefan Ortmann, Carmeq GmbH
Helmuth Partsch, Universität Ulm
Klaus Pohl, Univ. Duisburg-Essen
Wolfgang Pree, Universität Salzburg
Dieter Rombach, TU Kaiserslautern und FhG IESE
Alexandre Saad, BMW Group
Christian Salzmann, BMW Group
Jörn Schneider, Robert Bosch GmbH

Konzeption eines Austauschformates für AUTOSAR Diagramme

Vladimir Stantchev[1] und Dawid Kostrzycki[2]

Vladimir.Stantchev@sysedv.tu-berlin.de[1]
mystique@cs.tu-berlin.de[2]

Abstract: Dieser Beitrag evaluiert Ansätze wie das DI und das GMF Notation Model im Hinblick auf ihre Anwendbarkeit als Austauschformat für AUTOSAR (AR) Diagramme. Anschließend wird ein Ansatz zur Abbildung der Diagramme des AR Metamodells in das GMF Notation (Meta)Model vorgestellt.

1 Einleitung

Heutzutage werden viele innovative Funktionen eines Automobils, wie zum Beispiel Einparkhilfe oder Fahrerassistenz, in Software implementiert. Diese Funktionen sowie die Reduzierung der Hardwarekosten sind Treiber der Elektronikindustrie im Automobilbereich. Es ist zu beobachten, dass ein hoher Ressourcenaufwand besteht, um bestehende Softwarelösungen eines Umfelds in ein anderes zu überführen. Dieser Aufwand entsteht durch die Vielfältigkeit der Komponenten aus denen ein heutiges Fahrzeug besteht. So verfügen Fahrzeuge der Oberklasse über mehr als 60 Steuergeräte, einige Bordnetze und diverse Fahrzeugfunktionen. Diese Komplexität nimmt beständig zu und ist mit traditionellen Entwicklungsprozessen nur noch schwer beherrschbar. Daher wurde ein Konsortium vom Automobilherstellern und Zulieferern für die Standardisierung der Elektrik-/Elektronik-Infrastruktur im Automobil gegründet. Ein Ziel dieses Konsortiums, das den Namen AUTomotive Open System ARchitecture trägt, ist es, die Wiederverwendung von Software herstellerübergreifend zu ermöglichen [FBH+06]. Oft werden in diesem Zusammenhang Informationen, die diese Systeme beschreiben, unter den entwickelnden Parteien ausgetauscht. AUTOSAR (AR) definiert für die Beschreibung dieser Systeme ein Metamodell, welches den standardisierten Informationsaustausch XML-basiert [W3C06] fördert. Dieses Metamodell ist auch Basis für graphische Editoren, mit denen die Entwicklung von AR Modellen auf Basis einer graphischen Notation erfolgen kann. Im Rahmen dieser Arbeit geht es um die Beschreibung des Austauschs von Diagrammen, die mit Hilfe der graphischen Editoren verschiedener Hersteller angefertigt werden. Kap. 2 gibt einen Überblick über die Problemstellungen, die aus der UML Domäne bekannt sind und einen Ansatzpunkt für die Zielsetzung in AR liefern. Danach folgt eine Vorstellung der AR Domäne und ein Konzept zur modellbasierten Beschreibung von Diagramminformationen einer graphischen Fachsprache. Den Schluss bilden eine Zusammenfassung und ein Ausblick.

2 Problemstellung und Zielsetzung

Graphische Designwerkzeuge erleichtern die Beschreibung, Analyse oder Konzeption von diversen Systemen. Ein Beispiel für eine visuelle Sprache ist die Unified Modeling Language (UML)[OMG07b], für welche eine Reihe von Werkzeugen existiert, mit denen man UML Diagramme erzeugen kann. Die Definition dieser Sprache erfolgt in MOF (Meta Object Facility) [OMG06], einer Metasprache, die sich den Kern der Klassenmodellierung mit der UML teilt. Um Diagramme unter den verschiedenen UML Tools austauschbar zu machen, definierte die OMG das Diagram Interchange (DI) [OMG03], das den bestehenden Austauschmechanismus XML Metadata Interchange (XMI) [OMG07a] in seiner Aufgabe ergänzen sollte. Die Idee von XMI ist Datenaustausch MOF-basierender Metamodelle in XML (Extensible Markup Language). Bis zur Entwicklung von DI war es mittels XMI nur möglich den semantischen Gehalt der UML Diagramme nach XML standardisiert zu serialisieren. Diagramminformationen wie Größe, Position oder Farben von graphischen Objekten werden mittels XMI nicht erfasst und gehen verloren. Dabei ist DI kein Austauschformat, wie es der Name nahe legt. Vielmehr ist es ein Metamodell, um Diagramminformationen zu beschreiben. Das Problem der UML ist, dass sie nicht definiert, wie ein UML Modell in ein DI Modell überführt wird. Somit ist die Interoperabilität in Bezug auf den Diagrammaustausch implementierender Werkzeuge nicht gewährleistet, da jedes der DI implementierenden Editoren eine andere Abbildung von UML nach DI realisiert.
Die Domäne AR, die hinsichtlich der Metamodellierung analog zur UML aufgebaut ist, hat eine ähnliche Problematik. AR wird wie die UML durch ein Metamodell beschrieben und definiert eine graphische Notation [AUT06] für bestimmte Teilbereiche. AR-Editoren, die auf diesem Metamodell basieren und Diagramme miteinander austauschen wollen, brauchen folglich ebenfalls ein Austauschformat für Diagramminformationen.
Die Arbeit hinter dieser Publikation, evaluiert und bewertet Lösungen wie das DI und das GMF Notation Model, die für die Spezifikation eines Austauschformats auf Basis eines Metamodels für AR Diagramme in Frage kommen. Ferner erfolgt die Entwicklung eines Ansatzes zur Beschreibung der Abbildung der Diagramme von dem AR Metamodell in das GMF Notation (Meta)Model [ST06]. Dabei werden die essentiellen Bestandteile, die für diese Abbildung wichtig sind identifiziert und es erfolgt die Definition einer solchen Abbildung.

3 AUTOSAR und Austauschformate

AR verändert die Art und Weise der Sofwareentwicklung in der Automobilwelt dahingehend, dass AR von einem ECU (Electronic Control Unit) ausgerichtetem Entwicklungsprozess absieht und die Entwicklung des gesamten Systems adressiert. Ein fundamentaler Bestandteil dieser Entwicklung ist die Trennung von Software-Applikation und Infrastruktur, die einen modellgetriebenen Software Entwicklungsprozess fördert und von der zugrunde liegenden Hardware abstrahiert. Eigenschaften der Software-Applikationen werden mit Hilfe der AR Software Component Templates vollzogen, die ebenfalls einen Teil

des AR Metamodells bilden. Diese Templates beschreiben die Struktur, das Verhalten und die Implementierung der Software. Die Beschreibung der Struktur erfolgt in Form von einer Art UML Komponentendiagramm. So werden einzelne Funktionen in einer Software-Komponente gekapselt, die über Ports mit bestimmten Schnittstellen mit anderen Komponenten kommunizieren können. Auf Basis dieser Templates, von denen es mehrere Bereiche in AR gibt, ist es möglich graphische Editoren zu entwickeln, die das Ausfüllen dieser Templates unterstützen. Im Kontext der Automobil Branche ist es wichtig, dass diese mit den Editoren erzeugten Modelle unter Werkzeugen verschiedener Hersteller austauschbar sind. Ein erster Schritt für den Austausch von AR Modellen, der die Interoperabilität fördern soll, bildet das standardisierte Austauschformat auf XML Ebene, welches durch ein XML Schema [W3C04] beschrieben wird und direkt aus dem AR Metamodell generiert [PB06] wird.
Diese Arbeit unterstützt die Interoperabilität für den Austausch von Diagrammen unter AR Editoren und spezifiziert nach Evaluation zweier (Quasi-)Standards eine Abbildung in das GMF Notation Model. Dadurch wird ein Austauschformat auf Modellebene spezifiziert. Das GMF Notation Model ist Teil des Graphical Modeling Frameworks (GMF) und dient der Speicherung visueller Informationen von Editoren, die die GMF Laufzeitumgebung nutzen oder mittels GMF generiert werden. Dabei handelt es sich um ein Metamodell, das dem DI sehr ähnlich ist und von Diagrammen ausgeht, die hierarchische Graphen sind und im Kern aus Knoten und Kanten bestehen. Das Notation Modell ist in Anlehnung an das DI entstanden und fällt in seiner Anwendung kompakter aus, was die Lesbarkeit und programmatische Wartbarkeit erhöht. Die wichtigsten Aspekte sind, dass GMF seine Praxistauglichkeit in einigen Projekten (Apollo for Eclipse[1], Topcased[2]) unter Beweis gestellt hat und sich sehr gut in die bestehende AR Werkzeuglandschaft integriert.

4 Überführung in das GMF Notation Model

Unabhängig von der zugrunde liegenden Technologie soll eine Abbildung für die Diagramminformationen auf Basis des Notation Modells erfolgen. Diagramme werden für diesen Zweck nicht in ihrer konkreten Darstellung (wie sie in [AUT06] zu finden sind) sondern als UML Objektdiagramm, als Instanz von dem AR Metamodell und dem GMF Notation Metamodell, dargestellt.
Die Abbildung der AR Diagramme erfolgt in Form einer visuellen Grammatik, die durch Transformationsregeln ausgedrückt wird. Dieses Konzept stammt aus der Theorie der Graphtransformation, einer Theorie, die sich mit der Regel-basierten Modifikation von Graphen befasst. Graphen sind in der Informatik ein weit verbreitetes Konzept, welches in vielen Domänen, wie der Modellierung von nebenläufigen und verteilten System, dem Datenbank-Design oder der Sofwareentwicklung Einzug gefunden hat. Allen ist gemeinsam, dass ihnen eine graphbasierte Struktur zugrunde liegt, die aus Knoten und Kanten besteht und dessen Knoten durch Kanten miteinander verbunden werden. Die Gegebenheiten auf denen diese Arbeit aufbaut, legen nahe die Definition der Transformation mittels

[1] www.gentleware.com
[2] www.topcased.org

Konzepten der attributierten Graphtansformation[EEPT06] zu bewerkstelligen. Die grundlegende Idee der Graphtransformation ist die Repräsentation des Systemzustandes durch Graphen, wobei Entwicklungsschritte des Systems durch Anwendung einzelner Transformationsregeln erfolgen. Knoten und Kanten können bei dieser Form der Graphtransformation Attribute aufweisen. In diesem Zusammenhang können Systemzustände als Objektdiagramme angesehen werden, die zu einem bestimmten Klassendiagramm gehören und im wesentlichen Graphen sind [KGKK02].
Eine Regel entspricht einem Transformationsschritt, der ein Model von einem Zustand in einen anderen überführt. Die Menge der Regeln wird in diesem Kontext als visuelle Grammatik bezeichnet. Eine Transformationsregel $p : LHS \Rightarrow RHS$, wie sie in Abb. 1

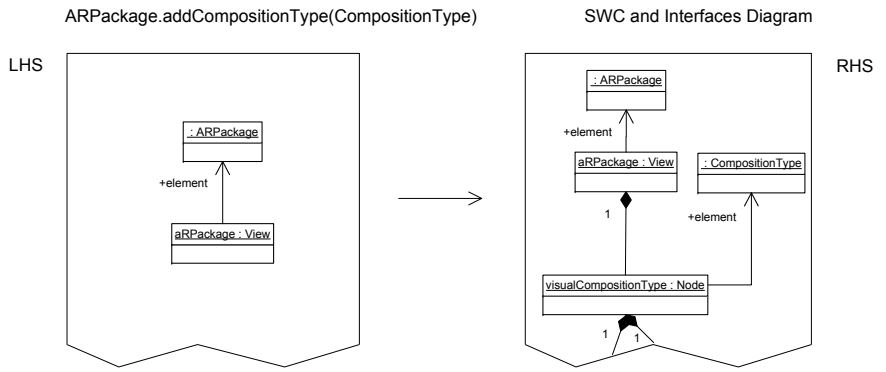


Abbildung 1: Regel für das Hinzufügen einer CompositionType

visualisiert ist, besteht aus einem Regelnamen und zwei Objektdiagrammen in abstrakter Darstellung, LHS und RHS deren Schnitt definiert ist. Die Anwendung der Regel p ist ein Finden des Sachverhalts LHS in einem Modell und das anschließende Ersetzen von LHS durch RHS. LHS ist somit eine Vorbedingung der Regelausführung, während RHS die Nachbedingung darstellt [EEPT06]. Eine Regel kann darüber hinaus durch die Angabe von OCL Constraints semantisch ergänzt werden. Ganze Diagramme werden schließlich durch die aneinanderreihung von Regeln erzeugt, die Regelsequenzen genannt werden. Jede Regelsequenz beschreibt den aufeinander folgenden Aufbau eines Diagramms. Um ein Austauschformat für Diagramme auf Basis von Notation vollständig zu machen fehlen neben der Angabe einer visuellen Grammatik zwei weitere Bestandteile. Der erste Teil betrifft domänenmodellabhängige Visualisierungen. Veranschaulicht sei dies an einem Port einer Software-Komponente. Abhängig vom Interface des Ports, das im semantischen Modell spezifiziert ist, wird der Port verschieden visualisiert. Diese Information ist allerdings nicht im Notation Model enthalten. Der zweite Teil betrifft die Angabe von Layoutalgorithmen, die für die Anordnung von Komponenten einer zu visualisierenden Figur wichtig sind. Zur Illustration dient hier ein Klassendiagramm mit drei Kompartments für den Namen, die Attribute und die Operationen, die untereinander angeordet sind. Würde man einen anderen Layoutalgorithmus wählen, würden dieses drei Kompartments nebeneinander und nicht untereinander dargestellt werden.

5 Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass diese Arbeit für die Domäne AR und damit für den Austausch von Diagrammen unter Werkzeugen verschiedener Hersteller einen weiteren Beitrag leistet, der die bestehenden Interoperabilitätsbestrebungen ergänzt. Ferner schließt diese Arbeit eine konzeptionelle Lücke der UML, die als Wegbereiter der Modellierung für AR gilt. Die UML definiert zwar ein Austauschmetamodell, aber keine ensprechende Abbildung für UML Diagramme. Dies macht den Austausch von Diagrammen unmöglich, da jede DI implementierende Partei seinerseits eine eigene Abbildung nach DI definiert. Die Arbeit hinter dieser Publikation definiert ein Konzept für die Angabe einer solchen Abbildung und darauf basierend eine Abbildung in das GMF Notation Model. Des Weiteren realisiert GMF ein leistungsstarkes Framework, mit welchem schnell graphische Editoren generiert werden können. Diese Editoren können bestehende AR Werkzeuge, wie zum Beispiel das Open Tool Framework (OTF) nutzen, die das Lesen, Schreiben, Zusammenführen und Validieren von Automobil Beschreibungen beherrschen.

Literatur

[AUT06] AUTOSAR GbR. Specification of Graphical Notation, 2006.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange und Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. 2006.

[FBH+06] Helmut Fennel, Stefan Bunzel, Harald Heinecke et al. Achievements and exploitation of the AUTOSAR development partnership. 2006.

[KGKK02] Sabine Kuske, Martin Gogolla, Ralf Kollmann und Hans-Jörg Kreowski. An Integrated Semantics for UML Class, Object and State Diagrams Based on Graph Transformation. 2002.

[OMG03] OMG Inc. UML 2.0 Diagram Interchange Specification, September 2003.

[OMG06] OMG Inc. Meta Object Facility (MOF) Core Specification, January 2006.

[OMG07a] OMG Inc. MOF 2.0/XMI Mapping, Version 2.1.1, December 2007.

[OMG07b] OMG Inc. Unified Modeling Language: Superstructure, February 2007.

[PB06] Mike Pagel und Mark Brörkens. Definition and Generation of Data Exchange Formats in AUTOSAR, 2006.

[ST06] Alexander Shatalin und Artem Tikhomirov. Graphical Modeling Framework Architecture Overview. 2006.

[W3C04] W3C. XML Schema Part 1: Structures Second Edition, 2004.

[W3C06] W3C. Extensible Markup Language (XML) 1.1 (Second Edition), 2006.

Defining a Timing Model for AUTOSAR – Status and Challenges

Kai Richter [1]

CTO, Symtavision GmbH
Frankfurter Straße 3B
38122 Braunschweig
richter@symtavision.com

Abstract: Software timing aspects have not received broad attention in the automotive industry until recently. New design trends and the ongoing work in AUTOSAR have significantly increased the industry's awareness to these issues. Now, timing is recognized a major challenge and has been put explicitly on the AUTOSAR agenda. The goal is to add timing models to the existing AUTOSAR templates. But it isn't all that simple. The paper highlights key technical and non-technical challenges for defining a comprehensive timing model for AUTOSAR, and outlines requirements for possible solutions. Examples from practice and a look into the industrial process of designing –and the way of thinking– shall help structuring the discussions. Finally, recent advancements of the new AUTOSAR timing subgroup and related projects are summarized.

1 Introduction

Current supply-chains in automotive E/E development contain hundreds of companies that design their individual hardware and software components based on requirement definitions from the OEMs or Tier-1 suppliers. Clearly, systems integration has become a key challenge.

To ease the integration in the future, the AUTOSAR partnership [He06] (http://www.autosar.org), an alliance of OEM manufacturers and Tier-1 automotive suppliers with many associates, has established a number of de-facto open industry standards for automotive E/E architectures. Many proven-valuable concepts are borrowed from earlier standards such as OSEK/VDX [OS04][OS05]. The main goal of AUTOSAR is to define a software architecture with standardized APIs and configuration files for application and basic software, which allows exchanging parts of the system's software in ways that programmers know from Java or C++. Key goals are modularity, scalability, transferability and re-usability of software among projects, variants, suppliers, customers, etc. without the time-consuming and costly need to re-configure, port, and re-build the code.

The standardized architecture is complemented with a standardized methodology that shall facilitate integrating components from different suppliers onto one ECU, re-mapping components from one ECU to another, or exchanging whole bus segments within a network of distributed functions; and (indirectly) new business models.

[1] This document is partially based on the TIMMO project in the framework of the ITEA 2, EUREKA cluster Programme Σ! 3674. The work of the German partners has been funded by the German Ministry for Education and Research (BMBF) under the funding IDs 01IS07002 (B-I,K). The responsibility for the content rests with the authors.

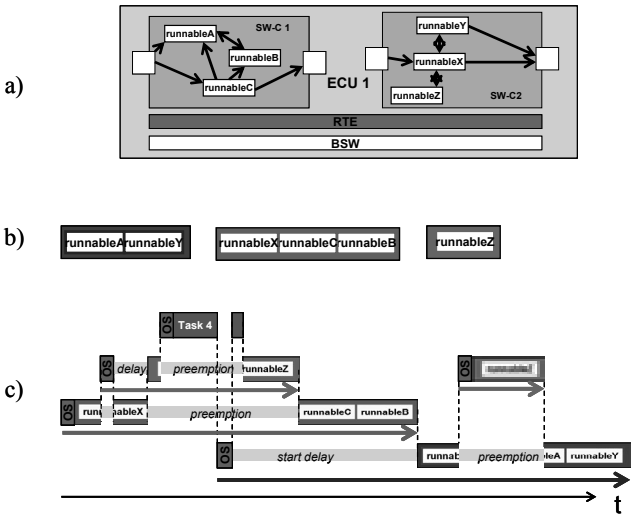


Figure 1 Software Components Structure vs. Task Structure

2 AUTOSAR and Timing

It is important to understand that the primary objective of AUTOSAR is not solving timing problems in particular but supporting integration from a software-engineering perspective. However, system timing properties have a strong impact on key steps in the newly envisioned AUTOSAR methodology. For instance, adding a software component to an existing ECU potentially introduces non-functional timing and performance interference with the original ECU software due to scheduling, arbitration, blocking, buffering etc., eventually generating hard-to-find timing problems, including transient overload, buffer under- and over-flows, and missed deadlines that can finally make the new or the old functions or the entire ECU fail.

Tier-1 suppliers must control such “timing side-effects” which requires knowing timing properties of more or less each involved component. This is already a challenge in the established design process in which a Tier-1 supplier typically has control over the entire software running on a particular ECU. With AUTOSAR, the Tier-1s have to deal with black-box SW components and must still be able to control timing. Of course, not having a systematic timing analysis procedure is challenging future design processes.

There is a large agreement that AUTOSAR needs a timing model. But it isn’t all that simple. In the next section we provide three examples of (technical) model mismatch between the software-engineering view of AUTOSAR and the implementation-concept view needed for timing analysis [Ri07a]. Methodology and supply-chain concerns add to that dilemma, which are summarized in Section 4. We outline recently started activities and their goals in Section 5, followed by a general conclusion.

3 Model Mismatch

In this section we present three key examples of model mismatches that emphasize the complex relations between the timing properties of the system components.

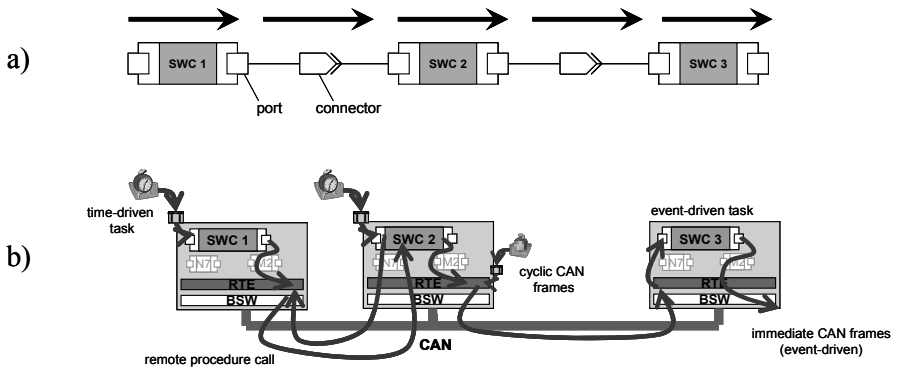


Figure 2 AUTOSAR VBF View and Implementation Mechanisms of "End-to-End Timing Chains"

Software Components and Tasks: At the VFB (virtual function bus) level, AUTOSAR defines so called *software components (SW-Cs)* as atomic entities (see Figure 1 a). However, when it comes to implementation, each SW-C comprises several so called *runnables* that are grouped into *tasks* (see Figure 1 b), which are finally put under operating system control. As a result from OS scheduling, runnables from any SW component can possibly interfere with any other runnable, irrespective of the actual SW-C structure (see Figure 1 c), finally leading to an implementation-dependent timing behavior crossing the SW component boundaries and introducing hidden timing dependencies between seemingly independent SW-Cs at the VFB level.

Logical End-to-End Timing and Interaction Mechanisms: With the increasing distribution of functions over several ECUs in a car, the importance of end-to-end timing (and deadlines) is also increasing. At the VFB level, AUTOSAR captures such *timing chains* composed of communicating *software components (SW-C)* that exchange *signals* via *connectors* and *ports* (see Figure 2 a). At the implementation level (see Figure 2 b), there exist several valid interaction and communication mechanisms including client-server (remote procedure call), periodic sampling including under- and over-sampling, polling, and event-driven. This leads to a variety of indirect *causality chains* that can significantly differ from the "logical causality". In absence of clear execution, communication, and buffering semantics, the timing will change with the implementation, uncontrolled by specification and untestable against test component models that lack the necessary details.

Complex Communication Drivers: A look at the communication driver structure reveals another type of mismatch. The AUTOSAR architecture defines a layered communication stack implementation that includes software and hardware buffers with different access strategies. Furthermore, AUTOSAR communication knows several *frame transmission modes (sporadic, periodic, and mixed)* and *signal transfer properties (triggered, pending)*. Despite their enormous influence on communication timing, these details are hardly visible at the AUTOSAR VFB level.

4 Methodology and Supply-Chain Issues

In addition to the formal definitions of a clear and intuitive timing model, designers also need a methodology to utilize that model in the established design flow. In order to deploy a reliable integration process, the supply-chain communications among all

involved will have to evolve, most likely by establishing timing contracts between OEMs and suppliers. This means that not only a model for annotating timing properties to AUTOSAR elements is needed, but also mechanisms for distinguishing between a property, a guarantee, a requirement, a constraint, etc...

Furthermore, the scope of exchanged information is likely to change. Even though OEMs do not develop large parts of the software, they are responsible for the network in the center of integrations. The network timing, however, depends as well on protocol parameters such as CAN Id and FlexRay slot number as on ECU driver hardware and software (SW-Cs and COM stack), which is –today– mostly out of the OEM's control. In the future, such information must become part of the supply-chain communication. In turn, the Tier-1 suppliers have to cope with more and more software that is supplied externally as object code; examples include OEM-specific functions or basic software (BSW). The challenge is to design the ECU to meet all timing and performance requirements without knowing details of the supplied software [Ri07b].

Based on the feedback we –as a tool provider– are receiving from our customers, future supply-chain communications must have some key properties in order to be accepted:

- Responsibilities and scope must be clearly defined, and must match the established roles of suppliers and OEMs.
- IP protection must be ensured, in particular on the supplier's side. Together with already existing standards like AUTOSAR, this will have a dominant impact on the abstraction of a timing model.
- A comprehensive and reliable timing verification methodology must be in place, since there is no point in modeling something that cannot be analyzed.
- It must be clarified what kind of analysis results and what level of accuracy can be obtained, and the required effort. Full accuracy might not be needed, if only the results are significantly better than overly simplified spread sheets or „gut feeling".

While the OEMs and Tier-1s have certain very relevant requirements, the tool-suppliers (and academia) are the technology providers. We have successfully shown with several customers [He05, Je08] that timing modeling and analysis is possible with today's standards such as OSEK/VDX, and can be introduced in AUTOSAR. All parties together have a good chance to find reasonable timing modeling solutions for AUTOSAR but it is important to not expect the one and only solution. Timing challenges likely differ among domains. Control functions in the chassis domain are typical real-time applications where end-to-end deadlines can be found. Other domains such as infotainment might be more concerned with throughput and quality-of-service. Hence, there might be different types of models that suffice for one or the other domain.

5 Status and Recent Progress

The bad news is that neither the currently published AUTOSAR standard version 2.1 contains enough relevant aspects of timing and performance, nor will the next version 3.0 do. The good news is that two activities have started recently with significant industrial participation to tackle this situation. Since 2007, several OEMs and Tier-1s discuss timing modeling options with selected researchers and tool vendors in the EU-funded TIMMO project [Je07]. TIMMO's goal is defining a timing augmented description language (TADL) for software component models such as AUTOSAR, along

with an appropriate methodology. Requirements are derived from scenarios that the industry partners (TADL users) deliver, while the technology partners contribute conceptual input. Bridging the gap between the API-centric software-engineering view and a math-centric timing analysis view is one key challenge, not only for the definition of the TADL but also to keep the discussion focused within TIMMO.

Meanwhile, also a new timing subgroup within AUTOSAR has been formed to introduce an initial set of core timing parameters into one of the next AUTOSAR releases [SR08], most likely focusing on the implementation view. Both activities tackle the challenges outlined in this paper but –due to their different timelines– with slightly different ambitions with respect to soundness, completeness, etc. AUTOSAR ends 2009, before TIMMO's results are finalized. The partners of both projects have committed to keep the work aligned.

6 Conclusion

AUTOSAR has introduced standardized APIs to ease system integration and enable new supplier-chain processes and business models (software as a product). However, not having a systematic timing modeling and analysis procedure in place is currently challenging key AUTOSAR goals. Several activities to add a "timing view" to the AUTOSAR standard have started recently, which is not all trivial. We have outlined several technical and non-technical challenges in the paper.

A sound solution must tackle modeling and analysis aspects as well as methodology and tools. Few tool vendors, typically those with an academic backing or background, already have parts of the solution available, and their participation in TIMMO and AUTOSAR is inevitable. First internal results indicate that the involved parties approach each other within a bounded scope of technical problems, clear goals, and willing to compromise because there will be no single solution that fits all. First results can be expected in AUTOSAR version 4.0.

7 References

[He05] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst. System Level Performance Analysis - the SymTA/S Approach. In IEE Proceedings Computers and Digital Techniques, Vol. 152, Is. 2, March 2005.

[He06] H. Heinecke, J. Bielefeld, K.-P. Schnelle, N. Maldener, H. Fennel, O. Weis, T. Weber, J. Ruh, L. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, T. Scharnhorst, B. Kunkel. AUTOSAR—Current results and preparations for exploitation. In Proc. 7th EUROFORUM "Software in the vehicle". Stuttgart, Germany, May 2006

[Je07] M. Jersak et.al. Timing-Modell und Methodik für AUTOSAR. Elektronik automotive, Special issue on "AUTOSAR", October 2007

[Je08] M. Jersak. New kid on the block: Scheduling Analysis improves quality and reliability of ECUs, Busses and Networks. In Proc. Embedded World Conference, Nürnberg, Germany, February 2008.

[OS04] OSEK/VDX Communication. v.3.0.3, OSEK/VDX Consortium, July 2004

[OS05] OSEK/VDX Operating System. V.2.2.3, OSEK/VDX Consortium, February 2005

[Ri07a] K. Richter, On the Complexity of Adding Real-Time Properties to the AUTOSAR Software Component Model. In Proc. OMER4 Workshop. Paderborn, Germany, October 2007

[Ri07b] K. Richter. How OEMs can get Suppliers On Board for Designing Extensible Networks. In Proc. Embedded World Conference, Nürnberg, Germany, February 2007.

[SR08] O. Scheickl and M. Rudorfer. Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification. In Proc. Embedded Real-Time Software Congress (ERTS), Toulouse, France, 2008

The legacy ECU software problem - approach and research challenges

Thomas Heinz, Jörn Schneider
Robert Bosch GmbH, Stuttgart
{*thomas.heinz2, joern.schneider*}*@de.bosch.com*

Abstract: While life cycles of electronic components tend to become ever shorter, automotive suppliers need to keep up ECU supply for up to 30 years. This causes high storage costs or requires redesigning legacy ECUs in a way that obsolete components are replaced by newer ones. These redesigns may cause tremendous software redevelopment efforts, e.g. to guarantee correct real-time behaviour. This paper presents a both academically and industrially challenging approach inspired by the legacy ECU software problem, namely static binary translation, which aims at automatic retargeting of real-time software. Eventually, the approach has the potential to significantly cut the redevelopment costs and facilitate the interchangeability of electronic components.

1 The legacy ECU software problem

Automotive ECUs are typically supported over a period of 10 to 30 years. This timeframe exceeds the life cycles of many of its electronic components (e.g. microcontrollers) by far. To enable production of ECUs in the original configuration beyond the point of discontinuation of its electronic components, these components need to be stockpiled in advance. However, it is very hard to come up with a tight estimate of the actual number of ECUs needed within the entire product lifetime. Hence, it can be necessary to redesign the ECU hardware such that obsolete components are replaced by up-to-date ones with similar functionality. This usually makes it necessary to port the software as well.

Porting and testing for correct functionality and real-time behaviour is tedious, very costly and comes with the significant risk of introducing errors. This inspires a both academically and industrially challenging approach, namely **automatic software retargeting** which is discussed in the next section.

2 Automatic retargeting of embedded real-time software

The legacy software problem is not specific to ECU software but is prevalent in virtually any real-time and non-real-time software field. A successful approach for non-real-time and soft real-time software is **emulation**, i.e. representing the original programming environment (e.g. instruction set architecture, device registers, application binary interface) on the target machine and instruction-wise interpreting the original machine code. Ideally, emulation is a black box approach in the sense that it does not require any knowledge about the legacy code. It is highly flexible but inherently inefficient which can be compensated by translating software such that it can be natively executed on the target machine. An es-

tablished method that builds upon this idea is **binary translation**, a compilation technique that transforms machine code for one architecture into machine code for another architecture while retaining its functional behaviour. Binary translation can be performed either at runtime (dynamic) [CLU02] or offline (static) [CvERL01]. The former takes advantage of runtime information which enables efficient code generation and simplifies some problems of the static approach, e.g. control flow analysis in the presence of branches whose target address depends on runtime data (indirect branches/calls). However, a major issue that renders dynamic binary translation unsuitable for legacy ECU software (or more general software involving hard real-time constraints) is its unpredictable temporal behaviour. This is because the translator is invoked at unpredictable times at runtime and the temporal behaviour of the translated code (= runtime data) cannot be predicted statically. In particular, the dynamic translator is not amenable to existing real-time scheduling methods because its invocation cannot be postponed (highest priority) and its minimum interarrival time cannot be reasonably bounded. A major advantage of the static approach is its amenability to program analysis as the target code is available offline.

A solution to automatic retargeting of embedded real-time software must ensure **preservation of functional and temporal behaviour** of the original software. The following sections discuss the suitability of **static binary translation (SBT)** with respect to these requirements and present promising approaches and open challenges.

3 Static binary translation - a functional point of view

Legacy ECU software is composed of application software and basic software (OS, drivers). Apart from the standardized operating system (OSEK), ECU software is proprietary and contains a considerable amount of architecture dependent low level code. Hence, SBT must address the entire instruction set architecture and computational resources (e.g. registers, CPU flags, memory, I/O ports, interrupts) of the source machine. Moreover, the functional behaviour of the devices on the source machine (e.g. SPI, CAN, ADC, timer) must be mapped onto appropriate target devices.

Figure 1 illustrates common transformation steps to translate the source binary into a functionally equivalent target binary. In a first step, the binary is disassembled into code and data sections. At this point, code sections may still be intermixed with data (e.g. switch table, constants). The subsequent control and data flow analysis separates code and data and builds a control flow graph (CFG). To enable architecture independent SBT and optimizations, the source CFG is mapped to an intermediate CFG based on an architecture independent intermediate language. Finally, the code generator turns the intermediate CFG into a target specific CFG on which optimizations tailored to the target architecture can be applied. To obtain the target binary, two simple steps need to be performed, namely translating the target specific CFG into target assembly and finally into machine code. Note that it is possible to transform debugging information associated with the source binary such that the translated binary can be related to the original source code.

SBT faces a number of challenges some of which can be reduced to theoretically undecid-

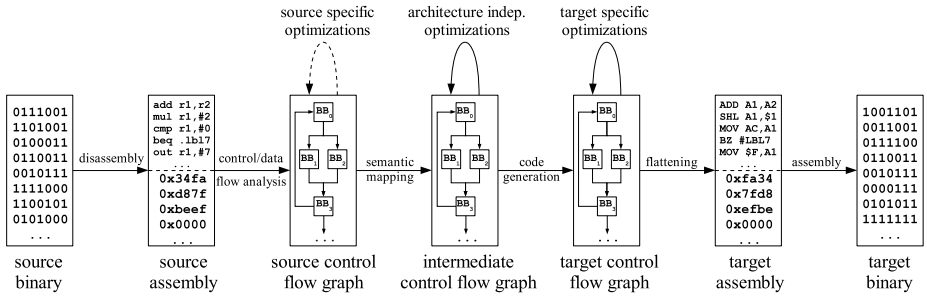


Figure 1: Common SBT transformation steps.

able problems. Among are:

- control flow analysis in the presence of indirect branches/calls
- efficient code generation
- data translation to enable native data access (alignment, data representation)
- separation of intermixed code and data
- handling of machine dependent issues (e.g delayed branches)
- distinguishing memory mapped I/O instructions from regular memory accesses
- instruction atomicity in the presence of hardware interrupts (a single source instruction may correspond to a sequence of target instructions)
- retargetability of the translator

Theoretically undecidable problems are approached by various program analysis techniques which compute safe approximations of runtime information (e.g. interval analysis of register values). Where not automatically deducible, the user has to provide the relevant information, e.g. in terms of source code annotations which are treated by the compiler. Particularly, these analyses are essential for efficient code generation. Note that SBT is not capable of handling self-modifying code. However, this is not a problem for automotive software and presumably neither for hard real-time software in general.

Recent advances in control flow analysis (CFA) [The02] and the success of static timing analysis tools, which face the same problem, show that CFA is reasonably mastered to be successfully applied to real world ECU software.

4 Temporal proximity

When replacing an ECU by its redesigned version, it is of crucial importance that the observable behaviour at its interfaces to sensors, actuators and the car network remains essentially unchanged. This includes the temporal behaviour in particular.

Before one can come up with methods to preserve the temporal behaviour, it is necessary to determine the level of accuracy as a criterion for precision of the preservation. Here, four levels of accuracy are distinguished.

Cycle accuracy: The functional behaviour of each source machine clock cycle is precisely reflected on the target machine. This is required for development of real-time systems in absence of the target hardware. Instead a cycle accurate system simulator is used. Real-time capability cannot be achieved using a fully software-based approach.

Instruction accuracy: The start or end of execution of each source instruction and its corresponding sequence of target instructions is synchronous on source and target machine. This level of accuracy is still too high. Moreover, to achieve real-time capability, the target machine must be vastly superior to the source machine.

Basic block accuracy: The scope is extended from a single instruction to a basic block. This accuracy level might already be too imprecise, e.g. in case a basic block contains an I/O instruction whose execution point in time is crucial to the correct behaviour of the software.

Synchronization point (SP) accuracy: SP accuracy is motivated by the observation that certain instruction sequences can be executed arbitrarily fast without affecting the correctness of the temporal behaviour, e.g. arithmetic computations involving only local variables. Hence, instruction accuracy is not demanded for each program point but only for certain critical program points, e.g. an I/O instruction, a system instruction or a store operation to a shared memory location. A naive way to safely approximate the set of SPs for a program is to divide the instruction set into potentially critical and non-critical instructions and to mark each program point containing a potentially critical instruction as SP. The number of SPs can be reduced by program analysis or user information.

Given a safe approximation of the SPs, SP accuracy is precise enough to capture the temporal behaviour of the original software that is to be preserved. It remains to devise a synchronization method that is real-time capable. Traditional approaches implement a dynamic delay mechanism that keeps track of the execution time on the source machine [Cog95]. The delay ensures that after the execution of an SP on the target machine, the system is idle until the execution of the respective instruction would have been finished on the source machine. This is efficient for simple architectures where the execution time of each instruction is constant but does not scale for machines with pipelines or caches as the execution time of an instruction depends on the execution history and thus all execution time affecting entities would have to be simulated on the target machine up to an extent.

As a solution, we propose an offline method to compute a set of delay constants $\{d^p_{c_1}, \ldots, d^p_{c_n}\}$ for each program point p where each constant is associated with a context c_i. A context allows to distinguish several control flows that may lead to the execution of the associated program point (see [The02] for details). The synchronization code that is executed at runtime mainly consists of determining in which context c_i the program point p is currently executed and waiting the precomputed constant time $d^p_{c_i}$. The problem of computing the delay constants is cast into an optimization problem whose objective is to minimize the maximum **temporal displacement** of all program points. Intuitively, the temporal displacement of a program point p is the maximum time that the execution on the target machine is ahead or behind the execution on the source machine for p. Moreover, the temporal displacement is a metric for the quality of the achieved SP accuracy.

The precise description of the optimization problem is beyond the scope of this paper. The basic idea is motivated by the integer linear program (ILP) used to compute an upper bound of the worst case execution time of real-time tasks [Wil05]. Essentially, the ILP constraints are used to describe a safe approximation of the temporal displacement for each program point, i.e. an upper bound of the actual temporal displacement. For this, an inner approximation of the execution time of each program point on the source machine is computed by measurements whereas on the target machine, an outer approximation is computed based on abstract interpretation. Note that an inner approximation $[s^i, t^i]$ of an execution time interval $[s, t]$ satisfies $[s^i, t^i] \subseteq [s, t]$ and an outer approximation $[s^o, t^o]$ of $[s, t]$ satisfies $[s^o, t^o] \supseteq [s, t]$. The approach is currently being implemented and refined as part of the Ph.D. thesis of the primary author.

Clearly, this approach does not guarantee perfect SP accuracy but it provides a guaranteed bound of the temporal displacement for each program point and thus quantifies the degree of **temporal proximity** between the execution of the original and the translated code.

5 Conclusion

The legacy ECU software problem motivates a number of challenging theoretical and practical problems. The approach presented in this paper seems very promising. However, as the temporal behaviour of the original software cannot be precisely reflected on the target machine, the deviation with respect to the temporal behaviour which can be tolerated must be quantified such that it is possible to decide whether correct real-time behaviour is achieved for the given temporal displacement. Note that a quantification of the tolerance cannot be deduced from the code but requires system knowledge, e.g. about the physical tolerance of the system.

Last not least, it remains to show that SBT is real-time capable for a reasonable pair of source and target CPU/microcontroller and thus prove its practicability. For this, it is essential to devise a method to safely and tightly approximate the necessary SPs.

References

[CLU02] Cristina Cifuentes, Brian T. Lewis, and David Ung. Walkabout - A Retargetable Dynamic Binary Translation Framework. Technical report, Sun microsystems, 2002.

[Cog95] Bryce Howard Cogswell. *Timing insensitive binary-to-binary translation*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, April 1995.

[CvERL01] Christina Cifuentes, Mike van Emmerik, Norman Ramsey, and Brian Lewis. *The University of Queensland Binary Translator (UQBT) Framework*. The University of Queensland, Sun Microsystems, Inc, 2001.

[The02] Henrik Theiling. *Control Flow Graphs For Real-Time Systems Analysis*. PhD thesis, Universität des Saarlandes, 2002.

[Wil05] Reinhard Wilhelm. Determining Bounds on Execution Times. In R. Zurawski, editor, *Handbook on Embedded Systems*, pages 14–1,14–23. CRC Press, 2005.

BRIEF: Systematisches Prozess-Briefing und –Debriefing. Ein bei der Erstellung komplexer Automobilsoftware praxiserprobtes Entwicklungsvorgehen, das Elemente klassischer und agiler Modelle kombiniert.

Hannes Omasreiter[1], Claudia Schlumpberger[1], Lorenz Widmann[2]

[1]Software Prozessgestaltung, Daimler AG
Wilhelm-Runge-Str., 89081 Ulm
{hannes.omasreiter, claudia.schlumpberger}@daimler.com

[2]Prozesse und Messverfahren, Fraunhofer IESE
Fraunhofer-Platz 1, 67663 Kaiserslautern
lorenz.widmann@iese.fraunhofer.de

Abstract: Wir skizzieren einen Entwicklungsansatz, der seit einigen Jahren als roter Faden zur Entwicklung von Motorsteuergeräte-Software für Mercedes-Benz-PKW eingesetzt wird. Der Ansatz kombiniert Elemente existierender Entwicklungsmodelle mit projektspezifischen Praxiserfahrungen.

1 Einleitung

Der Erfolg von (Software-)Organisationen und Projekten wird durch die drei Faktoren Mensch, Prozess und Technik bestimmt. Die genannte Reihenfolge der Faktoren spiegelt dabei unserer Erfahrung nach deren Wichtigkeit wider. Dass der Erfolg von Projekten oft von weichen Faktoren abhängt, belegen auch zahlreiche Studien [St06]. Menschen sind für Produktinnovationen verantwortlich und damit für die Basis des Unternehmenserfolgs. Außerdem legen sie die beiden anderen Faktoren fest. Prozesse tragen als Bindeglied zwischen Mensch und Technik ebenfalls erheblich zum Projekt- und Unternehmenserfolg bei. Sie bestimmen wesentlich Effizienz und Qualität und damit entscheidend die Wirtschaftlichkeit von Projekten [BK95]. Die Bedeutung von Prozessen ist dabei umso höher, je größer und komplexer die Projekte sind. Der Faktor Technik hat unserer Ansicht nach den geringsten Einfluss auf den Unternehmens- und Projekterfolg. Dies bedeutet nicht, dass wir den Faktor nicht ebenfalls als sehr bedeutend einstufen. Vielmehr sind wir der Auffassung, dass die technische Ebene (z.T. auch die Prozessebene, vgl. etwa [He06]) in der Regel – insbesondere von der wissenschaftlichen Seite her – sehr gut bedient wird. Während also für den Einfluss auf Projekterfolge die Reihenfolge Mensch, Prozess und Technik gilt, entsteht bei der Lektüre wissenschaftlicher Publikationen häufig der Eindruck, dass für Forschungsaufwände eine umgekehrte Reihenfolge gilt. Der Grund dafür liegt vermutlich darin, dass Forschungsfragen umso konkreter und damit attraktiver sind, je näher sie dem technischen Faktor kommen.

Außerdem sind Fragestellungen der menschlichen Ebene meist wissenschaftlich schwer greifbar. Dazu kommt, dass Problemstellungen auf Prozess- und menschlicher Ebene oft erst durch eigene intensive Projekterfahrung zugänglich sind und Lösungen häufig fachübergreifendes Wissen und Forschen erfordern. Aus Unternehmenssicht besteht also ein hoher Bedarf an praxistauglichen Konzepten, die Mensch, Prozess und Technik geeignet gewichten. In diesem Paper stellen wir einen Software-Entwicklungs-Ansatz vor, der diesen Anspruch hat. Der von uns als „systematisches Prozess-Briefing und –Debriefing" (kurz: BRIEF) titulierte Ansatz verwendet Elemente unterschiedlicher existierender Entwicklungs-Modelle und -techniken, nämlich u.a. folgender: Wasserfall-Modell, Spiral-Modell, V-Modell XT, Issue-Based-SW-Engineering, agile Methoden (v.a. Scrum), kontinuierliche Prozessverbesserung (KVP), Produktlinienentwicklung. Der BRIEF-Ansatz wurde von uns für die Entwicklung von Motorsteuergeräte-Software bei Mercedes-Benz entwickelt und ist dort seit mehreren Jahren im Einsatz. In den vergangenen beiden Jahren konnte damit u.a. die Entwicklung von komplexen Mildhybrid-Systemen schnell und effizient unterstützt werden. Der Ansatz stellt sowohl aus unserer als auch aus Anwendersicht ein gelungenes Beispiel dafür dar, wie auf Basis technisch und (zwangsläufig) generisch ausgerichteter Modelle aus der Wissenschaft in Kombination mit den Erfahrungen und Herausforderungen in einem Unternehmen eine spezifische praxistaugliche Lösung entstehen kann.

2 Herausforderungen und Grundideen des BRIEF-Ansatzes

Die Herausforderungen bei der Entwicklung von eingebetteter Kfz-Software sind vielfältig. Durch die zunehmende Komplexität und den steigenden Zeitdruck ist es dabei – neben zahlreichen technischen Zielen – von großer Bedeutung, den roten Faden in einem Projekt zu behalten. Den Entwicklern und dem Management soll nicht nur die *technische Ebene* bekannt sein, die vorgibt, wie konkrete Aufgaben erledigt werden können. Sondern es muss auch die *Prozessebene* transparent gemacht werden, die einen Überblick über aktuell anstehende Aufgaben, deren Abarbeitungsstatus und Relevanz im Projekt sowie der dabei beteiligten Personen gibt. Ein Software-Entwicklungsmodell kann Fragen dieser Prozess-Ebene lösen. Für unseren konkreten Anwendungszweck, nämlich die Prozess-Unterstützung für Entwickler von Motorsteuergeräte-Software auf operativer Ebene in einem extrem schwierigen Umfeld (hohe Komplexität, hoher Zeitdruck, hoher Innovationsgrad, zahlreiche neue Mitarbeiter), erfüllten jedoch vorhandene Vorgehensmodelle nicht alle gegebenen Anforderungen v.a. auf der *menschlichen Ebene* (z.B. operative Einbindung aller Entwickler bei möglichst geringem Aufwand für Prozess-Schulungen). Wir haben daher aus Elementen unterschiedlicher klassischer und agiler Entwicklungsmodelle den BRIEF-Ansatz erstellt. Dieser basiert auf folgenden Grundideen:

- Als Basis dient ein einfaches, checklistenbasiertes Vorgehensmodell, das sich bei der Dokumentation an Wasserfallmodell, Spiralmodell, V-Modell und Issue-Based-Modell orientiert.
- Bei der konkreten Anwendung des Modells rücken agile Methoden in den Vordergrund. Anwendungskern ist eine regelmäßige Briefing/Debriefing-Sitzung, die auf Ideen von Scrum und KVP basiert.

- Mit Methoden des Variabilitätsmanagements wird sichergestellt, dass einerseits projektübergreifend möglichst einheitliche Prozesse gelebt werden und andererseits eine projektspezifische Anpassung der Entwicklungsprozesse möglich ist.

3 Artefakte des BRIEF-Ansatzes

Die Dokumentation des Prozesswissens erfolgt beim BRIEF-Modell auf eher klassische Art und Weise. Der Entwicklungsprozess ist in Meilensteine (MS) unterteilt, wobei an jedem MS eine Checkliste (vgl. Abb. 1) hinterlegt ist. Die Checklisten beinhalten Checkfragen mit wichtigen Aktivitäten zur Erreichung des MS, denen u.a. ein Status zugeordnet ist (z.B. „offen“, „erledigt“). Die Checklisten werden wie beim Wasserfallmodell der Reihe nach abgearbeitet. Die Reihenfolge ist aber nur ein Vorschlag, d.h. das Modell kann – ähnlich wie ein Issue-basiertes Modell – als ein Pool von Checkfragen angesehen werden. Der Status der Checkfragen kann – unabhängig von deren Reihenfolge – verändert werden. Kernelemente der MS-Liste sind mehrere Entwicklungs-Zyklen, die sich am V-Modell orientieren. Die mehrmalige Wiederholung dieser Zyklen basiert auf Ideen des Spiral-Modells. Über Attribute können Checklisten und Checkfragen klassifiziert werden, z.B. danach, ob sie für Neuentwicklungs-Projekte gelten oder nicht. Auf dieser Basis lassen sich die Prozessinhalte projektspezifisch anpassen. Aus Benutzerakzeptanzgründen ist die Dokumentation bewusst einfach, klar und übersichtlich gehalten. Die Inhalte der Checklisten werden in Zusammenarbeit mit den Projektbeteiligten erarbeitet.

Abbildung 1: Ausschnitt einer Checkliste

Abhängigkeiten zwischen Prozessschritten sind bewusst nicht durchgängig und formal abgebildet, sondern nur in wichtigen Fällen textuell beschrieben. Somit war die Einführung eines eigenen Prozesstools nicht notwendig. Dadurch und durch die übersichtliche Strukturierung ist die Einarbeitungszeit extrem kurz.

4 Prozesse des Brief-Ansatzes

Für die Prozesse des BRIEF-Ansatzes sind zunächst drei Ebenen mit ihren zugehörigen Rollen wichtig: Die organisatorische Ebene mit der Rolle des organisationsweiten Koordinators (OK), die Projektebene mit der Rolle des Projekt-Koordinators (PK) und

die Entwicklungs-Ebene mit der Rolle des Prozess-Anwenders (PA). Diese Rollen sind hauptsächlich über (mündliche, nichtformale) Briefing/Debriefing-Beziehungen miteinander verbunden (vgl. Abb. 2). Die organisatorische Ebene ist dafür zuständig, für die betrachtete Organisationseinheit projektübergreifend die Dokumentation des Prozesswissens sicherzustellen, dieses in die Projekte zu bringen und es laufend aus den Projekten heraus zu ergänzen und zu aktualisieren. Die Projektebene fungiert als Bindeglied zwischen organisatorischer und Entwicklungs-Ebene. Die Entwicklungsebene führt Prozessvorgaben aus und dient als Quelle für neues Prozesswissen.

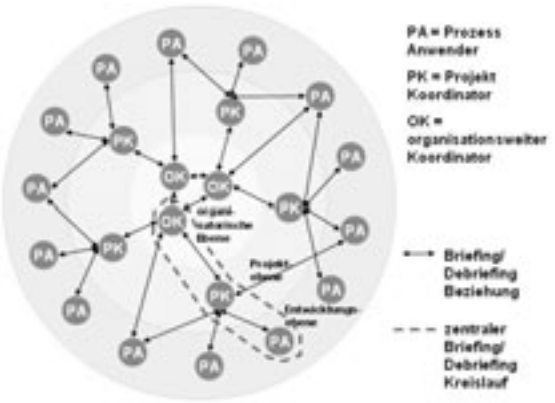


Abbildung 2: Briefing/Debriefing-Netz einer Organisationseinheit

Kernstück der Prozesse zur Anwendung des BRIEF-Ansatzes ist das BRIEF-Meeting, das in jedem Projekt regelmäßig (z.B. alle 6 Wochen) stattfindet und den zentralen Briefing/Debriefing-Kreislauf in Gang bringt. Am Meeting für ein Projekt sind mindestens der PK und ein OK beteiligt. OK und PK machen zunächst eine *retrospektive* Betrachtung des Projekts (bis zum Zeitpunkt des vergangenen Meetings). Dazu gehen sie insbesondere die checklistenbasierte Prozessbeschreibung des Projekts durch, aktualisieren z.B. Zustandswerte der Checkfragen und diskutieren ggf. aufgetretene Probleme. Der OK dokumentiert ggf. lessons learned und Verbesserungsvorschläge des PK zur Prozessbeschreibung. Im zweiten Teil des BRIEF-Meetings machen OK und PK eine *prospektive* Betrachtung der (mindestens bis zum nächsten Meeting) kommenden Prozessschritte. Der OK erklärt dem PK dabei evtl. unklare Aufgaben und gibt Ratschläge zur Durchführung oder nimmt Fragen zur Klärung mit. Der PK gibt Hinweise auf projektspezifische Gegebenheiten und Notwendigkeiten zur Ergänzung oder Anpassung der Checkfragen, die vom OK protokolliert werden.

Im Nachgang des Meetings klärt der OK ggf. Fragen des PK und aktualisiert anhand dessen Input das Prozesswissen zentral für alle Projekte. Der PK leitet die Prozess-Aufgaben an die zuständigen PA in seinem Projekt weiter und kontrolliert deren Ausführung. Die PA erhalten dadurch alle für sie aktuell und spezifisch wichtigen Informationen zu Prozessaktivitäten, ohne sich z.B. mit einem Prozesstool beschäftigen zu müssen. Probleme oder Verbesserungsvorschläge zum Prozess melden sie ihrem PK zurück, der dies wiederum ins nächste Meeting mitnimmt.

Auf diese Weise bildet sich ein Briefing/Debriefing-Kreislauf vom OK über den PK zu einem PA und vom PA wieder zurück über den PK zum OK. Der Kreislauf stellt sicher, dass die PA effizient über ihre Prozess-Aufgaben informiert werden und gleichzeitig die Prozesse laufend aus allen Projekten und für alle Projekte verbessert werden.

Das beschriebene Vorgehen des BRIEF-Ansatzes funktioniert aus unserer Sicht v.a. deswegen in der Praxis, weil es eine Reihe von Elementen einsetzt, die auf der menschlichen Ebene angesiedelt sind und so vermehrte Akzeptanz schaffen, z.B.:

- Inkrementelles Reviewen, Anpassen und Schulen der Prozesse
- Das Prozesswissen kommt zum Großteil nicht von außen, sondern von den Entwicklern, Input ist jederzeit möglich, das Wissen wird laufend aktualisiert
- Entwickler können prozesskonform arbeiten, benötigen dafür aber möglichst wenig Aufwand und Zeit (z.B. Dokumentation wird ihnen komplett abgenommen)
- Regelmäßig eingeplante Meetings verhindern, dass Prozessthematiken im Projektalltag untergehen

5 Fazit

An einem Beispiel haben wir skizziert, wie die Lösung einer praktischen Herausforderung bei der Entwicklung von eingebetteter Kfz-Software durch eine Kombination von wissenschaftlichen Modellen und praxisgetriebenen Anpassungen aussehen kann. Nicht zuletzt wegen der hohen Relevanz der menschlichen Ebene und der Prozess-Ebene auf den Erfolg einer Organisation befürworten wir eine stärkere wissenschaftliche Forschung auf diesem Gebiet. Notwendig dafür ist eine stärkere interdisziplinäre Zusammenarbeit, besonders aber auch ein intensiver Einblick in die Gegebenheiten der industriellen Praxis. Da sich reale Projekte nicht als Experimentierfeld eignen, schlagen wir vor, etwa an Universitäten verstärkt reale Projekte zu simulieren. Die industrielle Seite sollte dabei helfen, möglichst reale Bedingungen herzustellen, die universitäre Seite hätte Gelegenheit, Modelle selbst einzusetzen und praxisnah zu entwickeln. Das „Global Studio Project“ der Firma Siemens mit mehreren Universitäten ist ein gutes Beispiel, das in diese Richtung geht. Ein weiterer Ausbau solcher Projekte, evtl. auch universitäts- und firmenübergreifend, würde sicherlich auch für Studenten die Ausbildung noch spannender machen und sie noch besser auf Tätigkeiten in der Praxis vorbereiten.

Literaturverzeichnis

[St06] Steeger, O.: “Hidden Champions”: Was unterscheidet Spitzenprojekte von Millionenpleiten?, www.pmaktuell.org, 2006

[BK95] Butler, Kelley L. "The Economic Benefits of Software Process Improvement," CrossTalk, 7/95, pp. 14-17

[He06] Heidrich, J., et al.: "People-oriented Capture, Display, and Use of Process Information", In: New Trends in Software Process Modeling, Series on Software Engineering and Knowledge Engineering, Vol. 18, World Scientific Publishing Company, 2006

Vom modellbasierten Architekturentwurf über die Simulation verteilter Systeme zum Serienprojekt

Rainer Otterbach, Oliver Niggemann

dSPACE GmbH
Technologiepark 25
33100 Paderborn
ROtterbach@dspace.de
ONiggemann@dspace.de

Abstract: Methods and tools in line with automotive requirements extend model-based development from function to system and software architecture level. Executable system models allow the interacting software components of single and networked control units to be verified by PC-based simulation in early design steps. Such models comprise the hardware topology, communication buses, and basic software effects for instance on scheduling and timing. Existing control design tools are included to define the functional behaviour of software components. Production-ready application code and the run-time environment are generated with established autocode tools.

1 Ein Prozessmodell für den automotiven Software-Entwurf

Die modellbasierte Entwicklung der Steuergeräte-Software und die automatische Generierung des Seriencodes haben sich weithin etabliert, zumindest auf Einzelfunktionsebene. Die Einbeziehung der Software-Architektur steckt dagegen noch in den Kinderschuhen, obwohl gerade hier der Schlüssel zur Bewältigung der zunehmenden Komplexität zu finden ist. Ein Grund liegt darin, dass bisher Methoden und Werkzeuge für die Modellierung und erst recht für die Simulation von automotiven Software-Architekturen fehlten. Sie können nicht einfach aus anderen Anwendungsgebieten, zum Beispiel der Telekommunikation, übernommen werden, wie zahlreiche Versuche etwa auf der Basis von UML gezeigt haben. Ähnliches gilt für die heute bekannten, modellbasierten Entwurfswerkzeuge, die sehr gut die Sicht des Funktionsentwicklers widerspiegeln, aber nicht geeignet sind, Software-Architekturen im großen Stil darzustellen.

Methoden und Tools für den Software-Architekturentwurf im automotiven Umfeld müssen folgende Randbedingungen erfüllen:

- Verschiedene Sichten sind in einem einheitlichen Systemmodell darstellbar. Dies erweitert die Funktionsmodellierung und umfasst sowohl die Software-Architektur eines einzelnen Steuergeräts als auch die Systemarchitektur eines Steuergeräteverbunds bis hin zur gesamten E/E-Architektur eines Fahrzeugs.

- Typische Gegebenheiten wie die auf Hersteller und Zulieferer verteilte Entwicklung werden ebenso unterstützt wie automotive Standards (AUTOSAR®, DBC, FIBEX, ASAP2 usw.). Zu beachten ist, dass der Entwurf einer Software-Architektur mehr ist als das Editieren beispielsweise AUTOSAR-konformer Beschreibungsdateien.
- Kommunikationsbusse (CAN, LIN, FlexRay) und ihre Konfigurationen, Betriebssysteme (zum Beispiel AUTOSAR OS) und andere Basis-Software-Module (Mode-Management, Diagnose, NV-RAM-Manager, Kommunikationsstack etc.) sind mit einfachen Mitteln beschreibbar. Dies gilt auch für das Zeitverhalten der Einzelkomponenten und des Gesamtsystems.
- Die im automotiven Bereich etablierten Werkzeuge für den Funktionsentwurf, zum Beispiel MATLAB®/Simulink®, lassen sich nahtlos integrieren.
- Die Simulation auf allen Entwurfsebenen einschließlich der Architektur-Ebene erlaubt die frühzeitige Verifikation des Systems.
- Der direkte Weg ins Steuergerät über die automatische Seriencode-Generierung der Funktionssoftware und der Laufzeitumgebung sowie die Integration vorhandener Software-Komponenten ist gegeben. Dies beinhaltet die Generierung oder Konfiguration von Basis-Software-Schichten einschließlich der Bus-Kommunikation.

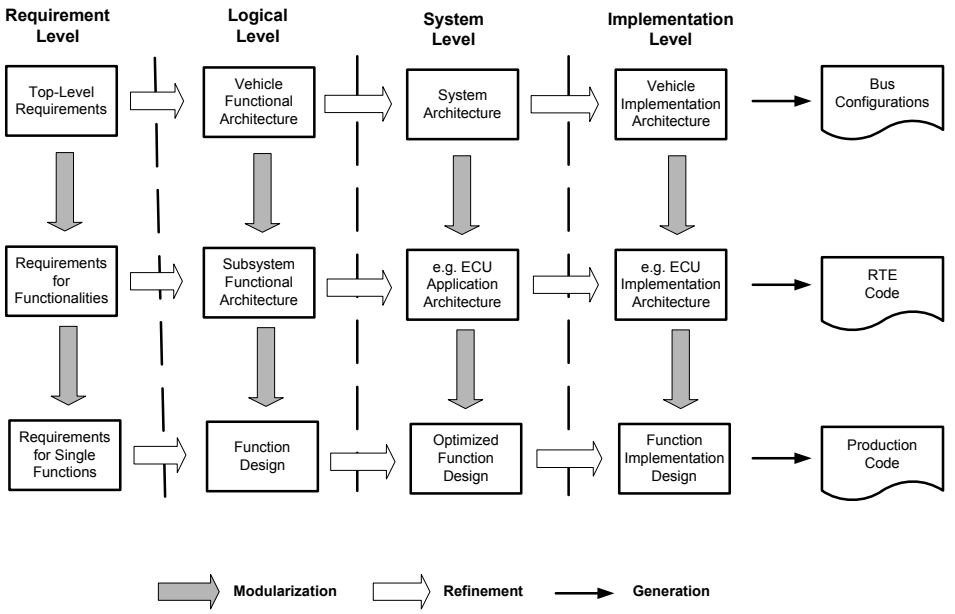


Abbildung 1: Prozessrahmen für den automotiven Software-Entwurf

Modelle werden in drei Kategorien eingeordnet, die grob durch Prozessschritte in der E/E-Entwicklung motiviert sind und sich durch Verfeinerung aus den Anforderungen ergeben (Abbildung 1, horizontale Richtung). Anforderungen liegen als informelle Textdokumente oder frühzeitig entwickelte Modelle vor, zum Beispiel kann eine Lichtsteuerung relativ einfach mit Hilfe eines endlichen Automaten definiert werden. Die Modularisierung (vertikale Richtung) ist für den Umgang mit komplexen Systemen unerlässlich und dient der Arbeitsteilung zwischen verschiedenen Entwicklern. Da je nach Modell-

kategorie (horizontale Richtung) unterschiedliche Effekte simuliert werden können, ist die Prozessdarstellung in Abbildung 1 geeignet, die durch eine PC-Simulation identifizierbaren Systemfehler den Phasen des Entwicklungsprozesses zuzuordnen und das Potential für eine Vorverlagerung von Testaufgaben zu erkennen [Ott07].

2 Entwurf und Simulation von Funktionsnetzwerken

Auf **logischer Ebene** wird das System als hierarchisches Modell vernetzter Funktionalitäten oder Software-Komponenten spezifiziert. Solche logischen Software-Architekturen werden häufig von Herstellern in frühen Phasen verwendet, um die Modularisierung und Verteilung von Fahrzeugfunktionen zu entwerfen, wiederverwendbare Software-Komponenten zu spezifizieren und Software-Schnittstellen mit Zulieferern zu vereinbaren. Auswirkungen der späteren Hardware-Plattform werden dabei nicht berücksichtigt. Abbildung 2 zeigt ein Beispiel für die logische Software-Architektur einer Blinkerregelung (hier mit dem Systemmodellierungs-Werkzeug SystemDesk von dSPACE erstellt).

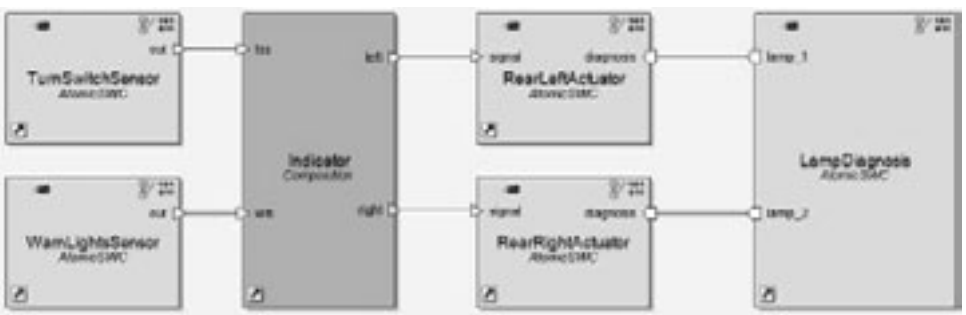


Abbildung 2: Funktionsnetz: Verschaltung von Software Komponenten auf logischer Ebene

Durch das Hinzufügen zumindest grober Verhaltensmodelle wird ein Funktionsnetz simulierbar. Dazu werden in frühen Phasen der Entwicklung entweder ausführbare Spezifikationsmodelle herangezogen, zum Beispiel in Simulink, oder es werden existierende Software-Komponenten aus früheren Projekten verwendet. Der Zweck der Modelle ist zu diesem Zeitpunkt nicht die Implementierung des Seriencodes, sondern die Spezifikation allgemeiner Eigenschaften des Komponentenverhaltens, oft durch den Fahrzeughersteller. Sie dienen dem Zulieferer als Basis für die serienreife Implementierung. Zu einem späteren Zeitpunkt kann man dann mit verfeinerten Modellen, die die Implementierungsdetails beinhalten, oder mit codierten Software-Komponenten auf die logische Ebene zurückkehren, um das funktionale Systemverhalten erneut zu verifizieren. Die Simulation auf logischer Ebene findet in erster Linie Fehler im Zusammenspiel vernetzter Komponenten, die in der gegenseitigen Abhängigkeit der Funktionen begründet sind. Je mehr das gewünschte Systemverhalten durch verteilte Funktionen realisiert wird, desto mehr lohnt sich die frühe Simulation. Beispiele dafür findet man im gesamten Bereich der Komfortelektronik, aber auch bei Fahrerassistenzsystemen wie adaptive Fahrgeschwindigkeitsregelung, Aktivlenkung oder Spurhalteunterstützung.

3 Verteilung auf Hardware und Simulation der Kommunikation

Modelle auf **Systemebene** definieren die Verteilung von Funktionen auf Steuergeräte und die Software-Struktur der Anwendungsebene auf einem einzelnen Steuergerät. Damit wird unter anderem die Planung und Erstellung der Kommunikationsmatrizen aus der Gesamtsicht der E/E-Architektur des Fahrzeugs ermöglicht und die Arbeitsteilung zwischen Fahrzeughersteller und Zulieferern unterstützt. Schnittstellen zwischen Funktionalitäten werden detaillierter definiert, zum Beispiel durch das Hinzufügen von Datentypen zu Signalen, die zuvor nur durch einfache Bezeichner definiert wurden. Später erlauben die so definierten Schnittstellen die Integration verteilt entwickelter Software-Module. Beispielsweise können vom Hersteller selbst entwickelte Software-Komponenten durch den Zulieferer in einem Steuergerät integriert werden. Die Software-Komponenten und ihre externen Schnittstellen werden auf eine Hardware-Topologie abgebildet, die die bestehenden Steuergeräte und deren Recheneinheiten, die Aktoren und Sensoren sowie Kommunikationsbusse umfasst.

In der Simulation können nun weitere Effekte dargestellt werden, insbesondere solche, die von den Kommunikationsbussen verursacht werden. Bei CAN werden Auswirkungen der Arbitrierung oder der in der Realität immer begrenzten Buskapazität simuliert. Dies erlaubt eine grobe Abschätzung der Busauslastung und die Berücksichtigung von Kommunikationsverzögerungen bei der Analyse des zeitlichen Verhaltens.

4 Generierung, Simulation und Integration des Seriencodes

Die **Implementierungsebene** ist die Basis für die Erstellung des Steuergeräte-Seriencodes. Das Systemmodell enthält jetzt alle Informationen, die für die Implementierung der Funktionen notwendig sind, wie Angaben zur Festkomma-Arithmetik oder Abhängigkeiten zum Mode-Management. Darüber hinaus erfolgt die Konfiguration der Basis-Software-Module auf den Steuergeräten. Hierzu gehört die Definition der Tasks, die Abbildung von Software-Komponenten auf Tasks, die Konfiguration des Kommunikationsstacks und der I/O-Treiber. Um die Wiederverwendbarkeit von Software-Komponenten sicherzustellen, muss der C-Code einer Komponente von der Hardware-Plattform unabhängig sein und standardisierte Schnittstellen für die Kommunikation zu anderen Komponenten aufweisen. Auf Basis des AUTOSAR-Standards lässt sich diese Forderung erfüllen [AUT] [St07].

Nach AUTOSAR sind Software-Komponenten in so genannte Runnables untergliedert. Das sind C-Funktionen, die später vom Betriebssystem aufgerufen werden. Sie werden mit einem Seriencode-Generator wie TargetLink oder manuell in C-Code implementiert. TargetLink erlaubt die zuverlässige Umsetzung von Simulink/Stateflow®-Modellen in hocheffizienten Code und unterstützt den AUTOSAR-Standard. Für den Datenaustausch zwischen den Komponenten und die Verbindung zur Basis-Software wird die so genannte AUTOSAR Runtime-Environment (RTE) generiert.

Die Implementierungsebene bietet naturgemäß die weitreichendsten Simulationsmöglichkeiten, da hier wichtige Parameter der Steuergeräte bekannt sind und der Seriencode

verfügbar ist. Einer der zentralen Aspekte ist die Nachbildung des zeitlichen Systemverhaltens. SystemDesk verwendet eine OSEK-Betriebssystemsimulation auf dem PC, um das Scheduling des modellierten Steuergeräts darzustellen. Daneben müssen Basis-Software-Komponenten emuliert werden wie Mode Manager (z.B. zur Simulation des Startup-Verhaltens der Steuergeräte), Diagnostic Error Manager (z.B. zur Prüfung, welche Fehlercodes während einer Simulation eingetragen werden), NV-RAM-Manager und COM-Stacks.

4 Virtuelle HIL-Simulation – ein Ausblick

Die Simulation des Seriencodes einer einzelnen Funktion wie einer Leerlaufregelung oder einer Lichtsteuerung auf einem PC ist als so genannte Software-in-the-Loop- (SIL-) Simulation bereits heute Bestandteil der modellbasierten Entwicklung und ermöglicht die Verlagerung von Modul-Tests in frühe Phasen. Während in der SIL-Simulation typischerweise einzelne Software-Komponenten getestet werden, umfasst die im vorhergehenden Abschnitt beschriebene PC-Simulation der Software-Architektur das Zusammenspiel mehrerer Software-Komponenten bis hin zur Simulation ganzer Steuergeräteverbunde. Auch die Berücksichtigung von Basis-Software- und Buskommunikations-Effekten geht über eine reine SIL-Simulation hinaus.

Langfristig wird sich damit eine virtuelle HIL-Simulation („Virtual HIL“) links im V-Entwicklungsprozess etablieren, mit der Steuergeräte- und Steuergeräteverbund-Tests, die heute am Hardware-in-the-Loop- (HIL-) Simulator durchgeführt werden, in die PC-Simulation vorverlagert werden können. Solche Simulationen umfassen dann nicht nur den i.O.-Fall, sondern auch Fehlersituationen wie I/O-Signalveränderungen, Bus-Fehler oder Scheduling-Probleme. Dazu wird es analog zu konventionellen HIL-Simulatoren eine virtuelle Failure-Insertion-Unit („Virtual FIU“) geben, die zum Beispiel den Test der Diagnose ermöglicht.

Zu berücksichtigen ist, dass die Einsatzmöglichkeiten der virtuellen HIL-Simulation auf absehbare Zeit durch die Simulationsgenauigkeit der Basissoftware und des Hardware-Verhaltens beschränkt werden. Unklar ist derzeit auch noch die Performanz der Simulation großer Systemmodelle auf PCs. Viele Fehler, zum Beispiel Kurzschlüsse oder andere elektrische Effekte, werden auch langfristig nur mit HIL-Tests am echten Steuergerät erkannt werden. Eine wichtige Anforderung an die virtuelle HIL-Simulation ist deshalb die Durchgängigkeit zu den konventionellen HIL-Tests. Hierzu gehört die Verwendung der gleichen Regler- und Streckenmodelle sowie die Übernahme der Testfälle, die in beiden Szenarien anwendbar sind, und die einheitliche Anbindung der Testautomatisierung.

Literaturverzeichnis

[AUT] AUTOSAR Partnership: http://www.autosar.org/

[Ott07] Otterbach, R.; Niggemann, O.; Stroop, J.; Thümmler, A.; Kiffmeier, U.: System Verification throughout the Development Cycle. ATZ Automobiltechnische Zeitschrift 04/2007, Vieweg Verlag / GWV Fachverlage GmbH.

[St07] Stichling, D.; Niggemann, O.; Otterbach, R.: Effective Cooperation of System Level and ECU Centric Tools within the AUTOSAR Tool Chain. SAE Congress, Detroit, 2007.

Model-Based Design for IEC 61508: Towards Translation Validation of Generated Code

Mirko Conrad
The MathWorks, Inc.

Abstract: Production code generation with Model-Based Design has replaced manual coding in various vehicle subsystems and generated code is increasingly being deployed in safety-related applications as well. To validate the translation process and to ensure quality and functional safety, generated application software components shall be subjected to a combination of quality assurance measures. Further, compliance with safety standards needs to be demonstrated.
On principle, translation validation of generated code could be carried out in the same way as for hand code. However, from a process efficiency point of view, this would leave something to be desired. Therefore, engineering methods and tools to validate and verify generated code are highly desirable. As a step towards this goal, a workflow for verification and validation of models and generated code is outlined in this paper and mapped onto the objectives of IEC 61508-3.

1 Introduction

The increasing application of **embedded in-vehicle software** has resulted in a staggering complexity that has proven difficult to negotiate with conventional design approaches. Due to its capability to address the software complexity and productivity challenges, **Model-Based Design** [CFG+05, ADL07, MBD] is becoming the preferred software engineering paradigm for the development of application software components in various vehicle subsystems. The core idea is that an initial **executable graphical model** representing the application software component to be developed is refined and augmented until it becomes the blueprint for the final implementation through automatic **code generation**. Simulink products have become popular for Model-Based Design. Graphical modeling with time-based block diagrams and event-based state machines is supported by **Simulink**® [SL] and **Stateflow**® products; production code generation is supported by the **Real-Time Workshop® Embedded Coder™** [RTW-EC] product.

Regardless of the software development paradigm applied, it is crucial to subject the software being developed to an appropriate combination of **verification and validation** (V&V) activities to detect errors and produce confidence in the correct functioning of the embedded application software, i.e. to carry out a **translation validation** [PSS98]. V&V of software developed using Model-Based Design could be carried out in the same way as for hand code. However, the presence of executable graphical models opens up new, more efficient translation validation approaches. Models provide an excellent source of information that could be exploited for various V&V techniques. Considering the question of process efficiency, one should make the most of these possibilities. Therefore, engineering approaches for the translation validation of generated code tightly integrated with the construction activities of Model-based Design are highly desired.

In the recent past, Model-Based Design with code generation has been successfully employed to produce software for **safety-critical applications**, see e.g., [Pot04, TMW05, ADL07]. In such application contexts, additional requirements imposed by safety **standards** have to be met and the objectives and recommendations outlined therein must be mapped onto Model-

Based Design. This could become difficult because various safety standards including **IEC 61508** were assembled in the 1990s, i.e. before Model-Based Design was broadly adopted.

Contribution: This paper proposes a **workflow for verification and validation of models and generated code** that is intended to be used as an IEC 61508-3 compliant translation validation procedure. The workflow carries the advantages of Model-Based Design forward to safety-critical applications. Available tool support is also discussed.

2 A Workflow for Application-specific Verification and Validation of Models and Generated Code

IEC 61508-3 and other safety standards call for **application-specific verification and validation** regardless of the tool chain and the development paradigm used. When using Model-Based Design it seems natural to carry out a large share of the necessary V&V activities on model level. Unfortunately, more detailed guidance for the translation validation is rare: IEC 61508-3 dates back to 1998 and therefore does not have a notion of model testing and code generation. Its objectives need to be interpreted and mapped onto Model-Based Design [Con07, EC07]. A model-based approach to application-specific V&V is to divide the problem into two steps: First to demonstrate that the model is correct and meets all requirements, and then to show that the generated code is equivalent to the model [Ald01]. The first step, referred to as **design verification**, combines suitable V&V techniques on model level. The second step, **code verification**, uses equivalence testing and complementary techniques to demonstrate equivalence between the model and the generated code (Fig 1).

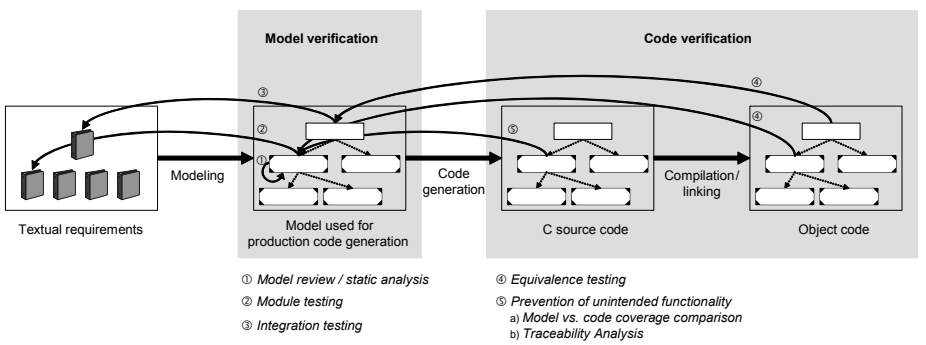


Fig. 1: Workflow for application-specific verification and validation

2.1 Verification and Validation on Model Level (Design Verification)

Design verification aims at gaining confidence in the model used as input for production code generation. It takes place before the actual code generation will be carried out. A **combination of reviews, static analyses and testing activities on model level** provides assurance that the design satisfies its associated requirements, i.e. it is verified. Requirements for design verification can be derived from IEC 61508-3 clauses 7.4.6 – 7.4.8.

Reviews and Static Analyses on Model Level: Model subsystems considered modules (model components) should be reviewed. Manual **model reviews** should be supported by automated **static analyses**. **Modeling guidelines** should be used and adherence with these guidelines should be assessed. Modeling constructs that are not suited or not recommended for production code generation should not be used. Model reviews are expected to provide an

efficiency gain over code reviews. [Kuf07] e.g. reports that model reviews could be about 5 times more efficient than source code reviews.

Tool support: The Simulink Block Support tables for Real-Time Workshop Embedded Coder identify blocks not suited for production code generation. The MAAB V2.0 guidelines [MAAB] contain general modeling guidelines for control algorithms. Adherence to guidelines and safety standards can be partially enforced by using modeling standard checks in Model Advisor [Beg07]. Model reviews can be facilitated by using reports or web views generated with Simulink Report Generator™ product.

Module and Integration Testing on Model Level: Model components should be functionally tested using systematically derived test vectors. These **module tests** should demonstrate that each module performs its intended function and does not perform unintended functions. As the module testing is completed then module **integration testing** should be performed to show that all model components interact correctly.

Tool support: The Simulink Verification and Validation product supports various facets of model testing. Model level test can be carried out within the Simulink environment.

2.2 Verification and Validation on Code Level (Code Verification)

By using **equivalence testing** between the model used for production code generation and the executable derived from the generated C code and augmenting measures to demonstrate the **absence of unintended functionality**, code verification demonstrates that the execution semantics of the model is preserved during code generation, compilation and linking. Successful code verification indicates that the outputs of Real-Time Workshop Embedded Coder and the compiler / linker tool-chain are correct for the design instance being checked. It's more difficult to derive requirements for the code verification part from IEC 61508-3, because such comparative testing is not addressed directly there. So, the standard's gist needs to be carried over.

Equivalence Testing: The core part of code verification is comparative testing between the model used for production code generation and the resulting object code. **Equivalence testing** (back-to-back testing) demonstrates numerical equivalence between the two artifacts. It's carried out by stimulating both the model used for code generation and the executable derived from the generated code with identical test vectors. Since complete testing is impossible, stimuli (test vectors) should sufficiently cover the different structural parts of the model. IEC 61508-3 doesn't call out specific coverage goals, but secondary sources such as [SS05] recommend that some test coverage metric should be visible at SIL 2 and above. Discussions of equivalence testing procedures can be found in [SC03, SC05, SCD+07]. Testing for numerical equivalence is unique in that the expected outputs for the test vectors do not have to be provided a-priori [Ald01]. This makes in-depth equivalence testing ideally suited to automation.

Tool support: The Simulink Design Verifier product helps create (additional) test vectors for structural testing. Model coverage information can be collected by using the Model Coverage Tool. Processor-in-the-loop (PIL) testing can be used to execute the object code on a target-like hardware.

Prevention of Unintended Functionality: Multiple techniques are available to demonstrate that the generated C code does not perform any **unintended function.** These include:

Model and code coverage are measured during equivalence testing and compared against each other **(model vs. code coverage comparison**). Discrepancies w.r.t. comparable

coverage metrics should be assessed. If the code coverage achieved is less than the model coverage, unintended functionality could have been introduced.

A **traceability analysis** of the generated code can be performed to ensure that all parts of the generated source code can be traced back to the model.

Tool support: Code coverage information can be derived from the IDE or by integrating a code coverage tool. Real-Time Workshop Embedded Coder provides a traceability report with mapping information between model blocks and the generated code.

3 Related work

Earlier work on V&V of production code generation includes [Edw99, Bur04, SWC05]. Typically a combination of translation validation and translator validation approaches is discussed. However, none of the papers discusses the proposed measures or techniques in the context of IEC 61508. Translator validation can be considered orthogonal to translation validation discussed in this paper. A popular means of translator validation is validation testing. MISRA-C:2004 [MISRA-C] suggests documented validation testing as a possible approach to gain confidence in the tools used. This approach has been applied to Model-Based Design by major automotive companies and the TÜV. A production code generation tool chain including Embedded Coder successfully passed the resulting Automotive Code Validation Suite (AVS), a test suite to validate quality and robustness of the code generator, compiler, linker tool chain [TÜV05; PSB07]. Underlying concepts on such test suites are laid out for example in [SC03, Stü06, ZPC+06, SCD+07]. The interplay between model and code coverage has been examined analytically by [Ald01] and statistically by [BCS+03].

4 Conclusions

The increasing usage of Model-Based Design with production code generation calls for new methods and tools for verifying and validating generated code. Model-Based Design allows for new V&V approaches utilizing the existence of executable graphical models as well as multiple executable artifacts which could be checked for equivalence. For safety-critical developments existing V&V approaches need to be reconciled with safety standards such as IEC 61508. The proposed workflow for verification and validation of models and generated code could be viewed as instantiation of the V&V requirements of IEC 61508-3, which exploits the possibilities of code generation as an integral part of Model-Based Design. Users of Simulink and Real-Time Workshop Embedded Coder can use this workflow as a reference to create their own project and SIL specific variants of this workflow. The presented workflow targets typical safety-critical in-vehicle applications, i.e. SIL 2 or SIL 3.

While adhering to the recommendations in this document will reduce the risk that an error is introduced in the development and not be detected, it is not a guarantee that the system being developed will be safe. Conversely, if some of the recommendations in this document are not followed, it does not mean that the system being developed will be unsafe.

References

[ADL07] Kosten-/Nutzenanalyse der modellbasierten Softwareentwicklung im Automobil. Arthur D. Little, 2007.

[Ald01] Aldrich, W.: Coverage Analysis for Model-Based Design Tools. TCS 2001.

[BCS+03] Baresel, A.; Conrad, M.; Sadeghipour, S.; Wegener, J.: The Interplay between Model Coverage and Code Coverage. EuroSTAR ‘03, Amsterdam, Netherlands.

[Beg07] Begic, G.: Checking Modeling Standards Implementation. The MathWorks News & Notes, June 2007.

[Bur04] Burnard, A.: Verifying and Validating Automatically Generated Code. IAC ’04, Stuttgart, Germany.

[CFG+05] Conrad, M.; Fey, I.; Grochtmann, M.; Klein, T.: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler. Inform. Forsch. Entwickl. 20(1-2): 3-10 (2005).

[Con07] Conrad, M.: Using Simulink and Real-Time Workshop Embedded Coder for Safety-Critical Automotive Applications. MBEES '07, Dagstuhl, Germany.

[EC07] Erkkinen, T.; Conrad, M.: Safety-Critical Software Development Using Automatic Production Code Generation. SAE World Congress 2007, Detroit, USA.

[Edw99] Edwards, P.D.: The Use of Automatic Code Generation Tools in the Development of Safety-Related Embedded Systems. ERA Report 99-0484, 1999.

[IEC61508-3] IEC 61508-3:1998. Int. Standard Functional safety of electrical/ electronic/ programmable electronic safety-related systems - Part 3: Software requirements. 1998.

[Kuf07] Kuffner, W.: Umsetzung der Anforderungen aus der funktionalen Sicherheit in der Fahrwerkselektronik. 11. Int. Fachkongress Fortschritte in der Automobil-Elektronik 2007; Ludwigsburg, Germany

[MAAB] Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow® V2.0, 2007. www.mathworks.com/industries/auto/maab.html

[MBD] Model-Based Design: www.mathworks.com/applications/controldesign/description

[MISRA-C] MISRA-C:2004. Guidelines for the use of the C language in critical systems. MIRA, 2004.

[Pot04] Potter, B.: Use of The MathWorks Tool Suite to develop DO-178B certified code. ERAU/FAA Software Tools Forum 2004, Daytona Beach, USA.

[PSB7] Pofahl, E., Sauer, T., Busa, O.: AVS - A Test Suite for Automatically Generated Code. MAC ‘07, Dearborn, USA.

[PSS98] Pnueli, A.; Siegel, M.; Singerman, E.: Translation Validation. TACAS'98, Lisbon, Portugal.

[RTW-EC] Real-Time Workshop® Embedded Coder™: www.mathworks.com/products/rtwembedded

[SC03] Stürmer, I.; Conrad, M.: Test Suite Design for Code Generation Tools. ASE '03, Montreal, Canada.

[SC05] Stürmer, I.; Conrad, M.: Ein Testverfahren für optimierende Codegeneratoren. Inform. Forsch. Entwickl. 19(4): 213-223 (2005)

[SCD+07] Stürmer, I.; Conrad, M.; Dörr, H.; Pepper, P.: Systematic Testing of Model-Based Code Generators. IEEE Transactions on Software Engineering, Sep 2007, pp. 622-634

[SL/SF] Simulink®, Stateflow®: www.mathworks.com/products/simulink

[SS05] Smith, D.J.; Simpson, K.G.L.: Functional Safety – A straightforward guide to applying IEC 61508 and related standards. Elsevier Butterworth-Heinemann, 2005

[Stü06] Stürmer, I.: Systematic Testing of Code Generation Tools - A Test suite-oriented Approach for Safeguarding Automatic Code Generation. TU Berlin, Germany, 2006.

[SWC05] Stürmer, I.; Weinberg, D.; Conrad, M.: Overview of Existing Safeguarding Techniques for Automatically Generated Code. SEAS '05, St. Louis, USA.

[TMW05] The MathWorks: Alstom generates production code for safety-critical power converter control systems. 2005. www.mathworks.com/products/rtwembedded/userstories.html?file=10591

[TÜV05] Validation of the MathWorks code generator Real-Time Workshop® Embedded Coder with the Autocode Validation Suite v3.0. Report No. 968/EL 211.02/05, TÜV Rheinland, 2005.

[ZPC+06] Zelenov, S.V.; Petrenko, A.K.; Conrad, M.; Fey, I.: Automatic Test Generation for Model-Based Code Generators. ISoLA ’06, Paphos, Cyprus.

Effizienzsteigerung im Systemtest von Steuergeräten

Eine empirische Untersuchung

Matthias Wiemann, Jörg Gericke

Siemens AG
Corporate Research & Technologies
CT PP 6
Otto-Hahn-Ring 6
81730 München
joerg.gericke@siemens.com

Abstract: Die Softwarequalität in Body-Controllern kann im Systemtest durch verhaltensorientierte Testmethodiken erhöht werden. In der Literatur wurden hierzu bereits zahlreiche Verfahren vorgeschlagen. Die vorliegende Arbeit stellt Ergebnisse eines empirischen Vergleichs einer herkömmlichen, intuitiven mit einer verhaltensorientierten Testmethodik vor. Die Analyse der Anzahl identifizierter Fehler bestätigt, dass durch die verhaltensorientierte Testmethodik eine deutlich höhere Effizienz und Effektivität bzgl. der Fehleridentifikation erzielt werden kann. Bereits das Aufstellen der erforderlichen Verhaltensmodelle ermöglicht die Fehleridentifikation in frühen Phasen. Der Aufwand zur Modellerstellung des verhaltensorientierten Tests wird von den Testmanagern als geringer eingestuft als der Aufwand für den intuitiven Test.

1 Motivation

In der Automobilindustrie werden heute 80-90% der Innovationen durch Software realisiert [DV05]. Dies beinhaltet ein exponentielles Anwachsen des Softwarevolumens in Automobilen [DK04] mit erheblicher Komplexitätssteigerung der entwickelten Systeme [Be06] und dadurch induzierte Qualitätseinbußen [Be06, Dp03, Eq07, OS04]. Fahrzeugherstellerübergreifend wuchs die Zahl der Fahrzeugpannen aufgrund von mängelbehafteter Elektronik stark an [Dp03, Eq07, OS04]. Funktions- und verhaltensorientierte Testverfahren sollen dieser Entwicklung entgegenwirken. Diese scheinen nur zögerlich Eingang in die Praxis zu finden. Ein möglicher Grund mag darin liegen, dass eine Überlegenheit hinsichtlich der Effektivität in der Fehleridentifikation bei vertretbarem Aufwand in der Praxis bezweifelt wird. In dieser Arbeit wird daher diese Effektivität durch eine funktions- und verhaltensorientierte Teststrategie im Kontext der Automobilindustrie empirisch untersucht und einer herkömmlichen, intuitiven Testmethodik vergleichend gegenübergestellt.

2 Konzept der durchgeführten Untersuchung

2.1 Iterative Systementwicklung

Untersucht wurden die Systementwicklungen von zwölf *Steuergeräten* (engl. *Body-Controller*). Die Steuergeräte wurden nach einer iterativen Anwendung des *V-Modells* entwickelt. In jeder Iteration wurde der Funktionsumfang erweitert und getestet. Betrachtet werden vier V-Modell-Iterationen mit Anwendung einer herkömmlichen, intuitiven Testmethodik. Vier Steuergeräte wurde zur Qualitätssteigerung in der dritten Iteration mit einer speziell für den Systemtest von Steuergeräten entwickelten, funktions- und verhaltensorientierten Testmethodik validiert.

2.2 Funktions- und verhaltensorientierte Testmethodik

Ausgangsbasis war eine intuitive Testfallgenerierung. Diese basierte auf der Erfahrung der Tester sowie auf der Vorgabe, dass für jede Anforderung mindestens ein Testfall auszuführen ist. Im Vergleich zur intuitiven Testmethodik soll eine funktions- und verhaltensorientierte Testmethodik die Qualität der Steuergeräte durch Verringerung der Restfehleranzahl erhöhen. Diese Methodik fokussiert das *Systemverhalten*, welches durch hierarchische *Verhaltensmodelle* beschrieben wird. Dieses Systemverhalten war in textuellen Spezifikationen beschrieben und wurde in Verhaltensmodelle, z.B. Zustandsautomaten, überführt.

Über den verhaltensorientierten Systemtest hinausgehend wurde die Steuergerätesoftware zusätzlich mit dem funktionsorientierten Testverfahren der *Äquivalenzklassenanalyse* [My79] geprüft. Hierbei wurden die Wertebereiche der Eingabevariablen unter Zuhilfenahme der *Klassifikationsbaummethode* [GG93] in Äquivalenzklassen unterteilt. Die Grenzen der Äquivalenzklassen wurden durch eine *Grenzwertanalyse* [My79] validiert. In der Praxis erwies sich die resultierende Testfallanzahl jedoch als zu hoch. Daher wurde die Testfallanzahl mit der bewährten *paarweisen Überdeckung von Parameterwerten* [Wi00] reduziert.

2.3 Manuelle Verhaltensmodellierung und automatisierte Testfallgenerierung

Steuergeräte eins bis vier wurden in der dritten Iteration mit der beschriebenen verhaltensorientierten Testmethodik validiert. Die zur Testfallgenerierung notwendigen Verhaltensmodelle wurden von den Testern manuell aufgestellt. Die Tester wurden bezüglich der Modellerstellung und Testfallgenerierung vor Durchführung der empirischen Untersuchung durch eine zweitägige Schulung befähigt, unter Coaching die notwendigen Verhaltensmodelle aufzustellen. Die Verhaltensmodellierung sowie die automatisierte Testfallgenerierung wurden durch die bei der Siemens AG, CT PP entwickelte Werkzeuglandschaft *GATE – Generic and Automated Test Environment –* unterstützt. Dieses generiert u.a. auf den Verhaltensmodellen basierend entsprechend des *Wp-Pfadüberdeckungskriteriums* [FBK91] automatisiert Testfälle.

2.4 Bewertungskriterien zur Effektivität der Testmethodiken

Die Effektivität einer Testmethodik soll anhand der Anzahl gefundener Fehler pro Iteration bewertet werden. Zum geeigneten Vergleich zwölf untersuchter Steuergeräteprojekte werden anstelle absoluter Fehleranzahlen relative Anzahlen der insgesamt gefundenen Fehler angegeben. Die Gesamtanzahl der in den betrachteten vier Iterationen identifizierten Fehler wird pro Steuergerätegruppe (s. Kapitel 3.1) als 100% gesetzt.

3. Ergebnisse

3.1 Effektivere Fehleridentifikation

Die zwölf untersuchten Steuergeräte sind in zwei Gruppen unterteilt: Steuergeräte 1-4 wurden in Iteration drei mit der verhaltensorientierten Testmethodik validiert. Steuergeräte 5-12 wurden ausschließlich unter Anwendung der herkömmlichen, intuitiven Testmethodik getestet. Abbildung 1 zeigt die durchschnittliche relative Anzahl gefundener Fehler für die untersuchten Steuergerätegruppen nach Iterationen aufgeschlüsselt.

In den ersten beiden Iterationen wurden alle zwölf Steuergeräte entsprechend der herkömmlichen Teststrategie validiert. Da die Werte sich nur in Nachkommastellen unterscheiden und Projekteigenschaften wie Komplexität, Systemgröße und Erfahrung der Entwickler und Tester vergleichbar sind, werden die Steuergeräte auch hinsichtlich anderer testrelevanter Eigenschaften wie Testbarkeit als vergleichbar angesehen.

In der dritten Iteration wurde der Funktionsumfang erheblich erweitert, was sich in allen zwölf Steuergeräten durch eine deutlich erhöhte Anzahl identifizierter Fehler widerspiegelt. Zur Verbesserung der Qualität wurden Steuergeräte 1-4 mit der vorgestellten funktions- und verhaltensorientierten Testmethodik getestet. Referenzsteuergeräte 5-12 wurden weiterhin auf herkömmliche Weise (s. Kapitel 2.2) getestet. Mit Ausnahme der Testmethodik blieben restliche Aspekte wie Programmier- und Testteams sowie Automatisierungen konstant. Steuergeräte 1-4 weisen aufgrund einer etwa 1,5-mal (s. Abbildung 1) so hohen relativen Anzahl identifizierter Fehler eine deutliche Steigerung der Effizienz und Effektivität im Vergleich zu den Steuergeräten 5-12 auf.

In der vierten Iteration wurden nur wenige Funktionen hinzugefügt. Sämtliche zwölf Steuergeräte wurden wieder entsprechend der herkömmlichen, intuitiven Testmethodik getestet. Die relative Anzahl identifizierter Fehler sank für die vorher mit der verhaltensorientierten Testmethodik getesteten Steuergeräte 1-4 deutlicher, als für die restlichen Steuergeräte 5-12. Diese Beobachtung ist auf zwei miteinander in Beziehung stehende Ursachen zurückzuführen: a) die herkömmliche Testmethodik identifiziert weniger Fehler als die funktions- und verhaltensorientierte Testmethodik und b) in den Steuergeräten 5-12 wurden Fehler erst jetzt identifiziert, die gemäß der Annahme der vergleichbaren Fehlerstruktur aller Steuergeräte bereits in der dritten Iteration hätten identifiziert werden können bzw. sollen. Die verhaltensorientierte Testmethodik führte folglich zu der gewünschten guten und früheren Fehlererkennung. Dieses führt entsprechend dem Kos-

tenmodell von Boehm [Bo81] zu einer 11%-igen Gesamtprojektkosteneinsparung.

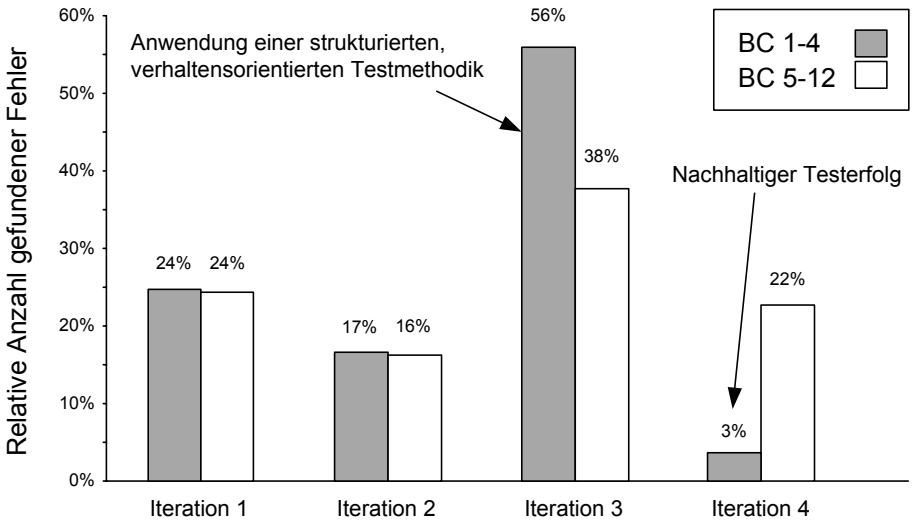


Abbildung 1. Relative Anzahl identifizierter Fehler nach Iteration

3.2 Effizientere Testfallerstellung

Die Erstellung der Verhaltensmodelle für den Systemtest wird in der Praxis von Testmanagern häufig als zu aufwendig kritisiert. Die Erstellung der verhaltensorientierten Testfälle erwies sich jedoch in allen Fällen innerhalb des ohnehin knapp kalkulierten Zeitrahmens als möglich. Die Testmanager beurteilten den Zeitaufwand der Verhaltensmodellierung mit einfachen Werkzeugen wie folgt als wirtschaftlich positiv:

- Unmittelbar vor der Testfallerstellung in der Testmethodik geschulte Tester benötigten vier Tage zur Testfallerstellung, wofür ursprünglich erfahrungsgemäß acht Tage mit bisheriger Testmethodik eingeplant waren. Die Zeitersparnis betrug in diesem Fall etwa 50%.

- Für die Testfallerstellung eines weiteren untersuchten Steuergerätes benötigte ein hinzugezogener, erfahrener Tester zwei der auf Basis der Erfahrung eingeplanten zehn Tage. Die Zeitersparnis betrug hier etwa 80%.

- Die Verhaltensmodelle eines der Steuergeräte konnten für ein parallel entwickeltes Fahrzeug wiederverwendet werden. Die Wiederverwendung war möglich, da sich das Verhalten beider Systeme nur marginal unterscheidet. Nach lediglich geringen Anpassungen lag das benötigte Verhaltensmodell vor. Somit entfiel fast der komplette Aufwand zur Erstellung eines Verhaltensmodells und Testgrunddaten.

Das Aufstellen der Verhaltensmodelle identifizierte grundsätzliche Spezifikationsfehler, die in vorangegangenen Iterationen nicht identifiziert wurden. Die Ursache liegt in der

expliziten Betrachtung von Abhängigkeiten zwischen Anforderungen. Folglich ist allein das Aufstellen – ohne Ausführung – von verhaltensorientierten Testfällen bereits ein Mittel zur Identifikation von Spezifikationsfehlern, welches in frühen Projektphasen parallel zur Implementierung angewendet werden kann.

3.3 Validität der Ergebnisse

Die untersuchten Steuergeräte hätten zur Maximierung der Aussagekraft der Untersuchung parallel einmal mit der intuitiven und einmal mit der funktions- und verhaltensorientierten Testmethodik validiert werden müssen. Eine solche Versuchsdurchführung war im Hinblick auf Ressourcen und Kosten leider nicht möglich. Die Vergleichbarkeit der zufällig gebildeten Steuergerätegruppen ist nicht bewiesen, dennoch sprechen zahlreiche Indizien (s. Kapitel 3.1) für diese Annahme.

4. Schlussfolgerung

In dieser empirischen Gegenüberstellung einer intuitiven und einer funktions- und verhaltensorientierten Testmethodik wird ersichtlich, dass eine Anwendung einer funktions- und verhaltensorientierten Testmethodik Fehler deutlich effektiver und früher identifiziert. Ein erhöhter Aufwand zur Modellerstellung wurde nicht beobachtet. Vielmehr wurde von Kosteneinsparungen im ersten Einsatz berichtet. Daher sollte der Praxiseinsatz einer verhaltensorientierten Testmethodik verstärkt durchgesetzt werden.

Literaturverzeichnis

[Be06] K. Bender. Embedded Systems-qualitätsorientierte Entwicklung. Springer. p. 1-10. 2006

[Bo81] W. Boehm, Software Engineering Economics. Prentice Hall PTR, USA. 1981

[DK04] J. Dannenberg, C. Kleinhans. The Coming Age of Collaboration in the Automotive Industry. Mercer Management Journal, 17:88–94. 2004

[Dp03] dpa Meldung vom 30.09.2003 zur 33. Jahrestagung der GI, zitiert nach http://www.heute.t-online.de/ZDFheute/artikel/21/0,1367,COMP-0-2070133,00.html

[DV05] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp. Boosting Re-use of Embedded Automotive Applications through Rich Components. Elsevier Science. 2005

[Eq07] Methoden zur Unterstützung der entwicklungsbegleitenden Qualitätssicherung von eingebetteter Software (EQUAL). http://www.embedded-quality.de/

[FBK91] S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou. Test selection based on finite state models. IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 591-603. 1991

[GG93] M. Grochtmann, K. Grimm. Classification Trees for Partition Testing. Software Testing, Verification & Reliability, Vol. 3, No. 2, pp. 63-82. 1993

[My79] G. Myers. The Art of Software Testing. J. Wilsey & Sons, pp. 147-174. 1979

[OS04] R. Otterbach, F. Schütte. Effiziente Funktions- und Software-Entwicklung für mechatronische Systeme im Automobil. Dspace. 2004

[Wi00] A. Williams. Determination of test configurations for pair-wise interaction coverage. Proceedings of TestCom 2000, Ottawa, Canada, pp. 59-74. 2000

Integration von Prozessen und Werkzeugen im agilen Zeitalter (IntegrA' 08)

Inhaltsverzeichnis

Workshop zur Integration von Prozessen und Werkzeugen im agilen Zeitalter 125
Walid Maalej, Oliver Creighton, Ernst Pohn

Integrating Software Lifecycle Models into a uniform Software Engineering Model 128
Jonas Helming

Integration des V-Modell®XT im Visual Studio Team Foundation Server – Erfahrungen aus dem Projekt CollabXT 135
Marco Kuhrmann

Integrationsnotwendigkeit im agilen Zeitalter 141
Jennifer Schiller

Das Link-System: Von Weblinks zu universellen, sensitiven Hyperlinks 149
Dirk Fischer

TIME - Tracking Intra- and Inter-Model Evolution 157
Maximilian Kögel

Change Propagation with the Change Notification Bus 165
Tobias Rodenbach, Lena Wiese

Workshop zur Integration von Prozessen und Werkzeugen im agilen Zeitalter (IntegrA' 08)

Walid Maalej[1], Oliver Creighton[1], Ernst Pohn[2],

[1] Technische Universität München,
Walid.Maalej@in.tum.de

[2] Siemens, CT,
Oliver.Creighton@siemens.com

[3] Rohde & Schwarz,
Ernst.Pohn@rohde-schwarz.com

1 Problemstellung

Die Entwicklung von modernen Softwaresystemen ist heutzutage ohne Unterstützung durch spezialisierte Werkzeuge nicht mehr vorstellbar. Es können zum Beispiel folgende Werkzeuge in einem einzelnen Projekt zum Einsatz kommen: Telelogic Doors [Do07] für die Verwaltung der Anforderungen, Microsoft Project [Pr07] für die Planung und Kontrolle von Ressourcen, IBM CleaseCase [Cl07] für das Konfigurationsmanagement, Bugzilla [Bu07] für die Verwaltung von Änderungswünschen und Bugs, Enterprise Architect [En07] für die Modellierung sowie Eclipse [Ec07] und Visual Studio [Vi07] für die Implementierung und Konstruktion. Die zielgerichtete Integration dieser heterogenen Werkzeuge stellt eine Herausforderung sowohl für Projektbetroffene als auch für Werkzeughersteller dar.

Aufgrund starken Wettbewerbs, schnelllebiger Märkte sowie hoher Flexibilitätsanforderungen gehören häufige Änderungen zum Alltag. Integrationsprobleme schränken dabei die Produktqualität und Entwicklungseffizienz ein. Fehlende Durchgängigkeit bei änderungsgetriebener, inkrementeller Entwicklung führt insbesondere zu folgenden Nachteilen:

- Die prozess- und werkzeugübergreifende Rückverfolgbarkeit [DP07] (Traceability) ist nur mit hohem, zum Teil manuellem, Aufwand möglich. Zum Beispiel benötigt der Nachweis über Abdeckung von Testfällen für bestimmte sich häufig ändernde Anforderungen eine stetige Verwaltung und Anpassung von den Beziehungen zwischen den Tests und den Anforderungen.

- Projektweite Beurteilung der Auswirkungen von Änderungen, z.B. von Anforderungen, auf die Implementierung, auf die Planung oder auf die Tests ist schwierig.

- Redundante Informationen, verursacht durch Überlappung des Funktionsumfangs, führen zur Inkonsistenz. Der Abgleich, z.B. eines Meilensteines im Release-Plan, im Anforderungsdokument ist schwierig und zeitaufwendig.

Forscher und Werkzeughersteller haben sich bisher auf technische Integrationsaspekte, z.B. Austauschformate, fokussiert [WD07]. Neue Erkenntnisse zeigen aber die Wichtigkeit der Benutzerbedürfnisse und Kollaboration, vor allem bei der agilen Entwicklung. Eine neue funktionale Betrachtung des Integrationsproblems wird benötigt, wobei Projektbetroffene in den Fokus rücken.

2 Ziele und Schwerpunkte

IntegrA'08 [In08] bringt u.a. folgende Interessengruppen zusammen:

- Erfahrene Forscher, die sich mit den Entwicklungsprozessen vor allem agilen Prozessen bzw. Integration der Werkzeuge befassen.

- Vertreter von Standardisierungsabteilungen, Infrastrukturbetreibern sowie Vertreter unterschiedlicher Projektbetroffener aus führenden Unternehmen im Bereich Softwareentwicklung, mit Erfahrungen in der Integrationsproblematik.

- Vertreter von Werkzeugherstellern, die bereits Erfahrungen mit Integration gesammelt haben und die Integrationsanforderungen genauer diskutieren und definieren wollen.

Dabei dienen die Vorträge und Diskussionen zum Erfahrungsaustausch zwischen den Interessengruppen sowie der Erstellung von zukünftigen Leitfäden und einer gemeinsame Arbeitsagenda. Folgende Schwerpunkte werden daher besonders im Fokus stehen:

- Erfahrungsberichte aus unterschiedlichen Perspektiven (Benutzer, Hersteller, Forscher) über bisherige Lösungen und noch vorhandene Probleme.

- Bedarfsermittlungsstudien, Ursachenanalysen und Lösungsvorschläge für die Integration der heterogenen und agilen Prozess- und Werkzeuglandschaften.

- Integrationsanforderungen nach den neuen Erkenntnissen der agilen und kollaborativen Produktentwicklung und Projektmanagement.

- Demonstrationen von innovativen Integrationslösungen, die den Bedürfnissen einer flexiblen, änderungsgetriebenen Softwareentwicklung entsprechen.

- Roadmaps für die verschiedenen Beteiligungsgruppen.

3 Programmkommitte

- Walid Maalej, Technische Universität München
- Oliver Creighton, Siemens
- Ernst Pohn, Rohde & Schwarz
- Werner Burger, Diehl Aerospace
- Jan Czaijka, Rohde & Schwarz
- Jenz Korte, SYNDATO IT solutions
- Tom Krauss, GEBIT Solutions
- Dirk Fischer, Rohde & Schwarz
- Robert Neher, POLARION Software
- Timo Wolf , University of Viktoria, Kanada
- Renate Stüka, Telelogic Europe
- Thomas Zimmermann, University of Calgary, Kanada

Literaturverzeichnis

[Be01] Manifesto for Agile Software Development, 2001. http://agilemanifesto.org/

[Bu07] Bugzilla Homepage, 5.12.2007. http://www.bugzilla.org/

[Cl07] ClearCase Homepage, 5.12.2007, http://www.ibm.com/software/awdtools/clearcase/

[Do07] Doors Homepage, 5.12.2007. http://www.telelogic.com/products/doors/index.cfm

[DP03] Dutoit A. and Peach B.: Eliciting and Maintaining Knowledge for Requirements Evolution, In Managing Software Engineering Knowledge, Aurum, A., Jeffery, R., Wohlin, C., Handzic, M., 2003.

[Ec07] Eclipse Homepage, 5.12.2007. http://www.eclipse.org/

[En07] Enterprise Architect Homepage, 5.12.2007. http://www.sparxsystems.com.au/

[FS84] Fischer, G. and Schneider, M.: Knowledge-based communication processes in software engineering. In Proceedings of the International Conference in Software Engineering. IEEE Press, Piscataway, NJ. 1984.

[In08] IntegrA' 08 Workshop Homepage, 2008. http://www1.in.tum.de/Integra08

[Pr07] Microsoft Project Homepage, 5.12.2007. http://office.microsoft.com/en-us/project/default.aspx

[Vi07] Visual Studio Homepage, 5.12.2007. http://msdn2.microsoft.com/en-us/vstudio/default.aspx

[WD07] Wicks, M.N. and Dewar, R.G.: A new research agenda for tool integration, Journal of Systems and Software, 2007.

Integrating Software Lifecycle Models into a uniform Software Engineering Model

Jonas Helming

Technische Universitaet Muenchen
Department of Computer Science
Chair for Applied Software Engineering
Bolzmannstraße 3
D-85748 Garching bei München, Germany
helming@in.tum.de

Abstract: Software lifecycle models differ in their artifacts as well as in the dependencies between the included tasks and activities. Therefore support-tools, which support these lifecycles, are typically specialized on a certain lifecycle. A tool for Scrum [Sc07] for example may model user stories and sprints, but exclude risksless shift of the applied software lifecycle, e.g. from a heavyweight to an agile methodology. Furthermore existing tools are mostly isolated from the product model, which complicates traceability between modeling tasks and the objects of tasks, e.g. UML models [O04]. This paper proposes a unified Meta-Model to integrate the software lifecycle model into the Rational-based Uniform Software Engineering model (RUSE) [Wo07]. The meta-model aims at two goals, and further evaluation is required to determine to which degree they are satisfied. First the integration between process and product model should enable a complete traceability from tasks to their objects. Second in the meta-model every artifact in the project can be potentially part of the process model, which should allow to seamlessly shift the applied software lifecycle.

1 Introduction

Triggered by the still ongoing debate between traditional and agile approaches, the search space for choosing a software lifecycle model has been expanding. The recommended practice is to select and tailor a particular methodology for a software project according to specific criteria. Glass, for example considers the following four criteria [Gl02]: size, application domain, criticality and innovativeness. Agile methods like Scrum [Sc07] or XP [Be00] are targeted at small to medium sized projects, whereas highly critical projects tend to be managed by risk-oriented methodologies like the Spiral model [Bo88]. If the values of these criteria are considered to be constant for the duration of the project (e.g. [KV00]), the choice of a software lifecycle model can be made at the beginning of a project. But we believe these criteria cannot be considered as constant, and therefore that a shift of the software lifecycle model must be supported. We propose a meta-model, which integrates the software lifecycle model into the Rational-based Uniform Software Engineering model (RUSE) [Wo07]. As a consequence tasks become completely traceable from their origin to their objects. Using

this meta-model every artifact and the associations between them can be part of the process model. We claim this allows modeling every software lifecycle. This is a precondition for a tool-supported shift of the software lifecycle model. Further techniques to handle change known from the field of product modeling like software configuration management can also be applied to the process model.

2 The unified Meta-Model

This section describes the unified meta-model, which integrates process modeling into the existing Rational-based Uniform Software Engineering model (RUSE) [Wo07]. RUSE supports distributed software engineering projects in system modeling, collaboration and organization. The model is implemented in a tool suite called Sysiphus [Br06]. The key idea of RUSE is the combination of different software engineering models into a unified model. Every element in this unified model is a ModelElement. The RUSE model supports three different categories of ModelElements: SystemElement, CollaborationElement and OrganizationElement. SystemElements are used to model the system under construction on different layers of abstraction. Most of the SystemElements are based on the Unified Modeling Language (UML) [O04], but RUSE also supports other abstractions like the UserStory used in XP [Be00]. CollaborationElements capture the communication and collaboration of users and are mainly based on the QOC [Ma96] model. CollaborationElements range from comments, action items, and risks to milestones and iterations. OrganizationElements describe the organizational structure of a software development project and are used to model organizational units such as teams and participants and their associations. Associations between different types of ModelElements are modeled as links between them. Links can relate two model elements inside a model diagram, for example two classes in a UML class diagram. But links can also relate two model elements from different diagrams, for example a link connecting a developer to a subsystem he is working on. RUSE is extensible, that is, new types of ModelElement and links can be added to the meta-model. For example, a new software process can be introduced by adding new model elements. To support Scrum [Sc07], a class Sprint has been added [Sc06]. This extensibility makes RUSE the ideal basis for integrating different software lifecycles. One could be tempted to add a new category for every software lifecycle, but the challenge is to interrelate different existing (and emerging) process models within RUSE. This is a precondition to shift between different software lifecycle models at “project runtime”. Therefore we define a meta-model for RUSE extracting the commonality of process models.

In RUSE only a few ModelElements like Issue or ActionItem have status information. Different methodologies use a variety of elements to represent the status of a project. We distinguish two different types: First elements with an explicit status, which can be declared and second elements with an implicit status, which is dependent on other elements. Typical examples of the first type are checklist items like “Sprintlog Items” in Scrum [Sc07] or “Risks” in the Spiral model [Bo88]. An example for the second type is a phase, which is dependent on the status of its containing activities. As the unified

meta-model must be able to model every methodology we extend the RUSE meta-model: Every ModelElement can now have a status, which can be observed with a specific perspective. In addition to the two states “open” and “closed”, we add the status “blocked” to indicate the dependency of an element on the completion of another element. We therefore extend the RUSE ModelElement with an attribute, which can be closed, open or blocked. We call this attribute COB (as an acronym for Closed Open Blocked). The acronym helps us to distinguish the new attribute from already existing attributes like status or state, which are already implemented by several subtypes of ModelElement (like the Issue). The meta-model allows, that the status of elements is dependent on other elements. Therefore the meta-model has to extract the communality of the dependencies used in different methodologies. There are types of dependencies called BlockingDependency and OpeningDependency. BlockingDependencies are used in activities that require sequential execution, for example the activities in the waterfall model are connected by BlockingDependencys. An example for an OpeningDependency in a Scrum-based project is the association between SprintBacklog and SprintBacklogItems. The status of the SprintBacklog is depending on the status of the included SprintBacklogItems. The SprintBacklog is open as long as there are open SprintBacklogItems in it. In the following we will define the effect of BlockingDependencies and OpeningDependencies in the meta-model. Dependencies connect two ModelElements. If two ModelElements are connected by a BlockingDependency we call one of them Blocker and the other Blocked. If the dependency is an OpeningDependency, we call the ModelElements Opener and the other Opened (see Figure 2). Opener and Blocker are also called Source, and the term target is used for Opened and Blocked. If the source of a dependency is open or blocked we call the dependency active. The COB of a Blocked and an Opened is affected by these dependencies if they are targets of active dependencies:

- One or more active OpeningDependency: The Opened is open.
- One or more active BlockingDependency: The Blocked is blocked.
- Both: The target is blocked.

In the meta-model the dependencies are not restricted to specific elements, as every element can be part of the process. Therefore even dependencies from the system model can be used for process modeling. Figure 1 shows an example for the effect of dependencies between various elements. A Task is connected to an Activity with an OpeningDependency as the Activity is open until all included tasks are closed. As long as this Task is open, the dependency is active. The Task also opens a Subsystem, the object of the task. This Subsystem is connected to second Subsystem by a BlockingDependency, because the second Subsystem depends on it. As this dependency is also active, the second Subsystem is blocked.

Figure 1: Example for the effect of dependencies

ModelElements in RUSE are connected by ModelLinks. There is a variety of existing types of ModelLinks, e.g. issues can be connected to sub issues. To integrate the dependencies in RUSE we don't necessarily introduce new ModelLinks, but we classify the existing types. Every ModelLink can either be an OpeningDependency, a BlockingDependency or NoDependency (default). To model Scrum, for example, we classify the link between a sprint backlog item and a sprint backlog as an OpeningDependency, with the sprint backlog item as source. As a consequence, the sprint backlog will be open as long as any of its sprint backlog items is open or blocked.

ModelElements are closed as long as they are not target of an active dependency. Also it is not allowed to change the COB explicitly. Therefore we need another type of element to indicate that there is open work to do in a project. We call these elements Checkables, the smallest units in the work breakdown structure of a project. The complexity of Checkables should be on a level, where project members are able to decide explicitly if they are closed. Checkable is a subclass of model element (see Figure 2) inheriting the attribute COB and the two types of dependencies. Checkable extends ModelElement by the attribute checked which is manipulated by the methods check() and uncheck(). The attribute checked affects the COB of the checkable in the following way: If a Checkable is unchecked and is not target of active dependencies it is open. In any other case the Checkable COB is handled the same way as in the super class ModelElement. Typical examples for Checkables in RUSE are ActionItems, Issues or Risks.

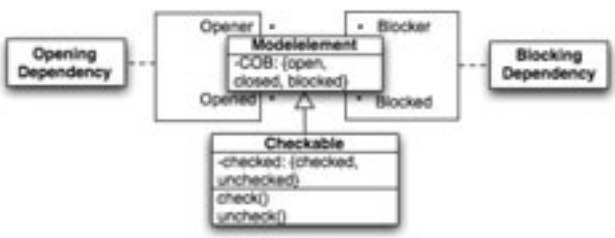


Figure 2: The meta-model

3 Examples for using the meta-model

In this section we will describe some examples how existing software lifecycle models can be modeled as instances of the introduced meta-model. We start with an activity-oriented perspective of a sequential waterfall model consisting of the activities RequirementsAnalysis, SystemDesign and ObjectDesign (see Figure 3)[1]. The activities block each other, thus they are linked by a BlockingDependency. Every activity results in a corresponding document. An OpeningDependency connects the activities to their

[1] In the figures we use highlighting schema to indicate the value of the COB attribute: Closed ModelElements are shown as white boxes, open instances are drawn in grey and blocked instances are are shown in black. Furthermore we highlight OpeningDependencies in grey and BlockingDependencies in black. Boxes with numbers denote arbitrary ModelElements.

respective documents. This implies, that the documents are open until the corresponding activity is closed. In this example, the SystemDesign activity and the ObjectDesign activity are still blocked. In the RequirementsAnalysis activity some work still needs to be done (see items “1”, “2” and “3”). These ModelElements are connected to the activity by an OpeningDependency. Therefore the activity remains open until all these ModelElements are closed.

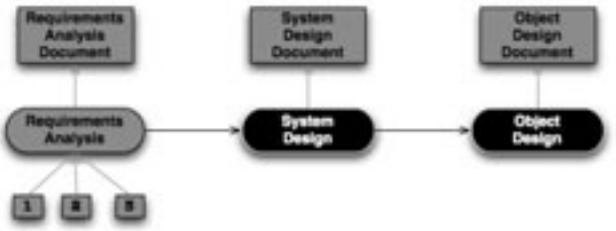


Figure 3: Waterfall model

In the next example (see Figure 4) we model a project using a Scrum process. We introduce more ModelElements to represent a progressed project. Also we introduce dependencies between these ModelElements, e.g. “3” blocks “5”. For example this can represent, that a component of a system cannot be realized until another component is finished. The Repository contains all ModelElements. The ProductBacklog in Scrum contains all tasks of a project over time. We model this by connecting every Modelelement in the repository to the ProductBacklog by an OpeningDependency. The ProductBacklog perspective shows all open ModelElements of a project, which are not yet connected to a sprint. We introduce two sprints and connect them by a BlockingDependency. Sprint2 is blocked until Sprint1 is closed. To plan sprints we connect selected ModelElements to the according sprint by an OpeningDependency. The event-oriented perspective shows the open elements of every sprint.

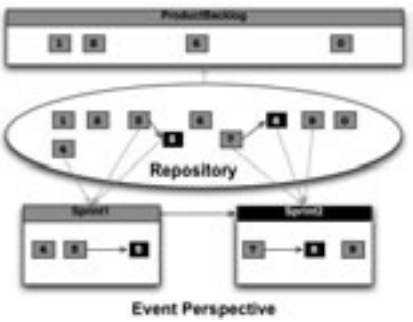


Figure 4: An instance of the meta-model modeling Scrum

Processes are modeled by connecting ModelElements in the repository to elements like activities, sprints or documents. This allows for developers to apply more than one process in the same project, even if they are working on the same repository. We demonstrate this in Figure 5: We use two processes, the waterfall model from the first example and a Scrum process. A developer can now seamlessly switch between Scrum and the waterfall process model since they are only two different perspectives on the same repository. This allows for interesting combinations. For example, the project can be internally managed with Scrum, while the manager can present the status of the project using the waterfall model perspective to a client. In Figure 5 all ModelElements of Sprint1 are already closed, which means that Sprint1 is also closed. Therefore Sprint2 is not blocked anymore, but open.

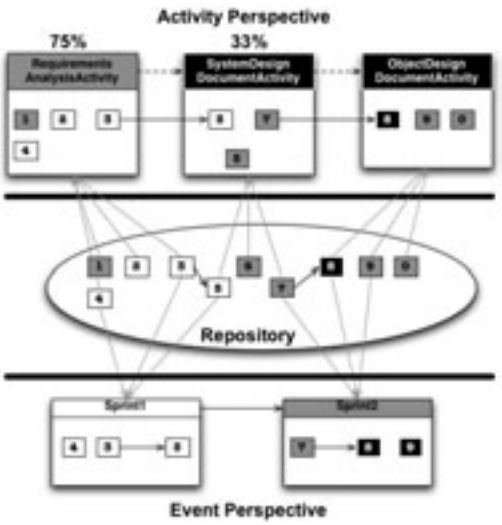


Figure 5: Modeling waterfall model and Scrum based on the same repository

5 Future work

The unified meta-model is currently being implemented in the Sysiphus tool-suite. Our next step is the creation of model-templates for the most common methodologies and perspectives. We already have experiences with projects, which evolutionary tailored their process from a traditional to an agile process. But this was done without the support of a process model like we suggest in this paper and without explicit tool-support. Beginning in October 2007 we have done an empirical evaluation of our ideas using a case study based on a large student project with an industrial partner. We will try to model a process dynamically retailored from a waterfall model to a Scrum oriented approach over the project duration.

Bibliography

[Be00] Beck, K.: Extreme Programming Explained, Addison-Wesley, 2000.

[Bo88] Boehm Barry W. A Spiral Model of Software Development and Enhancement, IEEE Computer, IEEE, New York, Mai 1988

[Br06] B. Bruegge, A. H. Dutoit, and T. Wolf. Sysiphus: Enabling in formal collaboration in global software development. In Proceedings of the First International Conference on Global Software Engineering, October 2006.

[Gl02] Glass, Robert L. "Agile Versus Traditional: Make Love, Not War!" in "The Great Methodologies Debate: Part 1", *Cutter IT Journal*, Vol. 15 No. 1 (January 2002)

[KV00] Khalifa M. and Verner June. Drivers for Software Development Method Usage, IEEE Transactions on engineering management, IEEE, New York, August 2000

[Ma96] MacLean A., Young R.M., Bellotti V. and Moran T. "Questions, options and criteria: Elements of design space analysis", Human-Computer Interaction, Vol. 6, 1996.

[O04] OMG Unified Modeling Language Specification Version 2.0, May 2004.

[Sc07] Scrum Alliance, URL: http://www.scrumalliance.org, 1. September 2007

[Sc06] Schiller, Jennifer. *Scrum in the unified project model*. Diploma thesis, Technische Universität München, September 2006

[Wo07] Timo Wolf. *Rational-based Unified Engineering Model*. Dissertation, Technische Universität München, July 2007

Integration des V-Modell®XT im Visual Studio Team Foundation Server – Erfahrungen aus dem Projekt CollabXT

Marco Kuhrmann

Technische Universität München
Institut für Informatik – Software & Systems Engineering
Boltzmannstr. 3
84748 Garching b. München, Germany
kuhrmann@in.tum.de

Abstract: Das V-Modell XT ist aufgrund seines formalen Metamodells als Basis für eine umfassende Werkzeugunterstützung prädestiniert. Bereits im Standardmodell sind Werkzeuge enthalten, die die Anpassung des Prozesses auf der Organisations- und der Projektebene unterstützen. Eine weitergehende Unterstützung, die eine unmittelbare Überführung in eine Projektarbeitsumgebung gestattet, ist im Standard jedoch nicht vorgesehen. Im Rahmen des Projekts CollabXT[1] (www.collabxt.de) wurden die Standardwerkzeuge erweitert. Nach der projektspezifischen Anpassung des V-Modells ist nun auch eine aufbauende, werkzeugspezifische Anpassung möglich. Dieser Beitrag stellt die Ergebnisse eines Teilprojekts vor und erläutert Vorgehensweisen, Optionen, Probleme und offene Punkte.

1 Einleitung

Dem V-Modell XT [VMXT] hängt oftmals der Ruf des V-Modells 97 an, schwerfällig, bürokratisch oder realitätsfern zu sein. Oft sind solche Aussagen darauf zurückzuführen, dass die Anwender nur die *Dokumentsicht* auf das V-Modell haben. Dabei bietet es umfangreiche Möglichkeiten an, um in Werkzeuge integriert zu werden. Eine solche Integration ist nicht nur wünschenswert, sondern für die Akzeptanz auch *erforderlich*. Gerade für das V-Modell wird seitens der Anwender die Verfügbarkeit von unterstützenden Werkzeugen begrüßt. Sie bieten Optionen an, vorgeschriebene Tätigkeiten und Prozesse zu unterstützen und durch Automatisierung den Anwender von nicht-kreativen Alltagsaufgaben zu entlasten.

Insbesondere im Umfeld von kleinen Projekten stehen Entwicklung und die dazu benötigten Werkzeuge im Zentrum. Wird ein vom Kunden dann Prozess verlangt, scheiden

[1] Dieser Beitrag ist fokussiert auf das Teilprojekt CollabXT-TFS, in dem das V-Modell XT mit dem Visual Studio Team Foundation Server kombiniert wurde. Dieses Teilprojekt ist ein Kooperationsprojekt zwischen der Technischen Universität München, Microsoft Deutschland und der Zühlke Engineering GmbH gewesen.

Prozesse wie das V-Modell oftmals von vorn herein wegen der o.g. Vorurteile aus. Hier werden agile Methoden wie XP [Be03] oder Scrum [Sch07] bevorzugt. Durch seine Anpassungsoptionen bietet sich das V-Modell aber auch für KMUs (Kleine und Mittelständische Unternehmen) an. Eine Zusammenführung von schnellen und kompakten Entwicklungsprojekten, etablierten Werkzeugen und dem V-Modell XT ist die Herausforderung, mit der auch das V-Modell XT seine Eignung für kleine Projekte nachweisen kann.

1.1 Das V-Modell XT und seine Werkzeuge

Werkzeuge sind wichtig für die Akzeptanz eines Vorgehensmodells. Insbesondere komplexe Prozesse, wie die organisations- oder projektspezifische Anpassung erfordern eine Werkzeugunterstützung. Das V-Modell bietet hierfür im Standard zwei Referenzwerkzeuge [KH+07], den *V-Modell XT Editor* und den *V-Modell XT Projektassistenten*, an. Die Werkzeuge unterstützen Prozessingenieure und Projektleiter. Ersteres bei der organisationsspezifischen Anpassung des V-Modells; letzteres bei der projektspezifischen Anpassung (Tailoring) sowie der initialen Planung und Erzeugung von Dokumentvorlagen. Die Einführung eines V-Modells wird hingegen nicht direkt unterstützt, ebenso wenig die Überführung eines projektspezifischen V-Modells in eine konkrete Projektumgebung. Hier müssen entweder integrierte Softwarepakete (z.B. in-Step von micro-TOOL, www.microtool.de) zu Einsatz kommen oder das projektspezifische Vorgehen *irgendwie* in die alltägliche Arbeit integriert werden.

1.2 Integrations- und Automatisierungsoptionen

Dabei ist die Integration, bzw. die teilautomatische Überführung des V-Modells nach dem Tailoring technisch gut umsetzbar. Das V-Modell baut durch seine Referenzwerkzeuge bereits erste Stufen einer Werkzeugkette auf (Abbildung 1, (1) und (2)).

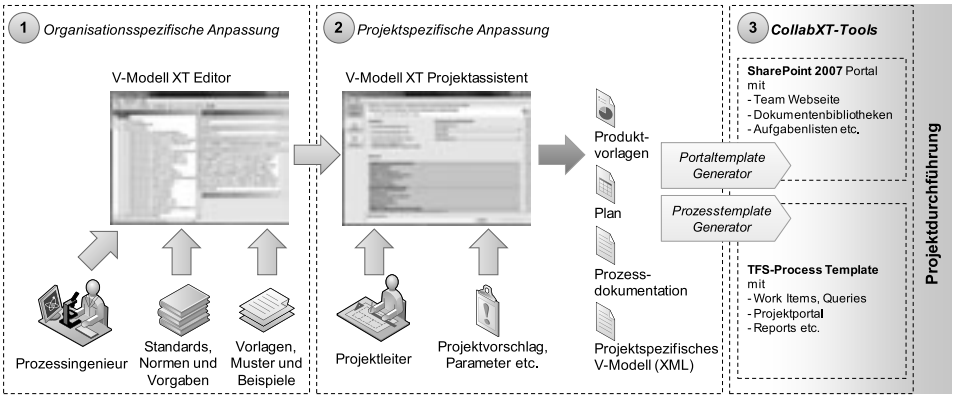


Abbildung 1 Werkzeugkette des V-Modell XT und Erweiterung durch CollabXT-Werkzeuge

Der Editor stellt ein V-Modell Derivat zur Verfügung, das der Projektassistent weiter anpassen und zur Planung verwenden kann. Weiterhin kann der Projektassistent ein ange-

passtes V-Modell exportieren, welches der Editor weiter bearbeiten kann. Das Datenaustauschformat ist durch das *V-Modell XT Metamodell* definiert. Dieses kann auch herangezogen werden, um ausgehend von einem projektspezifischen V-Modell eine weitergehende, werkzeugspezifische Anpassung vorzunehmen, wie es in Abbildung 1, (3) zu sehen ist. Die verschiedenen Optionen hierzu werden in [KK+08] detailliert erläutert.

1.3 Aufbau und Beitrag

Dieser Beitrag stellt kurz die Ergebnisse des Projekts *CollabXT* vor. Er beschreibt in Kapitel 2 das gewählte Zielwerkzeug *Team Foundation Server* (TFS) und die Optionen der Prozessunterstützung dieses Tools. Im Kapitel 2.3 gehen wir auf die Abbildung des V-Modells auf TFS ein. In Kapitel 3 betrachten wir kurz die Projektergebnisse. Kapitel 4 fasst diese mit einer Bewertung und neu aufgekommenen oder noch offenen Fragen zusammen und gibt einen Ausblick.

2 Integration des V-Modell XT in Team Foundation Server

Der Visual Studio *Team Foundation Server* (TFS) ist die Backend-Komponente für das Visual Studio Team System [GP05, Mic07]. Er stellt Umgebung verschiedene Serverdienste bereit: Work Item Tracking, Dokumenten- und Quellcodeverwaltung sowie Reporting. TFS baut auf SQL- und SharePoint-Server auf und bietet für weitere Clientwerkzeuge eine .NET- und eine Web Service Schnittstelle an. Diese werden bspw. durch die Plug Ins für MS Office und das Visual Studio genutzt. TFS kann somit direkt mit MS Project, Word oder Excel gekoppelt werden. Durch die Bereitstellung einer projekteigenen, SharePoint-basierten Teamwebseite, die neben der Prozessdokumentation auch alle Dokumente (Anforderungen, Berichte etc.) bereitstellt, unterstützt TFS auch große, verteilt ablaufende Projekte.

2.2 Process Templates, Work Items und weitere TFS-Prozesselemente

Da sich Projekte hinsichtlich vieler Parameter teils deutlich unterscheiden, muss TFS für die Unterstützung verschiedener Prozessmodelle eine hohe Flexibilität bieten. TFS unterstützt beinahe beliebige Prozesse. Er definiert ein Format, in dem die Prozesse beschrieben und hinterlegt werden müssen – sog. *Process Templates*[2]. Ein Process Template enthält alle für einen Prozess erforderlichen Beschreibungen und Vorlagen ([KK+08] geht detailliert darauf ein). Zentral sind *Work Items* und die *Reports*. Erstere bilden überwachbare Prozesselemente eines Projekts ab. Dies können z.B. Anforderungen, Risiken oder ähnliches sein. Work Items enthalten neben Daten noch einen *Workflow*, mit dem sich verschiedene Prozesse beschreiben lassen. Work Items können untereinander verknüpft sein, um bspw. Abhängigkeiten oder Beziehungen zu modellieren. Reports stellen verschiedene Sichten auf ein Projekt zur Verfügung und können über den Datenbanken des TFS Auswertungen erstellen. [GP05] zeigt hier viele prägnante Beispiele.

[2] Auf den Webseiten von Microsoft finden sich bereits verschiedene Templates, z.B. für XP oder Scrum.

2.3 Abbildung des V-Modells

Da TFS verschiedene Prozesse unterstützt, stellt auch das V-Modell XT eine Option dar. Jedoch unterscheidet sich das V-Modell in vielen Bereichen von den anderen, bislang durch TFS unterstützten Prozessen. Das V-Modell ist *ergebnisorientiert*, wohingegen das konkrete Entwicklungsprojekt im Visual Studio aufgabenorientiert ist. TFS fokussiert daher Work Items, die verschiedene Projektaufgaben repräsentieren. Work Items setzen auf integrierte Workflows, ein Konzept, das das V-Modell nicht modelliert. [KK+08] stellt detailliert die Identifikation und Modellierung von TFS Prozesselementen auf der Basis des V-Modells vor. Wesentlich dabei ist die Bereitstellung spezieller Work Item Typen, die wichtige Elemente des V-Modells im TFS abbilden:

- Produkte, Aktivitäten und Entscheidungspunkte
- Arbeitsaufträge
- Risiken und Maßnahmen
- Problem- und Änderungsmeldungen

Die Work Item Typen für Produkte, Aktivitäten und Entscheidungspunkte bilden die zentralen Metamodellelemente des V-Modells im TFS so ab, dass Produktstatus, sowie Erfüllungsgrade von Aktivitäten messbar und überwachbar sind. Auf der Basis von Entscheidungspunkten können Produkte und Aktivitäten verknüpft und gruppiert werden. Entscheidungspunkte dienen gleichzeitig zur Planung, da sie die Vorlage für einen Meilensteinplan bilden. Die anderen Work Item Typen bilden Prozesse aus dem V-Modell ab und ersetzen einige Vorgaben zu Managementprozessen wie z.B. Aufgaben-, Risikomanagement sowie Problem- und Änderungsmanagement durch konkrete Prozesse.

3 CollabXT-TFS: Erweiterung der V-Modell XT-Werkzeugkette

CollabXT-TFS erweitert die Werkzeugkette der V-Modell Referenzwerkzeuge, wie in Abbildung 1 gezeigt. Zur Anwendung kommt eine Kombination aus Template- und Generatorsystem. TFS beschreibt seine Prozesse in *Process Templates*, die die wesentlichen Strukturen in XML beschreiben. Das V-Modell liegt vollständig in XML vor. CollabXT-TFS fertigt daher ein TFS *Meta-Template* vor (Abbildung 2), in dem alle für das V-Modell XT relevanten Strukturen vorgefertigt werden. Dieses V-Modell TFS Meta-Template und ein projektspezifisch angepasstes V-Modell XT sind Eingaben für den *CollabXT-Generator*. Dieser Generator füllt das Meta-Template wie folgt:

- Aus Produkten, Aktivitäten und Entscheidungspunkten werden auf Basis der initialen Planung *Default Work Items* erstellt.
- Auf Basis der gewählten Projektdurchführungsstrategie(n) und der initialen Planung wir eine Projektstruktur (Areas, Iterations) erzeugt.
- Die im Tailoring ermittelten Rollen stehen nach der Instanziierung des Templates als Systemrollen zur Verfügung.
- Die durch den Projektassistenten generierten Produktvorlagen werden in der SharePoint Dokumentenbibliothek integriert.
- Die generierte HTML-Dokumentation wird in das SharePoint Template integriert.

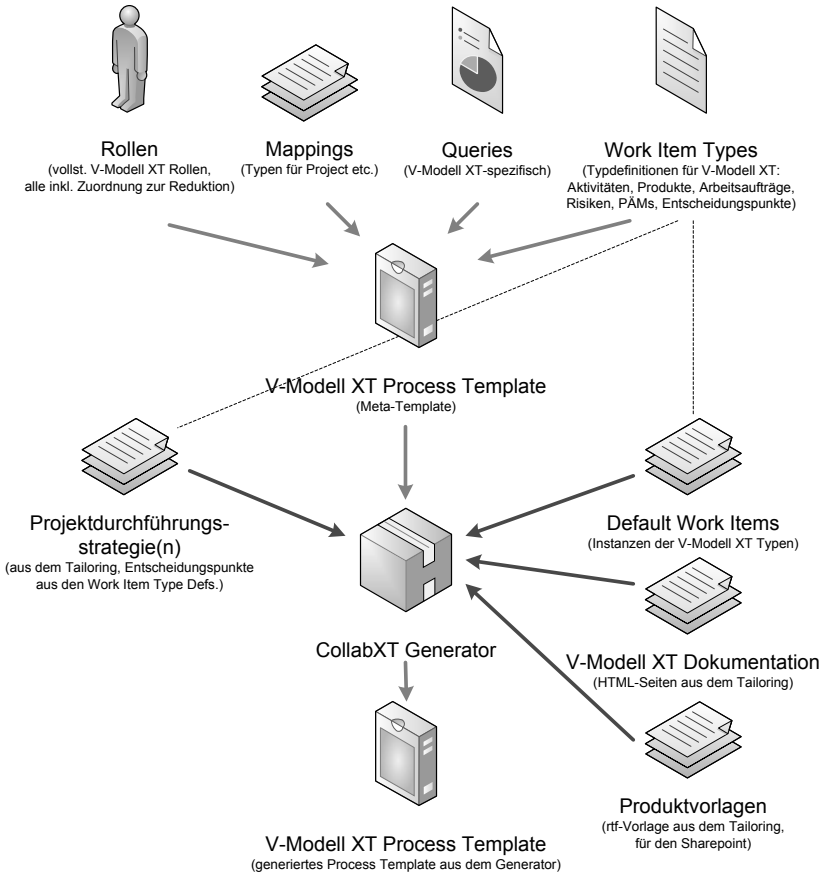


Abbildung 2 Ein- und Ausgaben für CollabXT-TFS (aufgeschlüsselt nach Typen)

4 Bewertung und Ausblick

Die Erfahrung zeigt, dass die weitergehende Anpassung und Anwenderunterstützung ein für das V-Modell XT essenziellen Akzeptanzkriterium darstellt. CollabXT-TFS stellt hierfür einen einfachen Automatisierungspfad zur Verfügung. Das V-Modell XT wird direkt in die Software Entwicklung überführt.

Das finden einer Abbildungsvorschrift gestaltete sich als große Herausforderung, da TFS und V-Modell XT zwei verschiedene Sichten auf ein Software Projekt haben. Hier mussten bei der Abbildung verschiedene Daten (z.B. Aktivität) und Prozesse (z.B. Risikomanagement) neu entworfen werden, da sie das V-Modell schlichtweg nicht explizit anbietet. Auf der anderen Seite mussten V-Modell Entitäten unter der Prämisse remodelliert

werden, dass TFS keine hierarchischen Work Items unterstützt. Strukturen wie *Produkt-Thema* sowie *Aktivität-Teilaktivität* sind daher nicht 1:1 abbildbar. Weiterhin differieren die Anforderungen an Rollen: das V-Modell beschreibt durch Rollen Zuständigkeiten, während TFS sie für das Sicherheits- und Berechtigungssystem heranzieht. Aufgrund der Sicherheitseinstufung über Areas und Iterationen ist eine vollautomatische Abbildung nicht möglich.

CollabXT-TFS stellt einen ersten produktiv einsetzbaren Generator zur Verfügung. Dieser erstellt auf der Basis eines projektspezifisch angepassten V-Modells ein valides Template, das alle relevanten Größen des V-Modells und enthält. Es enthält alle relevanten Dokumente, Queries und Reports, die ein teilautomatisiertes Berichtswesen (z.B. Statistik der Arbeitsaufträge, Teststatistik etc.) unterstützen. Durch die aus der initialen Planung generierten Default Work Items ist darüber hinaus auch ein sofortiger Projektstart inkl. Planung möglich. Die Lücke zwischen dem Prozess und dem Projekt ist durch die Generatorsoftware geschlossen worden. Dennoch gibt es offene Punkte, die noch optimiert, bzw. nach einer Pilotierung noch einmal überarbeitet werden müssen. Ein besonders kritischer Punkt ist die Kombination der Prozesslebenszyklen. Nachdem ein Process Template erstellt wurde, bleibt dieses noch bearbeitbar – in Teilen sogar im laufenden Projektbetrieb auf einem TFS. Eine Nachbearbeitung kann aber im schlimmsten Fall zu einem inkonsistenten Vorgehen und somit zum Verlust der V-Modell-Konformität führen.

Die Integration entlastet Anwender an vielen Stellen, dennoch *muss* eine entsprechende Prozesskompetenz vorhanden sein. TFS kann ein Projekt, ebenso wie das V-Modell, allenfalls unterstützen.

Literaturverzeichnis

[Be03] Beck, K.: *Extreme Programming*, Addison-Wesley, 2003.

[GP05] Guckenheimer, S., Perez, J. J.: *Software Engineering with Visual Studio Team System*, Addison Wesley, 2005.

[KH+07] Kuhrmann, M., Hammerschall, U., Ternité, T., Sihling, M.:, *Individueller Standard - Das V-Modell XT an Unternehmen anpassen (Teil 1)*, iX-04/07, 2007.

[KK+08] Kuhrmann, M., Kalus, G.: *Werkzeugspezifisches Tailoring für das V-Modell XT – CollabXT-SP, CollabXT-TFS – Projekt- und Ergebnisbericht*, Technischer Bericht, Technische Universität München, erscheint in 2008.

[Mic07] Microsoft Corp.: *Team Development with Visual Studio Team Foundation Server*, Microsoft Press, 2007.

[Sch07] Schwaber, K.: *Agiles Projektmanagement mit Scrum*, Microsoft Press, 2007.

[VMXT] *V-Modell XT Web Portal* des Innenministeriums: www.v-modell-xt.de.

Integrationsnotwendigkeit im agilen Zeitalter

Jennifer Schiller

CT SE 3
Siemens AG
Otto-Hahn-Ring 6
81379 München
jennifer.schiller.ext@siemens.com

Abstract: Die Einführung agiler Methoden und Werte hat der Softwareentwicklung viele Vorteile gebracht. Um eine agile Entwicklungsabteilung bestmöglich in das bestehende Unternehmen einzubinden, müssen alle involvierten Einheiten berücksichtigt und die Schnittstellen angepasst werden. Eine ganzheitliche Sichtweise ist dabei sehr wichtig.

Im Folgenden wird gezeigt, wie sich ein Unternehmen am Beispiel des Qualitätsmanagements auf die geänderten Anforderungen, die eine agile Softwareentwicklung mit sich bringt, einstellen muss, und dass eine optimale Integration der agilen Entwicklung in das Unternehmen und seine Prozesswelt unabdingbar ist.

1 Das Ganze sehen

Der agile Gedanke gewinnt zunehmend an Bedeutung und Beliebtheit. Das Ziel agiler Softwareentwicklung (SE) ist es, den Softwareentwicklungsprozess flexibler und schlanker zu gestalten, als es bei den klassischen Vorgehensmodellen der Fall ist. Das bedeutet, dass eine agile SE nicht nur mit der Anwendung einiger neuer Methoden oder Praktiken einhergeht, sondern eine Restrukturierung und vor allem einen Wertewandel im Unternehmen hervorruft. Je größer und komplexer das Unternehmen ist, desto schwieriger ist es, diese Veränderung der Methoden und Werte und der damit verbundenen Umstrukturierungen und Prozessanpassungen zu vollziehen.

Das typische Vorgehen bei Einführung eines agilen Vorgehens beschränkt sich in der Regel zunächst auf die Phasen der SE, um die Vorteile des agilen Ansatzes, wie Flexibilität, iterative und kürzere Entwicklungszyklen und Test Driven Development (TDD), auszunutzen. Angelehnt an das Lean Software Prinzip "See the Whole" [PP03] muss ein agiles Entwicklungsprojekt aber im Kontext des gesamten Unternehmens betrachtet werde, um eine Optimierung der Wertschöpfungskette zu erreichen. Daraus folgt, dass alle betroffenen Abteilungen zu berücksichtigen sind und die agile Entwicklungsabteilung optimal in das Unternehmen integriert werden muss, um eine ganzheitliche Optimierung des Gesamtprozesses, verringert Reibungsverluste, und dadurch reduzierte Kosten und positive Auswirkungen auf die Qualität zu erzielen.

Welche Schnittstellen von einer solchen ganzheitlichen Integration betroffen sind, und wie die Prozesse aussehen, um eine agile Entwicklung optimal in die bestehende Prozesslandschaft einzubinden, soll im Weiteren exemplarisch gezeigt werden.

2 Die Softwareentwicklungsabteilung und ihre Schnittstellen

Die SE ist in einen Prozess mit vor- und nach gelagerten Aktivitäten eingebettet, in dem sie mit anderen Unternehmensbereichen zusammenarbeitet, Input bekommt, oder Output liefert. Es wurden diverse Schnittstellen zwischen der SE-Abteilung und anderen Unternehmensbereichen identifiziert. Untersucht wurde dabei, welche Abteilungen Beziehungen zur Entwicklungsabteilung (inkl. Test, Dokumentation und Releaseerstellung) haben bzw. auf welche Bereiche die Entwicklungsabteilung einen Einfluss ausübt. Das Ergebnis dieser Auflistung umfasst die folgenden Unternehmensbereiche:

- Business Management
- Human Ressources
- Innovation Management
- Marketing & Sales
- Partnerprodukte & andere Komponenten
- Product & Portfolio Management
- Quality Assurance
- Service & Customer Relationship Management
- Supply Chain Management

Diese sind in der folgenden Abbildung zusammengefasst.

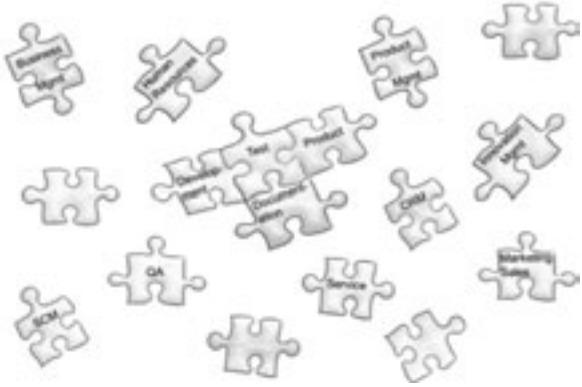


Abbildung 1: Softwareentwicklung und angrenzende Unternehmensbereiche

Wie in einem Puzzle müssen auch die Schnittstellen und Abläufe zwischen den Bereichen eines Unternehmens optimal und reibungslos zusammenpassen. D.h. es muss ein ganzheitlicher Prozess entwickelt werden, der das agile Entwicklungsprojekt bestmöglich in den bisherigen unternehmensweiten Prozess integriert. Wie solch eine Zusammenarbeit aussieht, welche Synchronisationspunkte es gibt und was sich an dieser Schnittstelle ändert, wenn die Entwicklung agile Methoden anwendet, soll im Folgenden beispielhaft an der Qualitätssicherung gezeigt werden.

3 Qualitätsmanagement in der Softwareentwicklung

3.1 Definition: Qualität & Qualitätsmanagement

Die Norm DIN EN ISO 9000:2005 definiert Qualität als den „Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt." Im Kontext der SE werden unter dem Qualitätsbegriff häufig verschiedene Dinge verstanden. Abhängig von der Sichtweise kann nach Anwenderzufriedenheit, Designqualität, Nachhaltigkeit, produktbezogene Qualität, Prozessqualität und Software Ergonomie unterschieden werden. Eine Priorisierung ist abhängig vom jeweiligen Standpunkt, d.h. Entwickler achten auf Klarheit des Designs und wartbaren Code, Projektleiter auf die Einhaltung des Projektprozesses, Anwender auf vollständige Umsetzung ihrer Anforderungen und Auftraggeber auf Budget- und Termintreue. Um sich innerhalb dieser vielfältigen Ziele auszurichten muss ein Qualitätsmanagement (QM) im Unternehmen eingeführt werden, das die Qualität systematisch steuert. QM wird bei CMMI [SEI] als die Sicherstellung, Überwachung und Verbesserung von Prozess- und Produktqualität verstanden und hat folgende Inhalte:

- Objektive Bewertung der angewendeten Prozesse, Arbeitsergebnisse und Services gegen die beschriebenen Prozesse, Standards und Verfahren.
- Identifizieren von "Nichteingehaltenen Vorgaben"
- Den Projektmitarbeitern und dem Management Feedback über die Ergebnisse der durchgeführten Qualitätssicherungsmaßnahmen geben
- Sicherstellen, dass die "Nichteingehaltenen Vorgaben" adressiert sind

Zu den Qualitätsmerkmalen nach ISO/IEC 9126 zählen Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.

Eine Qualitätsmanagementabteilung ist also notwendig, um die verschieden Sichtweisen bzgl. Qualität zusammenzubringen und die ganzheitliche Sicht auf Qualität zu wahren.

3.2 Brauchen wir überhaupt QM im agilen Zeitalter?

Die Aufgabe des QMs ist Prozess- und Produktqualität sicherzustellen. Die agilen Verfahren zeichnen sich allerdings dadurch aus, dass sie von ihren Ansätzen her bereits eine Qualitätsorientierung integriert haben und somit auf die Entwicklung qualitativ hochwertiger Software ausgelegt sind. Sie bieten Methoden und Praktiken, die zu „qualitätsinfizierten" Agilisten führen. Zu diesen agilen, qualitätsorientierten Praktiken zählen z.B. Refactoring, TDD, wobei Tests auch traditionelle Artefakte ersetzen können, Agile Model Driven Development (AMDD) und Retrospektiven.

Ein Verzicht auf ein explizites und bewusstes QM würde jedoch eine wesentliche Einschränkung bedeuten, denn ohne explizite Qualitätssicherung würde sich der jeweils subjektive Qualitätsbegriff der Beteiligten durchsetzten und Interessenskonflikte wären zu erwarten. Außerdem ist eine rein implizite Qualitätsorientierung nur für ein Projekt wirksam und trägt nicht zum Fortschritt der Gesamtorganisation bei. Durch die größere

Gewichtung der Qualität wird sich die Sichtweise und Rolle der Qualitätsverantwortlichen für agile Softwareentwicklungsprojekte allerdings ändern müssen.

[Am05] beschreibt erste Folgen der agilen Software Entwicklung für das QM:

- mehr Qualität impliziert weniger Bedarf für Qualitätssicherungsaktivitäten (wie z.B. Reviews und Prüfungen)
- alle müssen sich an „unvollständige" (in-Arbeit) Artefakte gewöhnen. Modelle, Dokumente und der Quelltext entwickeln sich über die Zeit.
- Der agile Entwickler ist ein „generalizing specialist" (mit einem oder mehreren Spezialgebieten, damit er dem Team einen Wert hinzufügen kann, aber auch mit einem generellen Verständnis der Softwareprozesse und des Geschäftsdomains, zur effektiv Zusammenarbeiten im Team.)
- Qualitätssicherung ist ein ständiger und kontinuierlicher Prozess
- Jeder Entwickler kümmert sich um Qualität, wenn er den agilen Prinzipien folgt
- Kurze Iterationen ermöglichen ein schnelles Feedback

Agil bedeutet also nicht, Qualität über Bord zu werfen und gar nicht mehr zu beachten, sondern dass Qualität anders gemacht wird als in einer traditionellen Entwicklung [Di07]. Die agilen Verfahren bieten viele qualitätssichernden Techniken, trotzdem muss das QM weiterhin die Unternehmensziele und ganzheitliche Sichtweise, sowie Prozess- und Produktqualität sicherstellen. Dies ist besonders wichtig, wenn agile und klassische Teilprojekte zusammenarbeiten. Eine „externe" Qualitätssicherung ist somit auch für agile Projekte unerlässlich.

3.3 Warum passt das klassische QM nicht zu agil?

Ein auf einen traditionellen Prozess ausgerichtetes QM kann nicht ohne weiteres auf ein agiles Vorgehen abgebildet werden. Beispielsweise sind Meilensteindefinitionen nicht für einen iterativen Prozess ausgelegt. Metriken sind ebenfalls auf traditionelle Projekte ausgerichtet und können dadurch unnötigen Aufwand verursachen, wenn im Agilen ‚das Falsche' gemessen wird. Des Weiteren ist das Testvorgehen mit TDD anders, es werden andere Methoden verwendet, neue Rollen sind entstanden, und auch die Aufgabenverteilung sieht anders aus als in klassischen Projekten, so dass sich auch das QM daran anpassen muss. Ferner wird dem klassischen QM nachgesagt, dass es häufig kontraproduktiv sei, da starre Vorgaben ein angemessenes und flexibles Vorgehen im Projekt verhindern, wodurch das QM zum Hindernis statt Hilfsmittel wird. Gerade ein unflexibles Vorgehen widerspricht aber den Prinzipien der agilen Methoden [Man01]. Daraus folgt, dass sich die Art und Weise des QM an die geänderten Anforderungen von agilen Projekten anpassen muss und nicht ohne weiteres von traditionellen Projekten übernommen werden kann.

3.4 Anforderungen an ein agil angepasstes Qualitätsmanagement

In der Literatur wird QM häufig nicht explizit adressiert, sondern in der Regel nur implizit: Qualität entsteht „von selbst", wenn gute Mitarbeiter gut kooperieren und die agilen Prinzipien anwenden. [vgl. Ec04, Co02]

Agiles QM strebt an, die agilen Methoden mit dem traditionellen QM zu verknüpfen, (indem es in die Projektteams das Qualitätsmanagement als explizite Aufgabe hineinträgt und weniger formal gestaltet), mit den agilen Methoden und Prinzipien konform zu sein, Produkt- und Prozessqualität sicherzustellen und gleichzeitig Ergebnisse zu liefert, die über das Projekt hinaus sichtbar sind und zum Organisationslernen betragen. Die QM Maßnahmen sehen also anders aus, aber das Ziel ist das gleiche. Ein Tailoring für agile Projekte ist wichtig. "Qualitätsmanagement wird nicht aus einem formalen Gesamtkonzept heraus in die Projekte getragen, sondern aus den Erfahrungen der Projekte heraus in die Organisation hinein entwickelt." [Bo06]

3.5 Wie sehen agile QM Maßnahmen aus?

Q-Plan
Auch im agilen QM muss ein Qualitäts-Plan aufgestellt werden. Dieser definiert u. a. die zu erstellenden Dokumente (wobei nur die wertgenerierenden Dokumente zu erstellen sind). Ferner müssen die Meilensteine neu definiert werden, um den Iterationen und dem flexiblen Charakter eines agilen Projektes gerecht zu werden. Vor Projektbeginn sollten außerdem Informationsflüsse und Schnittstellen zu anderen Teilprojekten, sowie Iteration Exit Kriterien (z.B. Goals erreicht, Build war erfolgreich, keine offenen Aufgaben, Testabdeckung erreicht, geplante Reviews sind durchgeführt …) definiert werden, zur späteren Einhaltungsüberwachung im Projektverlauf. Überdies ist auch auf die Erstellung eines Testplans zu achten.

Wie Reviews, Tracking, Tests, Reports und Metriken aussehen, muss im Einzelfall definiert werden und sollte änderbar sein (auch während des Projekts). Wichtig ist, dass die QM Maßnahmen in die Iterations-Planung integrieren werden. Die Planung erfolgt im Team, das QM übernimmt die Kontrolle der Einhaltung.

Rollenverteilung
Abhängig von der Projektgröße gibt es weiterhin mindestens einen Qualitätsmanager. Dieser wird z.B. bei Scrum durch zusätzliche "Prozess- und Produktqualitätsgehilfen" unterstützt. So achtet der Scrum Master etwa auf die Prozesseinhaltung. Auch das Team wird viel stärker in die Verantwortung gezogen, hochqualitative Software zu entwickeln (z.B. durch TDD).

Methoden & Best Practices
Agile Verfahren bieten Best Practices um hochqualitative Software zu erstellen (Tests, Metriken, Reports müssen trotzdem geplant, zugewiesen und durchgeführt werden). So muss die QS nicht mehr so viele und feste Vorgaben machen, da vieles bereits durch die agilen Methoden abgedeckt ist. Dass Methoden und Vorgehensweisen im Projekt flexibel und anpassbar gewählt werden können ist für ein agiles Team Voraussetzung. Die Aufgabe des QM besteht nun darin, die angewendeten Methoden und Templates zu überwachen, zur Sicherstellung, dass dieses Vorgehen mit den unternehmensweiten Standards konform ist. Zusätzlich sollen 'tolle' Beispiele und Best Practices der Firma und damit anderen Projekten durch das QM zur Verfügung gestellt werden.

Metriken
Die Metriken im agilen Umfeld unterscheiden sich z.T. sehr von traditionellen Projekten und müssen deshalb für agile Projekte neu definiert werden, damit sie nicht zu "waste" werden. Es wird unterschieden in Produktions-/ Build-Metriken (z.B. Burndown Chart, Velocity, Running Tested Features…), Software Fehler Statistiken und Test Abdeckung.

Tracking
Die Aufgabe des QM ist auch im agilen Umfeld die Kontrolle und Verfolgung von Qualität, z.B. ob die geplanten Tests durchgeführt sind, wie sich die gesammelten Metriken entwickeln, ob die Produktqualität den Anforderungen entspricht und ob sich auch der Prozess weiterentwickelt. Produktqualität wird in den agilen Methoden durch das agile Team adressiert, Prozesseinhaltung durch Retrospektiven und den Scrum Master, der für die Einhaltung des Scrum Prozesses verantwortlich ist. Die Tracking-Aufgabe des QM beschränkt sich somit meist darauf den Projektfortschritt mit anderen Projekten zu vergleichen und zu überwachen, ob z.B. Prozessverbesserungs-CRs aus den Retrospektiven auch umgesetzt werden, sowie die Einhaltung der Quality Gates zu kontrollieren.

Reporting
Eine klassische Qualitätssicherung fordert regelmäßige Berichte zum aktuellen Stand des Projektes. Nachdem diese Berichterstellung dem agilen Team häufig nur Aufwand, aber wenig Nutzen bringt, sollte sich ein externer Qualitätsverantwortlicher darum kümmern. Informationen bekommt er u.a. aus dem Burndown-Chart und Daily Scrum Meeting.

3.6 Agiles Qualitätsmanagement und Scrum

Das vorangegangene Kapitel hat agiles Qualitätsmanagement und seine Maßnahmen definiert. Beispielhaft soll nun gezeigt werden, wie ein agiles Qualitätsmanagement die agile Methode Scrum [SB01] ergänzt, bzw. welche Qualitätsaspekte bereits schon in der Methode integriert sind. Da Scrum eine reine Projektmanagementmethode ist, wird zusätzlich davon ausgegangen, dass die Entwickler ausgewählte XP-Praktiken verwenden.

Abbildung 2 fasst den Scrum-Prozess zusammen und zeigt, in welchen Phasen bereits Qualitätsmaßnahmen in Scrum integriert sind, bzw. welche Aufgabe eine „externe, agilisierte Qualitätsabteilung“ übernimmt.

Sprint 0
Die erste Phase eines neuen Projektes beinhaltet vielfach organisatorische Tätigkeiten, wie das Zusammenstellen des Teams, die Bereitstellung von Hard- und Software, Grobkonzepte, etc. Die Aufgabe einer agilen Qualitätsabteilung umfasst in dieser Phase die Erstellung eines Qualitäts-, sowie eines Test- und Reviewplans. Meilensteine müssen bestimmt, Iterations Exit Kriterien definiert, und Metriken festgelegt werden.

Sprint Planning Meeting
Im Sprint Planning Meeting bestimmt das Team die zu erledigenden Features für die kommende Iteration und legt sein Ziel, qualitativ hochwertige Software zu erstellen, die getestet und dokumentiert ist und einen Kundennutzen erfüllt, fest. Der Qualitätsbeauf-

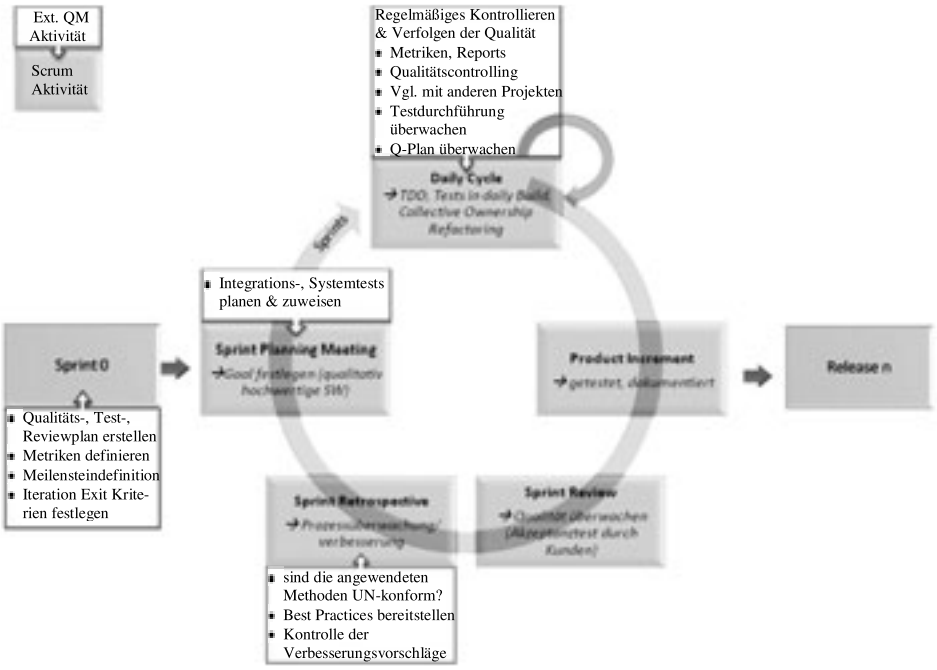


Abbildung 2: Agiles QM in Verbindung mit Scrum

tragte sollte hier speziell darauf achten, dass auch genügend Zeit für Tests (Unit-, System- und Integrationstests) mit eingeplant sind.

Daily Cycle
TDD ist zwar keine reine Scrum-Methode, kann aber sehr gut damit kombiniert werden. Wenn Tests so früh wie möglich auf verschiedenen Ebenen (Unit, Integration, System) bereitgestellt und in den Daily Build integriert werden, können Fehler frühzeitig entdeckt und behoben werden. Außerdem wird das Refactoring erleichtert, das bei kontinuierlicher Anwendung zur Erhöhung der Softwarequalität führt. Eine weitere qualitätssichernde Technik ist Collective Ownership, indem sich alle Entwickler um den Quelltext kümmern und gegenseitig verbessern. Dadurch wird gleichzeitig ein paarweiser Review durchgeführt und Probleme können aufgedeckt werden. Die Aufgaben eines QMs umfassen hier zusätzlich eine regelmäßige Kontrolle der Qualität. Dies beinhaltet die Erstellung von Metriken und Reports, um das Team zu entlasten, notwendige Informationen aber trotzdem bereitzustellen und Probleme frühzeitig aufzudecken, sowie ggfs. die Verfolgung eines zusätzlichen Review- und Testplans.

Product Increment
Scrum legt fest, dass ein Feature, das nicht getestet wurde auch nicht als fertig gilt und daher nicht ausgeliefert wird. Somit wird sichergestellt, dass nur fehlerfreie Software zum Kunden kommt.

Sprint Review
Im Sprint Review Meeting wird dem Kunden der aktuelle Stand seines Produktes gezeigt. Durch diese kontinuierliche Integration des Kunden kann sichergestellt werden, dass das entwickelt wird, was der Kunde braucht und dass es genau nach seinen Vorstellungen ist. Dieser feste Bestandteil von Scrum stellt die Kundenzufriedenheit, und so die Softwarequalität sicher.

Sprint Retrospective
Scrum hat eine iterative Rückschau integriert, um den Prozess zu überwachen und ggfs. zu verbessern. Zusätzlich kümmert sich der QM darum, ob der gelebte Prozess mit den Unternehmensvorgaben konform ist. Er stellt gute Beispiele als Best Practices anderen Abteilungen zur Verfügung; andersherum gibt er dem Team aber auch Tipps, was in anderen Projekten wie besser läuft. Ferner sollte ein Qualitätsbeauftragter sich die Ergebnisse früherer Retrospectiven anschauen und sicherstellen, dass Verbesserungsvorschläge auch umgesetzt wurden.

3.7 Fazit

Dieser Beitrag hat gezeigt, welche Auswirkungen die agilen Werte und Methoden auch auf andere Abteilungen haben können und wie erste Anpassungen am Beispiel des QM aussehen. Es wurde vorgestellt, wie eine externe Qualitätsabteilung sich in die agile Methode Scrum integrieren lässt. In Kapitel 2 wurde erwähnt, dass es darüberhinaus noch viele weitere Bereiche gibt, die in Zusammenhang mit einer Entwicklungsabteilung stehen. Das weitere Vorgehen wird sich nun darauf konzentrieren, systematisch die Schnittstellen zu den anderen Abteilungen zu überdenken, um die agile SE bestmöglich ins Unternehmen zu integrieren.

Literaturverzeichnis

[Am05] Ambler, S.: Quality in an agile world. In Software Quality Professional (ASQ), VOL. 7, No. 4, 2005

[Bo06] Borgmeier, E.: Mit agilem Qualitätsmanagement zum Projekterfolg. In JavaSpektrum, 5/2006

[Co02] Cockburn, A.: Agile Software Development. Pearson Education, Inc., 2002

[Di07] Dietrich, S.: Qualität durch Agilität. Anecon, 2007

[Ec04] Eckstein, J.: Agile Softwareentwicklung im Großen. dpunkt.Verlag, 2004

[Man01] Manifesto for Agile Software Development. URL:http://www.agilemanifesto.org (zugegriffen: 16.01.2008), 2001

[PP03] Poppendieck, M.; Poppendieck, T.: Lean Software Development. Addison-Wesley, 2003.

[SB01] Schwaber, K.; Beedle, M.: Agile Software Development with SCRUM. Prentice Hall, 2001

[SEI] Software Engineering Institute, Carnegie Mellon University. CMMI – PPQA. URL: http://www.sei.cmu.edu/cmmi/models/ACQcompares/PPQA.pdf (zugegriffen 16.01.2008)

Das Link-System: Von Weblinks zu universellen, sensitiven Hyperlinks

Dirk Fischer

Rohde & Schwarz GmbH & Co. KG,
Bereich Standardisierung und Entwicklungssupport
Mühldorfstraße 15, 81671 München
Dirk.Fischer@rohde-schwarz.com

Abstract: Der Artikel beschreibt die Lösungsidee eines universellen Hyperlinkings zwischen beliebigen Applikationen. Dies ermöglicht eine internet-artige Navigation zwischen Applikations-Inhalten. Damit wird eine für den Benutzer intuitive, transparente und vor allem effektive Integration dieser Applikationen geschaffen, die nur wenig invasiv ist und ein hohes Maß an Wertschöpfungspotenzialen schafft, wie in einer ersten Evaluierung in der Firma Rohde & Schwarz GmbH & Co. KG aufgezeigt wurde.

Zur Universalität dieser Hyperlinks trägt das im zweiten Teil des Artikels beschriebene Konzept der sensitiven Hyperlinks bei. Diese erlauben die punktgenaue Lokalisierung von Informationen innerhalb von Texten auch in den Fällen, in denen die zugehörigen Textpassagen nicht explizit durch ihren Autor als Link-Ziele markiert wurden. Bei der Navigation wird berücksichtigt, dass die Textpassagen ggf. verändert wurden, seitdem sie sensitiv verlinkt worden waren.

1 Die Ausgangssituation

Typische Fragestellungen bei der Erarbeitung einer Information sind: „Wie kommt dieses Ergebnis zustande, wie ist es zu erklären?“, „Wurden alle Hintergrund-Informationen berücksichtigt?“ oder „Könnte eine andere Information auf gleiche Weise begründet und hergeleitet werden?“. Ebenso stellt sich oft die Frage, was mit einer Referenz gemeint sein könnte. Beispiele dazu zeigt das Literaturverzeichnis dieses Artikels.

Heutzutage werden die meisten Informationen in IT Systemen erarbeitet und veröffentlicht. Trotzdem ist es Autoren oft nicht möglich, ihren Informationen Hintergründe in einer Form beizufügen, die es Lesern mit den für sie geringst möglichen Kosten erlaubt, auf diese Hintergründe zuzugreifen. Selbst wenn ein IT System für die Bereitstellung von Hintergrund-Informationen das Konstrukt von Hyperlinks erlaubt, werden diese oft wegen ihrer für Menschen unbrauchbaren Form so erheblich zu Störinformationen in der eigentlichen Information, dass ein Autor auf deren Einfügen verzichtet. Dies widerspricht seiner oft jahrelang geübten Praxis des Einfügens von Literaturverweisen, wie in jedem wissenschaftlichen Artikel, sogar in diesem, verlangt.

1.1 Informationsbeschaffungskosten

Die Kosten für die Beschaffung einer Information können wie folgt unterteilt werden:

- **Identifikationskosten**: Kosten für die Feststellung, dass Informationen benötigt werden und welche dies sind.

  Beispiel: Wäre der Artikel X bekannt gewesen, dann wäre der lange verfolgte Software-Design-Ansatz niemals in Erwägung gezogen worden.
- **Zuordnungskosten**: Kosten für die Identifikation möglicher Informationsquellen.

  ... Auch wenn bekannt ist, dass zum Thema relevante Beiträge existieren, muss festgestellt werden, welche (hier: Artikel X) und wo sie verfügbar gemacht wurden.
- **Lokalisierungskosten**: Kosten für die Lokalisierung möglicher Informationsquellen.

  ... Mit dem Wissen, dass Artikel X in der Zeitschrift Y von 1972 veröffentlicht wurde, muss festgestellt werden, wo auf die Zeitschrift Y zugegriffen werden kann.
- **Verfügungskosten**: Kosten für die Beschaffung von und den Zugang zu Informationsquellen.

  ... Dieser Ort muss ggf. aufgesucht werden, es müssen Lizenzen bezahlt und/oder auf eine Ausleihe gewartet werden.
- **Selektionskosten**: Kosten für die Lokalisierung der benötigten Informationen in einer Informationsquelle.

  ... Der Artikel X ist sehr lang. Explizit zum Thema wird etwas nur kurz in der Mitte erwähnt (was von außen nicht zu erkennen war). Eine verständlichere Detaillierung wurde in einem anderen, bisher nicht in Betracht gezogenen Artikel vorgenommen.
- **Interpretationskosten**: Kosten für die Verarbeitung der erhaltenen Information.

  ... Die benötigten Informationen sind zusammengetragen. Sie müssen verstanden und korrekt in den Kontext der eigenen Information übertragen werden.

1.2 Desinformationskosten

Die Kosten für die Nichtbeschaffung von Informationen entstehen im Wesentlichen durch die Unterschiede von vermuteten und tatsächlichen Informationen. Die Folgekosten solcher Fehlinformationen können beliebige Ausmaße annehmen. So fokussiert z.B. die RTCA DO-178B als Richtlinie zur Zulassung von Softwareteilen in Luftfahrzeugen auf eine lückenlose Rückverfolgbarkeit von Entscheidungen (Traceability), insbesondere dann, wenn Menschenleben durch Fehler gefährdet werden.

2 Der Lösungsansatz

Das **Link-System** soll einen **universellen Hyperlink-Dialog** zur Verfügung stellen, der das Navigieren zwischen beliebigen Informationsquellen erlaubt. In diesem (nicht modalen) Dialog soll ein Autor beliebige Ziele in beliebigen Datensätzen von Applikationen auswählen können. Diese sollen in seine Information in verständlicher und lesbarer Form eingetragen werden, sodass sie einem Leser dieser Information einen Hinweis darauf geben, welche Hintergrundinformation er erwarten sollte.

Wird ein solcher Hyperlink ausgewählt, soll er punktgenau an die vom Autor spezifizierte Stelle innerhalb der Information navigieren. Selbst dann noch möglichst präzise, wenn diese Information zwischenzeitlich verändert wurde oder die Stelle nicht gesondert markiert ist (z.B. als Marke in Microsoft Word).

2.1 Einschätzung der Lösung in einem Rohde & Schwarz Entwickler-Workshop

Als die wesentlichen Risiken eines solchen Systems wurden benannt: 1. der Projekt-Umfang; 2. der Verlust der Akzeptanz durch häufige ‚Broken Links' (z.B. wegen sich ändernder referenzierter Inhalte); 3. das Finden des richtigen Maßes für eine optimale Verlinkungsdichte oder –Häufigkeit in Projekten. Letzteres kann nach Einschätzung der Mitglieder nur durch Erfahrung mit dem System erlernt werden.

Wesentliche Nutzen waren die Einschätzung der Verringerung der Entwicklungsaufwendungen um ca. 7%, sowie die Verminderung des Arbeitsstresses durch bessere Entscheidungsgrundlagen. Es sollte eine funktionale Überfrachtung vermieden werden, die Links sollten lesbar sein, um nicht zu Störinformationen zu werden, und das System sollte einfach, ohne Handlungszwänge zu bedienen sein. Zuletzt sollte dieses System nicht als Wissensmanagement-System interpretiert und in dieser Hinsicht ausgearbeitet werden. Dies meint, eine Datenbank als Container aller Informationen, wie z.B. in einem Wiki sollte vermieden werden, um zu starke Eingriffe in die Applikationen zu vermeiden und im Arbeitskontext durch die Applikationen selber alle Informationen erreichbar und modifizierbar zu haben.

Aus diesem Workshop ergab sich für Rohde & Schwarz, dass mit der Integration von (nur) 17 Applikationen mehr als 65% aller Entwicklungs-Aufgaben durch dieses System unterstützt werden.

2.2 Architektur der Lösung in Microsoft Windows Betriebssystemen

Die Applikation, in deren Daten der Link eingetragen werden soll, öffnet einen **Hyperlink-Dialog**, der optisch dem von Microsoft Office ähnlich ist. Im Unterschied zu diesem Link-Dialog können sich hier Adapter als Datenquellen anmelden und über den Button „Textmarken..." die Link-Ziele ihrer Informationsquellen zum Browsen anbieten. Dies geschieht über einen **Adapter** für jede Applikation, der deren Daten standardisiert im System repräsentiert. Der Adapter wandelt oder erzeugt, falls nötig, einen für den Leser verständlichen Link (~> Abbildung 1).

Wird der Link ausgelöst, so wird ein **Moniker** [MSDN08] des Link-Systems aufgerufen. Dieser übernimmt entweder mit Hilfe des spezialisierten Adpaters die Navigation der Applikation (falls diese keine Hyperlinks unterstützt), oder bestimmt die von ihr benötigte URL und leitet diese an das Standard Link Management im Betriebsystem weiter. Unterstützt die Applikation keine Navigation innerhalb der Informationen, kann der Moniker nach dem Start der Applikation durch das Standard Link Management zusätzlich mit Hilfe des Adpaters in der Applikation punktgenau navigieren.

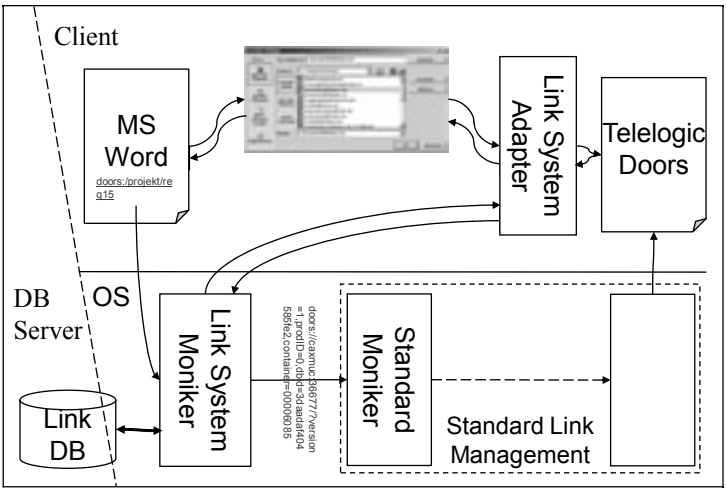


Abbildung 1: Architektur des Link-System
(Bsp: Microsoft Word / Telelogic Doors und Microsoft Windows)

Die **Link-Datenbank** wird im Hintergrund bei Speichervorgängen mit den aktuellen, in einer (versionierten) Informationsquelle verfügbaren Zielen aktualisiert, sowie mit den von dieser Quelle referenzierten Zielen. Sie erlaubt es dem Moniker, bei einer ungültig gewordenen URL die jeweils neue gültige zu ermitteln und über ein Service-Tool diese ggf. in den Informationen zu aktualisieren. Hauptzweck der Link-Datenbank ist somit die Vermeidung von Broken Links und die Möglichkeit zu einer performanten Forward- und Backward Traceability-Analyse.

Es kann eine zentrale Link-Datenbank für alle Client-Systeme verwendet werden, wenn die verknüpften Informationen ebenfalls z.B. durch ein Konfigurations-Management-Werkzeug oder gemeinsame Datenserver zentralisiert sind. Soll das Link-System auch bei rein lokalen Informationsquellen verwendet werden, muss die zentralisierte Link-Datenbank mit einer auf dem jeweiligen Client-System für lokale Quellen verwendeten Link-Datenbank im Verbund arbeiten.

Mit dieser Integration bleiben alle Funktionalitäten der Applikationen ohne Modifikation erhalten. Neuerungen in den Applikationen können genutzt und in das Link-System integriert werden, da es eine reine **Ergänzung** des Link Managements im Betriebsystem ist. Die Integration des Monikers erfordert allerdings einen erheblichen Eingriff in das Betriebssystem (Hooking). Ändern sich Applikationsschnittstellen müssen im schlimmsten Fall die Integration des Hyperlink-Dialogs und die Implementierung des spezialisierten Adapters angepasst werden.

Integriert werden können all jene Applikationen, die bereits die Navigation mit klassischen Hyperlinks in ihren Daten unterstützen, da sie in diesem Fall den Aufruf in das Standard Link Management des Betriebssystems implementiert haben. Der Aufwand zur Integration einer Applikation hängt dann wesentlich von der Qualität und Reife ihrer Programmierschnittstellen und der Anpassbarkeit ihrer Menüleisten ab.

3 Sensitives Hyperlinking

Sollte die zu referenzierende Informationsquelle keine Marken oder Strukturen in ihren Informationen unterstützen (z. B. Microsoft Notepad), oder sollte die zu referenzierende Stelle in der Information nicht durch deren Autor geeignet markiert worden sein, soll im Link-System das Konzept der **sensitiven Hyperlinks** [Fi06] eingesetzt werden, um auch in diesen Fällen navigieren zu können. Derartige Links sollen (möglichst) robust gegen Änderungen in den von ihnen referenzierten Informationen sein (vgl. 2.1).

Sensitive Hyperlinks werden innerhalb einer allen Adaptern gemeinsamen Adapter-Basis implementiert, die auch eine einheitliche Schnittstelle ins System sicherstellt (vgl. 2.2).

3.1 Die Erstellungs- und Wirkungsweise sensitiver Hyperlinks

Möchte ein Autor sensitiv auf eine Information in einem Dokument verweisen, soll er in diesem nur die gewünschte Passage markieren. Mit Hilfe unterlegter Wörterbücher und Thesauren wird dann von dem gesamten Dokument ein Profil erstellt, aus dem die Vorkommenshäufigkeiten aller Begriffe der markierten Passage einschließlich ihrer Flexionen und ggf. auch Synonyme passagenweise hervorgehen. Damit kann ermittelt werden, welche Begriffe ausschließlich in der vom Autor markierten Passage vorkommen. Diese werden in einem ersten Schritt ignoriert. Stattdessen wird eine (möglichst kleine) Menge von Begriffen gebildet, die ansonsten im Dokument nur selten vorkommen, die jedoch durch den Algorithmus zum Auffinden der Passage im Dokument (auf Basis des zuvor extrahierten Profils) zuverlässig gefunden werden. Begriffe zum Auffinden der Ränder der Passage müssen entsprechend dem obigen Verfahren gesondert der Begriffsmenge zugewiesen werden. Zuletzt werden die nur in der Markierung aufgetretenen Begriffe der Menge zugeordnet. Diese Begriffe einschließlich einer Information zum Umfang der Passage bilden die Indikator-Information im sensitiven Hyperlink, die zum erneuten Auffinden der Passage in diesem Dokument verwandt werden kann[1].

Der Algorithmus zum Auffinden der Passage bildet mit den im Indikator eingetragenen Begriffen ebenfalls ein Profil der Vorkommen dieser Begriffe und ihrer Flexionen und ggf. Synonyme im gesamten Dokument. Dann vergleicht er entsprechend der Umfang-Information des Indikators und einer einfachen Existenz-Vergleichs-Metrik die Anzahlen in den Passagen des Profils mit denen der Mengen. Die optimalen Fundstellen markieren die wahrscheinlichste vom Autor ursprünglich markierte Passage.

Für dessen Akzeptanz muss dieses Konzept performant umgesetzt werden. Zudem sollte es bzgl. Rechtschreibkorrekturen und Änderungen in Silbentrennungen etc. stabil sein. Daher bietet es sich an, den fehlertoleranten, bitparallelen Such-Algorithmus von Myers [My99] entsprechend der hier benötigten Anforderungen wie folgt zu ergänzen.

[1] Die Indikator-Information eines sensitiven Hyperlinks kann als Skizze der bei seiner Navigation zu identifizierenden Textpassage interpretiert werden. Diesen Gedanken weiterführend, lässt sich dieses Konzept auch auf Bild-Informationen erweitern.

3.2 Der bitparallele Such-Algorithmus von Myers [My99] mit Filter-Konzept

Das **k-Differenz-Problem** [St02] bezeichnet die Suche eines (Pattern-)Suchstrings $s=s_{[1]}...s_{[m]}$ in einem Textstring $t=t_{[1]}...t_{[n]}$ mit $m\equiv|s|\leq n\equiv|t|$, die einen **Levenshtein Abstand** [Le65] (entspricht der minimalen Anzahl von Einfügen-, Löschen- und Ersetzen-Operationen zur Überführung eines Strings x in einen String y) von höchstens k (Fehlstellen mit den Teilstrings aus t) haben. Die **Distanz-Matrix** $D := D_{i,j}$ mit:

$$D_{i,0} = i \qquad 0\leq i\leq|s|=m \qquad \textbf{[1]}$$

$$D_{0,j} = 0 \qquad 1\leq j\leq|t|=n \qquad \textbf{[2]}$$

$$D_{i,j} = \min\begin{cases} D_{i-1,j} + 1 \\ D_{i,j-1} + 1 \\ D_{i-1,j-1} + Eq(i,j) \end{cases} \quad 1\leq i\leq|s|=m,\ 1\leq j\leq|t|=n \quad \textbf{[3]}$$

$$\text{mit } Eq(i,j) = 0 \text{ falls } s_{[i]}=t_{[j]}, \text{ sonst } 1$$

löst dieses Problem für jedes $1\leq j\leq|t|$, für das $F(s)(t_i)\equiv D_{|s|,j}\leq k$ gilt [WMM92]. Dies bedeutet, dass der mit dem Zeichen t[j] endende Teilstring $t_{[j-|s|+1]}...t_{[j]}$ von t, mit höchstens k Fehlern, mit s übereinstimmt.

Myers [My99] konnte mit seinem Algorithmus die Rekursionsformel der Distanz-Matrix auflösen und aufgrund einiger Eigenschaften der Matrix deren Berechnung ausschließlich als inkrementelle Veränderung herleiten. In seiner Herleitung wird der Übergang von einer Spalte j zu der Spalte j+1 der Matrix für jedes i durch 2 Bits (+1, 0, -1) repräsentiert. Ihm ist es gelungen, alle einzelnen Bitoperationen als (logische) Wortoperationen eines Prozessors mit Bitbreite w zu formulieren (**Meyers-Funktion**). Damit erreichte er erstmals eine Laufzeit der Ordnung $O(|\Sigma| + |s| + |t|*\lceil|s|/w\rceil)$ zur Lösung dieser Aufgabe, wobei $|\Sigma|$ die Anzahl der Zeichen des Alphabets Σ beschreibt. Hyyrö [HN02] gelang es, die Anzahl der Wortoperationen zu reduzieren und mit einem dem folgenden Ansatz ähnelnden Verfahren, diese Laufzeit für lange Strings mit $|s|>w$ zu verbessern.

Die für die sensitiven Hyperlinks zweckmäßige Veränderung dieses Algorithmus verfolgt ein Filter-Konzept und basiert auf der folgenden Eigenschaft der $D_{i,j}$ der Distanz-Matrix D [Uk85]: $D_{i,j}\in\{ D_{i-1,j-1}, D_{i-1,j-1}+1\}$ **[4]**. Wird der zu suchende String s als aus den Strings s_1 und s_2 zusammengesetzt betrachtet ($s=s_1+s_2$), gilt wegen [*4*] für den Fehler an einer Stelle i: $F(s_1+s_2)(t_{i+|s_2|}) \geq F(s_1)(t_i) \ \forall\ 1\leq i\leq|t|-|s_2|$ **[5]**. Außerdem folgt aus [*1*]: $F(s_1+s_2)(t_i) \geq |s| - |s_2| \ \forall\ 1\leq i\leq|s_2|$ **[6]**.

Die Bedingung [*5*] rechtfertigt, dass der zweite Wortteil s_2 nur dann gesucht wird, wenn $F(s_1)(t_i) \leq k$ **[7]** gilt. Ebenfalls wegen [*5*] muss dies dann notwendig die nächsten $|s_2|$ Zeichen geschehen. Im Falle von [*7*] bedeutet dies, dass innerhalb der Ablaufschleife über die Zeichen von t und der Berechnung der Meyers-Funktion für s_1 ein Zonen-Zähler z auf $|s_2|$ gesetzt werden muss, der für $z>0$ zeichenweise dekrementiert wird und die Berechnungen der Meyers-Funktion für s_2 ebenfalls durchführt. Immer, wenn z von Null auf einen Wert größer Null wechselt, müssen die Parameter der Meyers-Funktion für s_2 neu initialisiert werden. Die generelle Algorithmus-Initialisierung der Ordnung $O(|\Sigma| + |s|)$ muss jedoch nur einmalig erfolgen. Wegen [*6*] ist es notwendig, den Algorithmus mit einem Zählerwert von 1 zu starten. Diese (iterativ für weitere Unterteilungen von |s|) fortsetzbare Modifikation, hat für w=3 und k=1 das in Abbildung 2 dargestellte

Ablaufverhalten und eine Ordnung kleiner gleich $O(|\Sigma| + |s| + |t|*\lceil |s|/w \rceil)$. Für Suchstrings mit $|s| > w$ kann dies den Algorithmus von Myers beschleunigen.

Neben dieser Veränderung kann der Algorithmus außerdem für die Suche mehrerer Strings $s^1,...,s^p$ modifiziert werden. Dazu kann der Suchstring s des Algorithmus als aus den $s^1...s^p$ zusammengesetzt aufgefasst werden. Innerhalb der Meyers-Funktion muss nur ein Übersprechen der Wortoperationen über die einzelnen Begrenzungen hinweg mit Hilfe einer einfachen Bitmaske verhindert werden, sodass der Algorithmus gleichzeitig diese Strings suchen kann. Er muss dann allerdings alle $F(s^q)(t_i)= D_{\sigma,j}$ mit $\sigma = \Sigma_{r=1...q}|s^r|$ und $1 \leq q \leq p$ bezüglich der Bedingung [7] überprüfen.

Im Falle sensitiver Hyperlinks werden mehrere Begriffe einschließlich ihrer Flexionen gesucht. Flexionen haben häufig die Eigenschaft, ein mit dem ursprünglichen Wort gemeinsames Präfix zu haben („*Tante*“, „*Tanten*“, „*Tantchen*“). Werden nun diese Präfixe der unterschiedlichen Begriffe gemeinsam in den Algorithmus von Myers eingespeist und wird entsprechend einer ‚Kreuzung' der obigen Konzepte nur ‚wenn nötig' nach den unterschiedlichen Suffixen gesucht, kann das Erstellen und das Auffinden des Indikators auch in großen Dokumenten ausreichend schnell geschehen [Fi06].

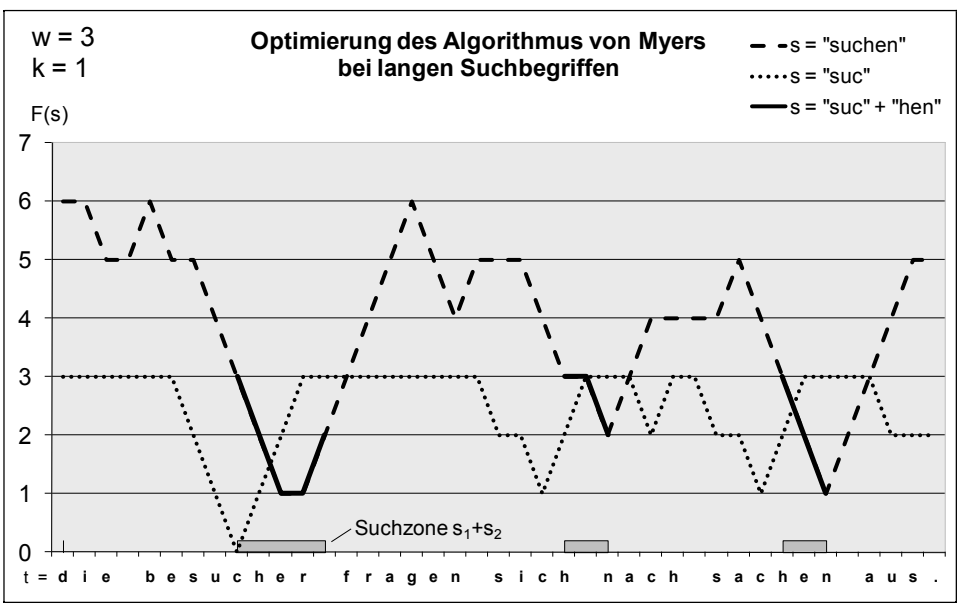


Abbildung 2: Optimierung des Algorithmus von Myers mit dem Filter-Konzept

4 Zusammenfassung

Im Gegensatz zu bestehenden Systemen vermeidet das hier vorgestellte Konzept die Zusammenführung und damit Kopie von Daten in eine große gemeinsame Datenbank.

Desweiteren verzichtet dieses Konzept auf die automatische Extraktion von Links, wie es in vielen Information-Retrieval-Applikationen praktiziert wird. Damit soll bewirkt werden, dass Autoren bewusst Hintergründe zu ihren Informationen benennen und so gezielt Störinformationen auf ein Minimum begrenzen können. Dies ermöglicht die in Kapitel 2.1 erwähnte Notwendigkeit zur bewussten Steuerung der Link-Dichte.

Zuletzt zeichnet sich das hier vorgestellte System dadurch aus, dass es den Anwender im Arbeitskontext einer Aufgabe innerhalb der dazu aktuell verwendeten Applikation unterstützt. Es stellt die Verknüpfung nicht in einem nachgeschalteten, im schlimmsten Fall zeitlich verzögerten und von den Arbeitsabläufen und den dazu verwendeten Applikationen entkoppelten System bereit. Die Fokussierung auf die Unterstützung während der Durchführung einer Aufgabe wird insbesondere durch die sensitiven Hyperlinks unterstrichen. Mit diesen kann auch dann innerhalb eines Dokuments referenziert werden, wenn ein Dokument nicht mit (klassischen) Markierungen als Hyperlinkziele versehen wurde.

Das hier vorgestellte System ist eine querschnittliche und schlanke Integration der Werkzeuge, die eingesetzt werden, ohne selber als Werkzeug in Erscheinung zu treten.

Literaturverzeichnis

[Fi06] Fischer, Dirk: „Flexible Informationssuche", Diplomarbeit am Lehrstuhl für Betriebswirtschaftslehre, insbesondere Operations Research an der Fernuniversität in Hagen (2006), unveröffentlicht.

[HN02] Hyyrö, Heiki, Navarro, Gonzalo: „Faster Bit-parallel Approximate String Matching" in: Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching, July 03-05 (2002) S. 203-224, Web-Download: http://portal.acm.org/citation.cfm?id=736383&dl=GUIDE&coll=GUIDE&CFID=10952486&CFTOKEN=81722893

[Le65] Levenshtein, Vladimir I.: „Binary codes capable of correcting deletions, insertions, and reversals." in: Doklady Akademii Nauk SSSR, 163(4) S. 845-848, 1965 (Russisch). Englische Übersetzung in: Soviet Physics Doklady, 10(8) S. 707-710, 1966.

[MSDN08] Microsoft Developer Network Library, 2008, Web-Download: http://msdn2.microsoft.com/en-us/library/ms691261.aspx

[My99] Myers, Gene: „A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming", in: Journal of the ACM, Volume 46 [3] (1999) 395-415, Web-Download: http://citeseer.ist.psu.edu/cache/papers/cs/3956/http:zSzzSzwww.cs.arizona.eduzSzpeoplezSzgenezSzPAPERSzSzbit.mat.pdf/myers99fast.pdf.

[St02] Stamme, Katrin: „Einsatz von bitparallelen Algorithmen und Filtern zur approximativen Zeichenkettensuche", Diplomarbeit am Institut für Informatik der Universität Hannover (2002).

[Uk85] Ukonen, E.: „Algorithms for approximate string matching", in Information and Control 64 (1985) 100-118.

[WMM92] Wu, Sun, Manber, Udi, Myers, Gene: „A Sub-quadratic Algorithm for Approximate Limited Expression Matching", in: Algorithmica 15 [1] (1992) 50-67,Web-Download: http://citeseer.ist.psu.edu/cache/papers/cs/3178/ftp:zSzzSzftp.cs.arizona.eduzSzreportszSz1992zSzTR92-36.pdf/wu92subquadratic.pdf.

TIME - Tracking Intra- and Inter-Model Evolution

Maximilian Kögel
Technische Universität München
Department of Computer Science, Chair for Applied Software Engineering
Boltzmannstrasse 3, D-85748 Garching, Germany
koegel@in.tum.de

Abstract: Modern software development approaches, especially the model-driven approaches, heavily rely on the use of models during the whole development process. With the increasingly integrated tools these models converge into one unified model for various artifacts from various development activities. These unified models evolve over extended periods of time, which creates an emerging demand for versioning, comparing and merging them. Current approaches for Software Configuration Management (SCM) systems are note adequate for the special requirements of these integrated models. Most of them have been designed to manage changes in textual artifacts such as source code in a file system. Consequently they operate on the abstraction of a file system and represent change in a line-oriented way. Some of them have been developed as a solution for a single tool and can therefore not provide a seamless solution for an integrated model.

We propose TIME (Tracking Inter- and Intra-Model Evolution), a novel software configuration management approach for software engineering artifacts that is able to manage change in graph-structured artifacts and supports traceability. Our approach is based on operation-based deltas, change packages and product versioning. To demonstrate feasibility we implemented a TIME prototype in Sysiphus, a tool for collaborating over software engineering artifacts.

1 Motivation

Change pervades the entire software life cycle. Requirements change when developers improve their understanding of the application domain, the system design changes with new technologies and design goals, the detailed design changes with the identification of new solution objects and the implementation changes as faults are discovered and repaired. These changes can affect every work product, from system models to source code and documentation. It is widely recognized that software configuration management (SCM) is crucial for maintaining consistency among while minimizing the risk and cost of changes to *all* of these artifacts [Vie05]. We claim that Software Engineering models are essentially graphs. For many artefacts such as UML class or use case diagrams this seems to be very obvious, however even other artifacts such as release plans or even design decisions are graph-based. Wolf proposed such a model with graph-based meta model called Rational-based Unified Software Engineering model (RUSE) in [Wol07]. In an integrated (unified) model we distinguish two different types of links. *Intra-model links* connect model elements *within one model*, such as a use case model. In a use case model a link from a

use case to a participating actor is an intra-model link. *Inter-model links* connect model elements of *different models*. A link from a use case in the use case model to an open issue in the issue model is an inter-model link.

Integrating different tools, also implies integrating their models. Intra-model links are already part of the models, while inter-model links are added during integration. Only by adding inter-model links models from different tools can be set into relation and additional value is generated compared with having separate models. Inter-model links may be added for numerous reasons but a major motivation for adding them is traceability. Only by adding inter-model links elements from the previously isolated models can be traced to model elements from different models. As intra-model links are defined in the context of the respective tool they can be supported by the tools' SCM capabilities if there are any. However this will create a media break when viewing models from different tools. In contrast the inter-model links are unknown to the tools, subject to integration, and are not supported by their SCM capabilities. So in summary for supporting the management of change in an integrated environment we need an integrated approach for SCM that is not limited to a single tool and intra-model links. Support for managing change in an integrated model such as RUSE essentially requires support for managing change in models with graph structure. The traditional SCM systems are geared towards supporting textual artifacts such as source code. Therefore changes are managed on a line-oriented level. In contrast, many software engineering artifacts are not managed on a line-oriented level and therefore a line-oriented change management is not adequate. For example adding an association between two classes in a UML class diagram is not line-oriented nor can the change be managed in a line-oriented way. A single structural change in the diagram will be managed as multiple line changes by traditional SCM systems. Nguyen et al. describe this problem as an *impedance mismatch* between the flat textual data models of traditional SCM systems and graph-structured software engineering models. [NMBT05]. We conclude that the traditional SCM systems are inadequate for managing artifacts with graph structure and for supporting traceability. This paper addresses the problem by proposing a novel approach for a software configuration management system for integrated models such as RUSE featuring support for intra-model and inter-model links. Our approach builds on operation-based deltas, change packages and product versioning.

2 TIME

The discussion and evaluation of various SCM techniques throughout this section largely follows Conradi and Westfechtel's *uniform version model* presented in [CW97, CW98]. This framework provides a common terminology and classification the available alternative approaches for SCM system design can be uniformly expressed and compared with. We extended the categorization of the original framework by adding the aspect of delta representation since it is important for change management of graph-structured artifacts. In the first three subsections we discuss three dimensions from the uniform version model. We present the different design alternatives and explain our choice. Finally we present the resulting version model based on the decisions from the previous subsections.

2.1 Delta Representation

Delta Representation deals with the representation of differences between two models at different points of time or at different variants. There are two basic approaches for Delta Representation: *state based deltas* or *operation based deltas*. The differences between the two approaches can be very subtle in many SCM systems but are highly relevant. In the *state based approach* only the state representations of different versions are stored, possibly using compression or sharing of common parts. Deltas are reconstructed using a differencing algorithm that compares the different state representations. In contrast in the *operation based approach*, changes are described by using the original sequence of editor operations that caused the changes.

With state based deltas, the semantic context of the original operations that caused the change has to be recalculated with the deltas. This recalculation is expensive, time complexity is dependent on the project size and in some cases it does not work at all [Let05]. For example, it can be impossible if the changes of one operation are partially or completely masked by those of a later operation.

Storing the original editor operations automatically captures the original semantic context of the changes. Several other research efforts have successfully employed the operation based approach in similar environments [RW98, NMBT05, OS06]. A drawback of the operations based approach is that the operations have to be recorded while the models are manipulated. This will either result in coupling the editor tools with the SCM engine or in coupling a model layer underneath the editor with the SCM engine. However, this can be resolved by defining a standardized language to express changes in the means of operations.

For the reasons mentioned above we use an operation based delta representation in our approach.

2.2 Delta Granularity

Another important question regarding an SCM system is the granularity with which changes are described. This is called the *delta granularity*. In an unified model, such as the RUSE model, changes occur on three different semantic layers of granularity.

Logical Layer These changes are sets of logically coherent work as seen by the user, e.g. ”I updated the use cases, their analysis and the glossary according to today’s client review”.

Model Layer These changes are atomic changes as far as a specific model is concerned, e.g. ”set a new initiating actor for a use case”. They correspond to model specific operations on the model elements and are usually comprised of several changes on the meta model layer.

Meta Model Layer These are the changes as seen by the meta model layer, basically

a graph. These changes represent simple modifications in the graph structure or node or edge attributes. Users of the system are usually not aware of and do not understand this layer of change.

The SCM approach needs to be able to describe and track changes on all three layers of granularity. Change tracking can easily be achieved on the meta model layer since changes can be described here with the finest granularity of changes to single attributes of single model elements. Meta model change description has the additional benefit of being independent of the model layer. Therefore, changes should be described and tracked on the meta model layer.

Unfortunately, this alone is not sufficient since it does not capture enough context. A meta model change on its own will not be meaningful to a user of the system since he will be working on the semantic level of the model layer and generally not be aware of the mechanics of the meta model layer. Reconstructing the original model layer changes from a series of meta model changes is a difficult task [Let05]. Furthermore it is nearly impossible in an integrated model like RUSE since it would require the SCM engine to have detailed knowledge of every model on the model layer. Additionally operations on the model layer often do not have an injective mapping to the meta model, making an unambiguous reconstruction impossible even in theory.

An SCM system can automatically track and describe changes on the meta model and model layer, but not on the logical layer. Therefore, the SCM approach must provide a mechanism for manually grouping and describing logical changes. Our approach provides change packages with log messages to achieve this on the logical layer. Changes on the model and meta model layer are automatically captured.

2.3 Version Granularity

Three possible approaches for *version granularity* are described in [CW97]: Component Versioning, Total Versioning and Product Versioning

Component Versioning lacks intrinsic support for managing consistent configurations, as every configuration item is in its own version space. However the inter-model and intra-model links in the RUSE model introduce dependencies among the model elements and therefore a configuration as a set of versions of model elements needs to be managed.

Total Versioning is not a big improvement in this respect since it still requires explicit management of consistent configurations. The model elements managed by the SCM system should completely and unambiguously describe exactly one system under development. Many problems of this approach are described in [NMBT05].

Product Versioning lacks any modularity of the version space since it only has one uniform global version space for all configuration items. The main advantage of this approach is that version spaces of different configuration items are naturally related, alleviating the need to find combinations that produce valid configurations. Product versioning thus automatically provides consistent configurations without the need for explicit management

of configurations or the SCM engine having to know about the exact nature of the data model. Furthermore, product versioning is a natural match with change packages since both approaches handle changes spanning multiple configuration items as a coherent entity. The most important disadvantage of product versioning is that variants need to be global, too [CW98]. In cases where this is required, such variants can easily be provided by branching.

In summary product versioning provides the version granularity most suitable for configuration management on graph-structured artifacts.

2.4 Version Object Model

The version model resulting from the previous analysis is shown in a UML class diagram in Figure 1. The version model classes are shown with their dependencies on the data model.

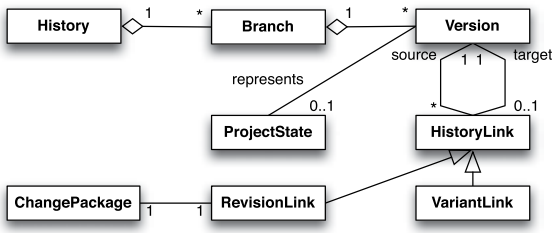


Figure 1: Version Model (UML class diagram)

The Version space is represented by a version tree graph structure consisting of branches, version nodes and edges for revision and variant relationships. The revision edges are associated with the change packages describing the changes between its two version nodes. The *History* class represents the history of a project. It provides operations for creating revisions, branches and tags and for accessing specific versions, differences and history information. The *Branch* class represents a branch of concurrent development in the version space and is composed of all versions on that branch.

Versions represent the nodes in the version graph. The state of the project at a specific version can either be represented explicitly by an instance of *ProjectState* or implicitly by its position in the version graph and the appropriate deltas.

The *HistoryLink* class and its subclasses represent the edges in the version tree graph. A revision relationship between two *Versions* is represented by the *RevisionLink* class. The changes that caused the revision are described by the associated *ChangePackage*. A variant relationship is represented by a *VariantLink*. Note that a *Version* can have at most one incoming and at most one outgoing *RevisionLink*. If it has no incoming *RevisionLink*, it is the initial version of a branch. If it has no outgoing *RevisionLink*, it is the head revision of that branch. A *Version* can have an arbitrary number of outgoing *VariantLinks*

since it can have an arbitrary number of variants. However, a version can have at most one incoming *VariantLink*, in which case it has to be the initial revision of a new branch.

2.5 Change representation

In section 2.4 we did not show how the three layers of change introduced in section 2.1 are reflected in the version model.

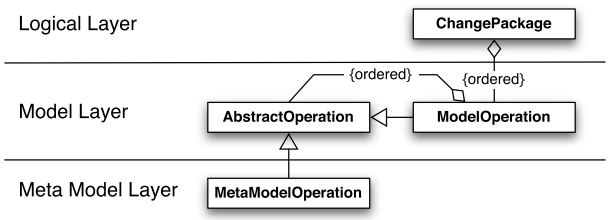


Figure 2: Three layers of change (UML class diagram)

Figure 2 shows the four classes for representing change on their appropriate layers. The *ChangePackage* class represents change on the logical layer. It is an ordered aggregation of all instances of *ModelOperation* that are generated between two commits of a workspace. As mentioned earlier grouping and describing change on the logical layer is up to the user by providing meaningful and descriptive log messages. Instances of *ModelOperation* are automatically captured by the implementations of the various model elements and their methods (e.g. setInitiatingActor(Actor actor)). *ModelOperation* is part of an ordered composite pattern with *AbstractOperation* and *MetaModelOperation*. Thereby we obtain an ordered tree structure of instances of *ModelOperation* as inner nodes and with instances of *MetaModelOperation* as leaf nodes. This is useful for structuring change for visualization. Complex changes such as refactoring are more easily apprehensible in this way. The *MetaModelOperation* class represents changes on the meta model layer. An instance of it always reflects *exactly one change* affecting *exactly one model element*.

3 Future work

To demonstrate feasibility, we implemented the proposed SCM system in *Sysiphus* [BDW06a], a tool for collaborating over software engineering artifacts following the RUSE model. We are currently evaluating the SCM in a student project with 40 students working on a problem in the area of airport logistics. Extensive data has already been collected and will hopefully help to improve our approach. We plan to extend the approach by improved conflict detection and visualization. Especially conflict detection will need further research since false positives heavily impact the productivity while merging.

References

[BDW06a] Bernd Bruegge, Allen H. Dutoit, and Timo Wolf. Sysiphus: Enabling Informal Collaboration in Global Software Development. In *Proceedings of the First International Conference on Global Software Engineering*, October 2006.

[BDW06b] Bernd Bruegge, Allen H. Dutoit, and Timo Wolf. Sysiphus: Enabling Informal Collaboration in Global Software Development. In *Proceedings of the First International Conference on Global Software Engineering*, October 2006.

[BW07] Brian Berenbach and Timo Wolf. A unified requirements model; integrating features, use cases, requirements, requirements analysis and hazard analysis. *icgse*, 0:197–203, 2007.

[CW97] Reidar Conradi and Bernhard Westfechtel. Towards a Uniform Version Model for Software Configuration Management. In *ICSE '97: Proceedings of the SCM-7 Workshop on System Configuration Management*, pages 1–17, London, UK, 1997. Springer-Verlag.

[CW98] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[Fei91] Peter H. Feiler. Configuration Management Models in Commercial Environments. Technical report, Software Engineering Institute, Carnegie Mellon University, 1991.

[Let05] Kim Letkeman. Comparing and merging UML models in IBM Rational Software Architect. Technical report, Modeling Compare Support, IBM Rational, 2005.

[LvO92] Ernst Lippe and Norbert van Oosterom. Operation-based merging. In *SDE 5: Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*, pages 78–87, New York, NY, USA, 1992. ACM Press.

[MGH05] Akhil Mehra, John Grundy, and John Hosking. A generic approach to supporting diagram differencing and merging for collaborative design. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 204–213, New York, NY, USA, 2005. ACM Press.

[NMBT04] Tien N. Nguyen, Ethan V. Munson, John T. Boyland, and Cheng Thao. Flexible Fine-grained Version Control for Software Documents. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 212–219, Washington, DC, USA, 2004. IEEE Computer Society.

[NMBT05] Tien N. Nguyen, Ethan V. Munson, John T. Boyland, and Cheng Thao. An infrastructure for development of object-oriented, multi-level configuration management services. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 215–224, New York, NY, USA, 2005. ACM Press.

[Ohs02] D. Ohst. A Fine-Grained Version and Confguration Model in Analysis and Design. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 521, Washington, DC, USA, 2002. IEEE Computer Society.

[OMW05] Hamilton Oliveira, Leonardo Murta, and Claudia Werner. Odyssey-VCS: a flexible version control system for UML model elements. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 1–16, New York, NY, USA, 2005. ACM Press.

[OS06] Takafumi ODA and Motoshi SAEKI. Meta-Modeling Based Version Control System for Software Diagrams. *IEICE Trans Inf Syst*, E89-D(4):1390–1402, 2006.

[OWK03] Dirk Ohst, Michael Welle, and Udo Kelter. Differences between versions of UML diagrams. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 227–236, New York, NY, USA, 2003. ACM Press.

[RW98] Jungkyu Rho and Chisu Wu. An Efficient Version Model of Software Diagrams. In *APSEC '98: Proceedings of the Fifth Asia Pacific Software Engineering Conference*, page 236, Washington, DC, USA, 1998. IEEE Computer Society.

[Vie05] TOC View. IEEE Standard for Software Configuration Management Plans. *IEEE Std 828-2005 (Revision of IEEE Std 828-1998)*, pages 0_1–19, 2005.

[Wol07] Timo Wolf. *Rationale-based Unified Software Engineering Model*. Dissertation, Technische Universität München, July 2007.

Change Propagation with the Change Notification Bus

Tobias Rodenbach[1] and Lena Wiese[2]

[1]Rosenhagen 12
31134 Hildesheim
tobias.rodenbach@udo.edu

[2]Fakultät für Informatik
Technische Universität Dortmund
44221 Dortmund
lena.wiese@udo.edu

Abstract: We present requirements and a basic architecture for change management across application boundaries. The Change Notification Bus addresses heterogeneity of applications and working environments as well as distribution of team members. Change Notifications are handled in a decentralised and event-based way.

1 Introduction

One key advantage of agile software development is its ability to flexibly react to changing requirements or conditions; this feature specifically allows for incremental development of prototypes with an increased range of functionalities. One crucial aspect are the interdependencies between documents of the different development stages – as for example the product specification depends on the requirements documentation and later on product tests depend on the specification. Ideally, the software product and all documents should be in a consistent state after processing a requirement change.
As a necessary condition for document and product consistency, software development requires "traceability": the ability to identify all artifacts that might be affected by a change starting from the changed artifacts. However, maintenance of traceability is a difficult task during the whole product life cycle which is costly and error-prone if done by hand. This challenge is even more evident as soon as distributed development teams are concerned as changes and follow-up steps cannot be communicated directly. Although management of traceability changes is a major task in software development, there is no off-the-shelf product available that offers comprehensive support and realising traceability involves a lot of manual work of developers.[1]
Although it may be desirable to provide an integrated solution which covers all needs of the development process, this is usually impeded by practical prerequisites; we propose to employ application coupling instead of a fully integrated environment.

[1]This is particularly the case for embedded software; see Section 2 for details.

2 Application Integration versus Application Coupling

In order to automatically support traceability and (at least approximately) achieve a consistent documentation state, all employed development and management applications have to be combined. As is customary in the literature (see [CHKT05, Kel02]), we consider two forms of application combination in this article:

- **Application Integration:** Original applications are altered (or are replaced) to form a single new application. As the data exchange in the new application is direct, external interfaces of the original applications can be removed during this process.
- **Application Coupling:** Existing external interfaces of original applications can be used to exchange data between them. Due to proprietary interfaces this typically involves creation of some glue software that transforms the data from one interface format to another. In this scenario the applications remain self-contained but still form a continuous environment.

When talking about application integration, do the stakeholders involved in the development process (as e.g. customers, managers, developers, testers and administrators) benefit from having an integrated all-purpose software development and business process application; that is a tool that supports and facilitates the whole software development process covering both business and engineering needs? An integrated development environment offers a unified look and feel for all tasks during the development process for as well manager as developers and testers and in the basic setting might reduce installation and maintenance effort for administrators.
However such an environment can only either be a predefined one which does not leave much options for customisation or a very comprehensive one that offers more than is needed which increases configuration effort and potential confusion. An optimal integrated environment can be custom-made for every field of application but in most cases the accompanied development cost and effort will bar this option.
Especially in embedded software development, some tools are only available as stand alone versions or in specialised environments. Frequently legacy systems need to be maintained using old-fashioned tools. These environments usually do not offer adequate possibilities to integrate other parts of the tool chain as it is known from more recent products. They are on the other hand hard to integrate into a larger environment, e.g. because they are lacing remote control or command line interfaces. Even very comprehensive modern integrated environments do not cover all aspects of the development cycle, in particular competitive project management components are rarely found.

There are several reasons why existing applications should be kept in use. First, there are commercial aspects to consider as tools have already been bought or licensed. Second, the users are already acquainted with the existing tools and switching to a new tool would mean starting a new learning curve thus resulting in additional cost due to lower efficiency. In some areas the choice of the respective tool can be even left to the user as long as the tool's output complies with the expected result. Third, working with several independent applications also increases flexibility. When new constraints demand switching to a new

tool for a specific task, the other tools can be kept in use unchanged. Lastly, companies can also choose products by proficiency and cost instead of being forced to use what comes packaged with an integrated environment.

3 An Application Coupling Framework for Change Notifications

This chapter describes the basic architecture of a Change Notification system coupling applications in a distributed environment. The following terms will be used to refer to the different aspects of change management:

Configuration Item	Any item that goes into configuration management. Most likely to be a file, but can also be a folder or all files belonging to a MS Project plan.
Configuration Atom	The smallest coherent part of a Configuration Item that can be affected by a change (subsection describing one function, a class diagram, one requirement, one test case).
Traceability Link	A directed dependency between Configuration Atoms signifying that a change of an Configuration Atom requires the linked Configuration Atoms to be checked for needed changes.
Change Notification	Notification that a Configuration Atom might be affected by the change of another Configuration Atom.

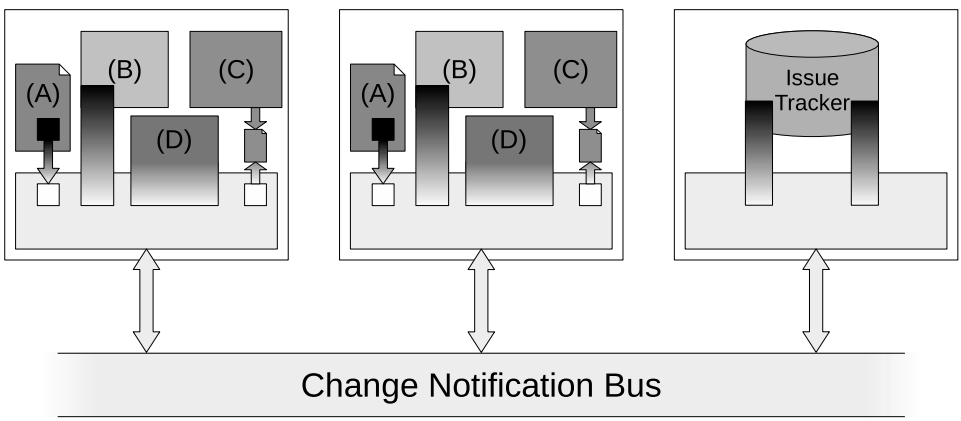


Figure 1: The Change Notification Bus

Figure 1 depicts the proposed architecture. It is divided into a (virtual) “Change Notification Bus” (CNB) and a local framework instance running on each workstation taking part in the development process, the “CNB environment”. Those two elements will be described in the following sections.

3.1 The Change Notification Bus

The CNB is not a physically existing data bus, although it is essential for the proposed architecture that all participating workstations are connected via a network. It is rather a model for the fact that all Change Notifications that are posted reach all connected CNB environment instances immediately. A message bus is a well known design pattern for application coupling, e.g. described in [HW03] as the "Message Bus" pattern.

3.2 The CNB Environment

Each CNB environment instance provides the applications with the means to

- post Change Notifications
- subscribe for Change Notifications
- mark a Change Notification as processed
- create/delete a Traceability Link [2]

For each type of document and/or application, a plugin is hooked to the CNB environment. The different types of coupling are depicted in Figure 1 and explained in the following:

(A) Document level coupling
Some applications allow for functionality augmentation by adding macros into their documents. Those documents can be coupled to the respective plugin in the CNB environment by inserting a macro that will detect changes in the document and report them to the CNB environment. Further, it will receive Change Notifications for the Configuration Atoms in the Documents and inform the user in an appropriate way, depending on the document type.

(B) Application extension
Other applications offer the possibility to extend them by plugin interfaces or application level macros. Those can be extended by bi-lateral plugins, being plugged into the application and the CNB environment at the same time. Analogously to case (A) for document level macros, Change Notifications can be posted and presented to the user in an appropriate way.

(C) Framework-internal handler for not extensible applications/document types
Applications that cannot be extended and handle only passive documents are hard to couple to the CNB environment. However, their documents can still be monitored by the respective plugins in the CNB environment. In this case, the plugin has to provide more functionality than offering the interface for Change Notification

[2] The last item is just mentioned for completeness as building and maintaining the traceability database is out of scope of this document (see Section 4).

handling; it has to be able to read the application's document format in order to detect changes. If relevant Change Notifications shall be displayed in the document, it has to write to the document file. The advantage of this kind of plugin is that document can be accessed "offline", i.e. without a running application.

(D) Native CNB application
For newly developed applications, the CNB environment's interface can be utilised directly without the need for plugin development.

If an application that is not explicitly designed for extension is available in source code it can be altered to access the CNB environment directly. This can be regarded as a blend of the types (B) and (D).

3.3 The Issue Tracker

Although it is convenient to immediately inform the user about Change Notifications referring to parts of a document that he is currently working on, the majority of Change Notifications will refer to currently inactive documents. To manage these pending Change Notifications, they are added as trackable items to the issue trackers database. The issue tracker will monitor the activity on the CNB and close the respective item as soon as a processed message for the Change Notification is posted. Using an issue tracker to track the Change Notifications allows to utilise the means of classical change management for our approach. In particular those changes can be mapped to baselines or branches, so that the respective Change Notifications will also refer to the baseline or branch. This means that the version of the linked Configuration Items can be determined.
The reflection of the Change Notifications in the issue tracker also allows for controlling the changes to the artifacts. Different policies can be applied for different notification types:

- Change Notifications triggered for *tight* links (e.g., from implementation specification to source code) may be processed without further checking; this is the main advantage of our bus architecture because events posted on the bus can be handled by registered application without delay and inside the artifacts directly (as opposed to server-based notifications of [CHCC03]).
- Change Notifications for artifacts with a more complex link structure (e.g., horizontal traceability between equally levelled requirements) need to be reviewed and approved before further action is taken; in this manner the issue tracker supports the implementation of a specified workflow in the framework.

Moreover, the issue tracker is a normal application connected to the CNB as described in Section 3.2. Whenever Configuration Atoms in tracked items like defect reports or change requests are changed, it will post a Change Notification on the CNB.

4 Conclusion and Related Work

We presented a distributed framework for Change Notification handling in a heterogeneous tool environment. The framework focuses on coupling established applications to support a high "return on investment". Notifications about changed Configuration Atoms are based on instantaneously posted events. Event-based notification for change management is also proposed by Cleland-Huang et al. in [CHCC03]. While they rely on a event server, our proposal utilises a bus architecture for broadcasting Change Notifications directly among the participating applications if no approval of the change is needed; otherwise change management workflows can be modelled in the issue tracker. Further, instead of listing the pending Change Notifications in a separate event log, we aim for a *tight* integration with the utilised development and management applications as realised by our four different coupling types; this specifically allows for handling changes *inside* an artifact (e.g., by highlighting affected sections in a text document by coupling of type (A)).
Retrieval of initial traceability information is out of the scope of this article. This topic is addressed by Poirot ([CHBC+07]) and RETRO ([HDS+07]) using information retrieval techniques. Update of traceability information based on predefined policies is covered by ArchTrace ([MvdHW06]).

Acknowledgements
We thank the anonymous reviewers for their helpful comments.

References

[CHBC+07] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. Best Practices for Automated Traceability. *IEEE Computer*, 40(6):27–35, 2007.

[CHCC03] Jane Cleland-Huang, Carl K. Chang, and Mark J. Christensen. Event-Based Traceability for Managing Evolutionary Change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[CHKT05] Stefan Conrad, Wilhelm Hasselbring, Arne Koschel, and Roland Tritsch. *Enterprise Application Integration*. Elsevier, München, 2005.

[HDS+07] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, Elizabeth Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[HW03] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2003.

[Kel02] Wolfgang Keller. *Enterprise Application Integration. Erfahrungen aus der Praxis.* Dpunkt, Heidelberg, 2002.

[MvdHW06] Leonardo G. P. Murta, André van der Hoek, and Cláudia Maria Lima Werner. ArchTrace: Policy-Based Support for Managing Evolving Architecture-to-Implementation Traceability Links. In *21st IEEE/ACM International Conference on Automated Software Engineering*, pages 135–144. IEEE Computer Society, 2006.

Modellgetriebene Softwarearchitektur

–

Evolution, Integration und Migration

(MSEIM 2008)

Inhaltsverzeichnis

Workshop Modellgetriebene Softwarearchitektur – Evolution, Integration und Migration (MSEIM 2008) 173
Michael Goedicke, Maritta Heisel, Sascha Hunold, Stefan Kühne, Matthias Riebisch, Niels Streekmann

Using collaborative business process models for the configuration of distributed workflow environments 178
Dirk Werth, Jörg Ziemann, Philipp Walter, Peter Loos

MDA und KI: Domänenspezifische Modellierung und Umsetzung wissensintensiver Prozesse 184
Steffen Kruse, Malte Zilinski, Hilke Garbe, Claus Möbus

Systematische Berücksichtigung von Abhängigkeitsbeziehungen bei Architekturentscheidungen 191
Sven Wohlfarth, Matthias Riebisch

Grafische Validierungsregeln am Beispiel von EPKs 198
Sven Feja, Daniel Fötsch, Sebastian Stein

Towards Evaluating Maintainability Within Model-Driven Environments 205
Thomas Goldschmidt, Jens Kübler

UPPAAL-Modelle als ausführbare Spezifikation in Java 212
Michael Goedicke, Moritz Balz, Michael Striewe

Modellgetriebene Integration von Altsystemen 219
Dieter Hildebrandt, Michael Gründler, Heiner Feislachen

Inkrementelle Transformation einer monolithischen Geschäftssoftware 226
Sascha Hunold , Matthias Korch , Björn Krellner, Thomas Rauber, Thomas Reichel, Gudula Rünger

Workshop Modellgetriebene Softwarearchitektur – Evolution, Integration und Migration (MSEIM 2008)

Michael Goedicke[1], Maritta Heisel[2], Sascha Hunold[3], Stefan Kühne[4], Matthias Riebisch[5], Niels Streekmann[6]

[1] Specification of Software Systems, Universität Duisburg-Essen,
michael.goedicke@s3.uni-due.de

[2] Software Engineering, Universität Duisburg-Essen,
maritta.heisel@uni-duisburg-essen.de

[3] Lehrstuhl für Angewandte Informatik II, Universität Bayreuth,
hunold@uni-bayreuth.de

[4] Betriebliche Informationssysteme, Universität Leipzig,
kuehne@informatik.uni-leipzig.de

[5] Fachgebiet Softwaresysteme / Prozessinformatik, Technische Universität Ilmenau,
matthias.riebisch@tu-ilmenau.de

[6] Betriebliches Informationsmanagement, OFFIS,
niels.streekmann@offis.de

1 Motivation

Softwarearchitekturen sind wichtige Artefakte zur Unterstützung der Software-Entwicklung und Software-Evolution. Sie dienen der grundlegenden Beschreibung von Softwaresystemen und dienen damit auch als Ausgangspunkt zur Weiterentwicklung bestehender Systeme. Diese Weiterentwicklung umfasst neben der Evolution bestehender Softwaresysteme auch die Integration mit anderen Systemen und die Migration in neue Systemumgebungen. Die modellgetriebene Softwareentwicklung, die zunehmende Verbreitung sowohl in der Wissenschaft als auch im Einsatz in der Praxis findet, stellt mit den verfolgten Zielen Automatisierung, Komplexitätsreduzierung und Qualitätssteigerung einen vielversprechenden Ansatz zur Unterstützung für wichtige Aufgaben des Designs dar. Nicht nur der Einsatz modellgetriebener Entwicklungsansätze im Kontext bestehender Legacy-Systeme wirft jedoch auch zahlreiche Fragen hinsichtlich Vorgehen, Methoden, Werkzeuge und Strategien auf.

2 Ziele des Workshops

Der Workshop „Modellgetriebene Softwarearchitektur - Evolution, Integration und Migration“ soll dazu beitragen, die Ansätze zur Einbindung bestehender Systeme in einen modellgetriebenen Softwareentwicklungsprozess zu thematisieren. Dies soll sowohl auf akademischer Ebene als auch im Austausch mit Industrievertretern geschehen, um praktische Problemstellungen und Lösungsansätze mit akademischen Methoden integrieren zu können.

Der Workshop soll in erster Linie als Diskussions- und Austauschplattform dienen. Wissenschaftler und Praktiker sind dazu aufgerufen aktuelle Probleme und Lösungsansätze zu präsentieren, um diese mit den Workshop-Teilnehmern zu diskutieren. Das Ziel dabei ist, die vorgestellten Arbeiten voranzutreiben und neue Ansatzpunkte für Problemlösungen und Kooperationen zwischen Forschungs- und Industriegruppen zu finden.

Beiträge aus den folgenden Bereichen waren erwünscht:

- Modellierungsmethoden und Architekturmodelle für Integration und Migration
- Methoden und Verfahren architekturbasierter Migration bestehender Systeme
- Werkzeuge für modellgetriebene Integration und Migration
- Methoden und Verfahren zur Ableitung und Validierung von Architekturen bestehender Systeme
- Modellanalysen im Integrations- und Migrationsprozess
- Visualisierung und Simulation von Integration
- Modellgetriebene Weiterentwicklung integrierter Informationssysteme
- Wiederverwendung durch modellgetriebene Entwicklungsansätze
- Re-Engineering monolithischer Legacy-Systeme
- Reverse-Engineering und Codeextraktion von Nutzerschnittstellen und Geschäftsprozessen
- Performance und Skalierbarkeit von Integrations- und Migrationslösungen
- Test und Verifikation von Software- und Modelltransformationen
- Validierung dynamischer Aspekte in Integrationsmodellen
- Modellgetriebene Produktlinienarchitekturen für die Integration in spezifischen E-Business-Domänen
- Gestaltung und Modellierung Serviceorientierter Architekturen
- Standards und Standardtechnologien für die Softwarearchitektur
- Vorgehensmodelle für die architekturbasierte Evolution, Integration und Migration von Softwaresystemen

3 Organisation

Der Workshop wurde im Rahmen der folgenden GI-Fachgruppen und BMBF-geförderten Projekte organisiert:

- Fachgruppe Objektorientierte Software-Entwicklung (OOSE) der Gesellschaft für Informatik (http://wwwcs.upb.de/cs/ag-engels/ag_dt/GI/gi-fg219.htm)
- Fachgruppe Software-Architektur (FG-SWA) der Gesellschaft für Informatik (http://sdqtest.ipd.uka.de/research/fgswarch/)
- Fachgruppe Software-Reengineering (SRE) der Gesellschaft für Informatik (http://www.uni-koblenz.de/sre/)
- Projekt MINT – Modellgetriebene Integration von Informationssystemen (http://mint-projekt.de/)
- Projekt OrViA – Orchestrierung und Validierung integrierter Anwendungssysteme (http://www.orvia.de/)
- Projekt TransBS – Transformation monolithischer Business-Softwaresysteme in verteilte, workflowbasierte Client-Server-Architekturen (http://www.tu-chemnitz.de/informatik/PI/transbs/)

4 Workshopverlauf

Der Workshop konnte mit 32 Teilnehmern einen regen Zuspruch verzeichnen, davon waren 5 aus der Industrie, 5 aus Institutionen der angewandten Forschung und 22 aus Institutionen der Forschung und Lehre.

Die 8 angenommenen Beiträge deckten einen breiten Bereich der modellgetriebenen Entwicklung und angrenzender Gebiete ab. Zusätzlich zu Diskussionen nach jedem Vortrag gab es eine rege Beteiligung an der Abschlussdiskussion. Dabei wurden folgende Themen angeschnitten:

- Wann lohnt sich Model-Driven Development
    - Vorteile nicht nur in Wiederverwendung
    - Auch Time-to-Market
    - Legacy-Systeme und –Komponenten in Weiterentwicklung aufnehmen
    - Komplexität beherrschen - Kommunikationsmittel
    - Entscheidungen später treffen
- Modellgetriebene Entwicklung ohne Quellcode-Bearbeitung
- Synchronisierung von Modellen und Implementierung - zur Entwicklungszeit
    - Generieren von Code, ohne Notwendigkeit der manuellen Bearbeitung des Quellcodes
    - Annotationen
    - 3 zu 1 Transformation
    - Nur Quellcode betrachten, er enthält die Modellinformation eingebettet

    - Round Trip Engineering – Nebeneinander von direkt generiertem (aus Modell, one shot) und manuell erstelltem Code – bei letzterem wieder mit Synchronisationsproblem
- Korrektheit von Modellen … semiformal, formalisiert
- Modelle in der Sprache des Anwenders (Modellierung per Teach–in – End User Modelling)
- Simulation und Visualisierung von Modellen
- Validieren von Modellen und Testfälle aufnehmen
- Synchronisieren von Modell und Implementierung zur Laufzeit

4 Organisation

Veranstaltungsreihe

Der Workshop wurde als Nachfolger vorangehender Veranstaltungen organisiert. Die Workshops ORA2006 und ORA2007 „Architekturen, Objektorientierung und Reengineering“ wurden als gemeinsame Veranstaltungen der oben bereits genannten GI-Fachgruppen OOSE, FG-SWA und SRE durchgeführt. Als weiterer Vorläufer für diesen Workshop wurde im Februar 2007 ein gemeinsamer Workshop der Projekte MINT, OrViA und TransBS mit dem Titel „Transformation of Legacy-Software“ auf Schloß Dagstuhl veranstaltet. Weitere Workshops, die aus diesen Projekten heraus organisiert wurden, sind der Workshop MSI 2007, der die Synergien zwischen modellgetriebener Entwicklung, serviceorientierten Architekturen und IT-Management thematisierte und von Mitarbeitern des Projekts MINT organisiert wurde, sowie der OrViA-Workshop „Integration betrieblicher Informationssysteme“, der dem Erfahrungsaustausch auf dem Gebiet modellgetriebener Entwicklung diente, im Rahmen der 14. Leipziger Informatik-Tage.

Review-Prozess

Alle Beiträge wurden durch mindestens 3 Mitglieder des Programmkomitees anonym begutachtet. Im Workshopband wurden die entsprechend der Gutachterhinweise überarbeiteten Beiträge aufgenommen.

Programmkomitee

- Klaus-Peter Fähnrich, Universität Leipzig
- Rainer Gimnich, IBM Frankfurt
- Michael Goedicke, Universität Duisburg-Essen
- Peter Hänsgen, Intershop Research
- Wilhelm Hasselbring, Universität Oldenburg
- Florian Heidenreich, Technische Universität Dresden
- Maritta Heisel, Universität Duisburg-Essen

- Oliver Höß, Fraunhofer IAO
- Jürgen Meister, BTC AG
- Claus Möbus, Universität Oldenburg
- Klaus Pohl, Universität Duisburg-Essen
- Elke Pulvermüller, Universität Osnabrück
- Thomas Rauber, Universität Bayreuth
- Ralf Reussner, Universität Karlsruhe
- Matthias Riebisch, TU Ilmenau
- Gudula Rünger, Technische Universität Chemnitz
- Andreas Speck, Christian-Albrechts-Universität zu Kiel
- Ulrike Steffens, OFFIS
- Maik Thränert, Universität Leipzig
- Andreas Winter, Johannes-Gutenberg-Universität Mainz

Using collaborative business process models for the configuration of distributed workflow environments

Dirk Werth, Jörg Ziemann, Philipp Walter, Peter Loos

Institute for Information Systems
at the German Research Center for Artificial Intelligence
Saarbrücken
{dirk.werth, joerg.ziemann, philipp.walter, peter.loos}@iwi.dfki.de

Abstract: In order to realize collaborative value generation, it is necessary to define and implement collaborative business processes. Models have been proved in practice as an adequate instrument to fulfil this task and consequently models of such processes can be used to plan and steer collaborative business. This paper proposes a model based configuration of distributed workflows based on conceptual business process models. An outside-in approach is followed, where the overall collaboration is first defined conceptually but allows for a de-central implementation. The concept of developing the necessary conceptual model parts and distributed implementation was prototypically implemented using peer-to-peer and workflow technologies.

1 Introduction

An increasing fragmentation is ongoing in worldwide economical structures. Whereas in the past, enterprises tended to incorporate as much value generation as possible within themselves, nowadays enterprises strongly concentrate on their core competencies. In the consequence, the added value is generated by these specialized value units that intensively interact along the value chain order to together generate the intended product. The generation itself is performed through the synchronized execution of associated tasks by different, independent enterprises. Applying the ideas of process orientation to this constellation, this sequence of activities constitutes a *collaborative business process* (CBP) [Wert07]. Actual examples for such CBPs are in aerospace or automotive industry – there OEMs build extensive organizational networks with their 1st, 2nd and 3rd tier suppliers.

In order to manage the collaborative value generation, it is mandatory to manage the CBPs. In this respect, models have been proved in practice as a adequate instrument to fulfil this task and consequently CBP models can be used to plan and steer collaboration businesses. In regards to a lifecycle view on CBPs, they have to be modelled, executed and controlled [WeWa+07].

Since the development of models for CBP is fundamentally different from modelling intra-organizational business processes [LiGr+05], special methods for modelling collaborative business processes are required. However, there are several approaches to construct CBP models [GrLi+06, Werth06]. In the approach described in this paper an outside-in approach is pursued where all participants of a collaborative business possess a CBP model, which specifies all tasks and synchronization information needed to describe the value generation in the network. However, having this information does not mean to be able to realize it. It is necessary to transfer this information into the operations. Thus, from the ICT point of view, CBP models needs to be transformed to configure ICT execution system. In our approach, we focus on workflow engines as an archetype for ICT execution systems[1]. This paper presents a concept to use CBP models as the base for the configuration of a set of interworking workflow engines. As a result, CBP models specifying a collaborative business scenario can directly be transformed into the collaborative execution environment realizing the demanded behaviour.

2 Collaborative Business Process Models

Collaborative business processes are a special kind of (conventional) business processes. However, they imply special properties that strongly differ from the regular case. First, they are spanning over multiple organizations, because the generation of added value is performed through cross-organizational division of labour. Second, the individual business activities that compose the process clearly belong to a unique organization. Thus, having groups of activities that are processed in a direct sequence, we can state that the collaborative business process is partitioned into a set of these parts where each one is distinctively associated with an organization, fully controlled by it in the sense that it independently executes, administrates and manages it. Therefore those parts of cross-organizational business processes can be characterized as autonomous fragments.

The classical Business Process Management concept consists of three phases: business process design, implementation and controlling [ScJo02]. Business process implementation summarizes all operative steps that are necessary to execute a process which was modelled before. Business process controlling denotes all actions that aim towards measurement and analysis of processes. In the context of this paper, controlling is neglected. The resulting life cycle phases are conceived for a single organization. In the design phase, each process model is changed by a single modeller at a time. During the execution phase, the process is handled by a single execution system within a single organization. However, in collaborative environments, CBPs cannot be regarded as monolithic anymore, since different parts of them are designed, executed (and controlled) by independent organizations [LuBu99]. Consequently the lifecycle abruptly gets very complex and difficult to handle [WeWa+07]:

[1] In fact, workflow engines can act as an external interfacing proxy for all kind of ICT systems.

- The *design* (respectively modelling) task comprises multiple autonomous modellers that act independently and follow different goals. This results in self-contained parts of the collaborative business process. Therefore the process design can rather be characterized as an assembly task of autonomous process parts.

- The *execution* is distributed over different enterprises. Consequently there is no central processing engine. Instead each autonomous process part has its own independent processing engine, so classic workflow concepts and technologies have to be extended to match the new cross-organizational requirements [Schu02].

Thus, in order to efficiently execute CBPs, the CBP models created in the design phase have to be transferred into data that can be used by execution systems. This transformation procedure is the central task in the implementation phase of a CBP lifecycle and is described in the next section.

3 Transformation Procedure

Transferring the concept of business process management for single organizations to cross-organizational environments characterized by the involvement of multiple actors in the different phases requires the shift from a centralized paradigm to a support for distributed environments. For these actors a collective behaviour cannot be supposed. Thus the implementation phase requires new techniques that are different to those of the classical business process management and that incorporate the split activities. Therefore we do not focus on bilateral processing of business processes, but on end-to-end processes with a potentially huge number of contributors. The distributed execution of a business process starts with a common process model that all participants share and that is business oriented, i.e., its content is mainly conceptual and its purpose is organizational management. From this model every participant extracts those parts that he has to execute and augments them with arbitrary information he needs for execution, e.g., refinements of process sub-parts or execution context parameters (cf. Figure 1). Thus the business model is transformed into an IT-oriented workflow model, the main purpose of which is the execution of the contained process. The following section introduces the steps from the conceptual shared process model to de-central, technically detailed workflow models:

1. Splitting Up the Common Process Model: All activities in the common process model are annotated with the executing organization unit (“Company X”), or with an organization unit role (“Customer”) that can be mapped onto a concrete actor within the execution context. So the common model disaggregates in disjoint process model fragments that are executed by exactly one actor each. Because the process modules, which were composed to the common process model during the modelling phase, have interface descriptions, it is possible to define exactly which goods and which information must be transferred from one actor to another.

2. Apart from goods and information, the execution of the whole process devolves from one actor to another at an interface. Therefore it is necessary to define how the control of the process is transferred. At process junctions it may be even possible to split up process control or join multiple execution threads again.

3. Augmenting the Process Fragments: Execution of a process fragment usually requires considerable prearrangements on the part of the executing actor. Therefore the process fragment is first transformed from the modelling language into an executable language. Since the business process model is business oriented, it usually does not contain information about execution parameters, e.g., an IP address of an interface or authentication credentials for an ERP system. So it must be augmented with these missing execution parameters during or after transformation to the executable language. After that, the process fragment is contained in an executable workflow model.

4. Usually the common business process model disaggregates into multiple process fragments, each of which is transformed into a single workflow model. These workflow models are deployed to the respective IT systems then, which are finally configured with the contained information.

5. Executing the Process: Figure 1 shows how the whole top-level process is implemented by executing the workflow models of the process fragments which it consists of. After configuration of all involved systems this happens automatically, i.e., without interaction with individual process instances.

For step 3 we propose the use of Event Driven Process Chains (EPC) for design time modeling and the Business Process Execution Language for Web Services (BPEL) for the workflow configuration. As described in [ZiMe05] in a first step technical EPC models aligned to the BPEL syntax have to be derived in order to specify the interactions between the parties. Since BPEL can represent both a graph based and a block-oriented control flow (e.g. a containing a While loop), nearly all control flow elements of EPC can be transformed to BPEL. If the EPC functions represent interactions (e.g. “receive message”, “send message” etc.), they can be transformed to corresponding BPEL activities, if an EPC function represents an activity not captured by the BPEL syntax an individual Web Service has to be created that will be invoked by the BPEL process. In [ZiMe05] further details regarding the transformation of EPCs to BPEL are described.

Since the whole process is executed fragment-wise by multiple separate systems, there must be transition points from one system to another where execution is finished or suspended at the source system and perpetuated at the target system. This transition has two different aspects: data flow and control flow. Data transfer between separate IT systems is widely used already, e.g., between departments within a single organization. However, the transfer of process execution control and context via push and pull mechanisms is not common. Especially in split and join situations, e.g., when a simultaneous execution of multiple process parts on multiple systems begins or finishes, the process context must be duplicated and merged accordingly. During execution, performance data is gathered as a means for the next step: the controlling phase.

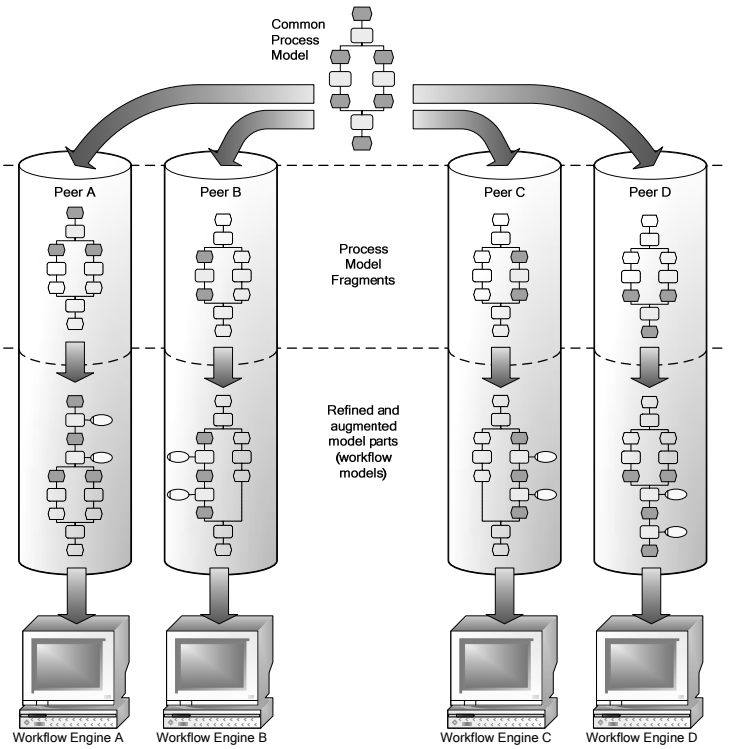


Fig. 1: Transformation procedure for collaborative business process execution

4 Conclusion and Outlook

In this paper, we have presented a concept for using CBP models for the configuration of distributed workflow environments including 1. distributed modelling, 2. execution and 3. controlling, that is already widely-implemented. In particular we addressed and ensured the continuous IT support of all three phases, the decision autonomy and secrecy demand of the participating organizations during the process.

The concept was prototypically implemented in the project "P2E2 – Peer-to-Peer Enterprise Environment" using peer-to-peer and workflow as base technologies [WeLo07]. Within this project we created two demonstration scenarios coming from the finance and the logistics sector. Though we succeeded to create an ICT supported approach to transfer CBP models into execution environments, this task still required a significant ammount of human support. Additional research is needed to explore more information that is not included in the CBPs models but can be gathered from other sources to further automate the transformation itself and reduce the manual work.

Currently the concepts created in P2E2 are extended in the R4eGov project[2], where model-driven, SOA focused CBP development is pursued based on combinations of outside-in (deriving internal processes from CBPs) and inside-out (combining existing internal processes to a CBP) approaches.

The work published in this paper is (partly) funded by the E.C. through the R4eGov project. It does not represent the view of E.C. or the R4eGov consortium, and authors are solely responsible for the paper's content.

References

[GrLi+06] Greiner, S. Lippe, T. Kahl, J. Ziemann, F. Jaekel: Designing and Implementing Crossorganizational Business Processes - Description and Application of a Modelling Framework. Interoperability for Enterprise Software and Applications Conference, Bordeaux 2006.

[LiGr+05] Lippe, S.; Greiner, U.; Alistair, B.: A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes. In: Nüttgens, M.; Mendling, J. (eds.): XML4BPM 2005 - Proceedings of the 2nd GI Workshop XML4BPM., 2005, pp. 7-22

[LuBu99] Ludwig, H., Bussler, C., Shan, M., & Grefen, P.: Cross-Organizational Workflow Management and Co-ordination - WACC99 Workshop Report. In: ACM ASIGGROUP Bulletin 20 (1999) 1.

[ÖsFl00] Österle, H., Fleisch, E., & Alt, R.: Business Networking - Shaping Enterprise Relationships on the Internet. Springer, Berlin et al 2000.

[RöSc01] Röhricht, J. & Schlögel, C.: cBusiness. Erfolgreiche Internetstrategien durch Collaborative Business. Addison-Wesley 2001.

[Schu02] Schulz, K.: Modelling and Architecting of Cross-Organizational Workflows, PhD Thesis. University of Queensland 2002.

[ScJo02] Scheer, A.-W. & Jost, W.: ARIS in der Praxis - Gestaltung, Optimierung und Implementierung von Geschäftsprozessen. Springer, Berlin et al 2002.

[WeLo07] Werth, D., Loos, P.: Zusammenfassender Abschlussbericht des Verbundprojektes P2E2 – Peer-to-Peer Enterprise Environment, Saarbrücken 2007.

[Wert06] Werth, D.: Kollaborative Geschäftsprozesse - Integrative Methoden zur modellbasierten Deskription und Konstruktion. Logos, Berlin 2006.

[Wert07] Werth, D.: About the Nature of Collaborative Business Processes, in: Arabnia, H., Bahrami, A. (eds.): Proceedings of the 2007 international conference on e-learning, e-business, enterprise information systems and e-government, Las Vegas, p. 252-257.

[WeWa+07] Werth, D.; Walter, P.; Loos, P.: Conceiving an Environment for Managing the Lifecycle of Collaborative Business Processes In: Oberweis, A. et al.: eOrganisation: Service-, Prozess-, Market-Engineering. Universitätsverlag Karlsruhe, Karlsruhe 2007, p. 805-822.

[ZiMe05] Ziemann, J. and Mendling, J.: Transformation of EPCs to BPEL – A pragmatic approach. 7th International Conference on the Modern Information Technology in the Innovation Processes of the industrial enterprises, Genoa 2005.

[2] www.r4egov.eu

MDA und KI: Domänenspezifische Modellierung und Umsetzung wissensintensiver Prozesse

Steffen Kruse, Malte Zilinski, Hilke Garbe, Claus Möbus

Learning Environments and Knowledge-based Systems,
Carl von Ossietzky Universität Oldenburg,
Ammerländer Heerstr. 114-118, 26129 Oldenburg
s.kruse, m.zilinski, h.garbe, claus.moebus@uni-oldenburg.de

Abstract: In dieser Veröffentlichung wird ein Verfahren vorgeschlagen, domänenspezifische, wissensintensive Prozesse zu erfassen und diese für einen Softwareentwicklungsprozess nutzbar zu machen. Dabei wird mit Hilfe von kognitiven Mustern das von Domänenexperten zur Verfügung gestellte Wissen strukturiert und in Form von spezialisierten Modellen aufbereitet. Es wird gezeigt, dass sich diese als *Computational Independent Models* (CIM) für die Softwareentwicklung nach *Model Driven Architecture* (MDA) nutzen lassen. Als Zielplattform für den MDA-Prozess werden von uns Methoden bzw. Sprachen aus dem Bereich der Künstlichen Intelligenz (KI) angestrebt, da sie für die Umsetzung wissensintensiver Prozesse besonders geeignet sind. Es wird beispielhaft gezeigt, wie dieses Verfahren auf die Studienplanung angewendet und auf die Planungsdomäne mit ausführbaren Planungssprachen abgebildet wird. Es wird gezeigt, wie die Integration bewährter Ansätze aus dem Bereich des Software Engineering (MDA) und der Künstlichen Intelligenz (in dem hier vorgestellten Beispiel Planungsprobleme) mithilfe von geeigneten Metamodellen für die Lösung komplexer und hoch-spezialisierter Probleme möglich ist. Dabei ist der entwickelte Ansatz innerhalb der Problemklasse der Planungsprobleme generisch und kann für verwandte Probleme in anderen Anwendungsdomänen wiederverwendet werden. Das vorgeschlagene Verfahren ermöglicht es den beteiligten Rollen (Domänenexperten, Software Engineers und KI-Experten) für sie spezialisierte Modelle zu verwenden.

1 Einleitung

Ansätze der modellgetriebenen Softwareentwicklung (*Model Driven Development*, MDD), insbesondere der modellgetriebenen Architektur (*Model Driven Architecture*, MDA) etablieren sich zunehmend als Standard im Bereich des Softwareengineering [GPR06]. Gerade die MDA bietet ein Vorgehen, das für die Entwicklung von Softwaresystemen benötigte Wissen ganzheitlich in Modellen zu fassen und über deren Ausdrucksstärke für unterschiedliche Domänen und Experten besser zugänglich zu machen. Gestützt wird die MDA von Methoden der Modelltransformation bzw. generativen Ansätzen, die es ermöglichen, die Erstellung von plattform- bzw. domänenspezifischen Artefakten weitestgehend zu automatisieren und die so einen hohen Grad an Wiederverwendung und Robustheit von entwickelten Lösungen bieten [MM03].
Die Generierung von Code aus Modellen lässt sich sehr einfach für statische Artefakte

eines Softwaresystems durchführen, zum Beispiel für Datenbankenstrukturen, Objektmodellen oder Schnittstellen. Dynamische Aspekte sind ungleich schwerer vollständig zu modellieren und werden häufig durch das manuelle Auffüllen von generierten Stubs eingefügt oder in sehr speziellen Templates für die Codegeneratoren verborgen. Im Umfeld von betrieblichen Softwaresystemen wird häufig ein anderer Ansatz gewählt: die dynamischen Eigenschaften werden als Geschäftsprozesse modelliert und zur Laufzeit von einer Geschäftsprozessengine interpretiert. Ein solches Vorgehen tritt häufig in Verbindung mit serviceorientierte Architektur (SOA)[MLM+06] auf. Im Sinne der MDA stellt die Geschäftsprozessengine eine Plattform dar, die es ermöglicht, das plattformspezifische Modell (PSM) oder sogar das plattformunabhängige Modell (PIM) als berechnungsvollständiges Modell ohne weitere Anpassungen auszuführen, da die plattformspezifischen Details in der Geschäftsprozessengine verborgen bleiben.
In vielen Arbeiten zu MDA liegt das Augenmerk auf den letzteren Schritten des MDA-Prozesses, also der Generierung von Code aus den PSMs und der Modelltransformation von PIM (und Plattformmodell) zu PSM. Eine formale Beschreibung des berechnungsunabhängigen Modells (CIM) und ein systematisches Vorgehen zur Erstellung (bzw. Transformation) des PIM aus dem CIM werden ausgelassen.
Das hier vorgestellte MINT-Vorgehensmodell stellt einen Ansatz vor, das Wissen von Domänenexperten zu erfassen und als CIMs für die MDA verfügbar zu machen. Dafür wird eine Planungsmethode aus dem Bereich der KI adaptiert. Darüber hinaus wird gezeigt, wie komplexe Probleme für den Entwicklungsprozess modelliert und spezialisierte Softwarekomponenten (in diesem Fall „off-the-shelf"-Planer) analog zu Geschäftsprozessengines als Plattformen für die Lösung verfügbar gemacht werden können. Für Software Engineers erleichtert dieses Vorgehen die Integration spezieller Sprachen, ohne in der Regel deren Details kennen zu müssen.

2 MINT Vorgehensmodell

Die Betrachtung wissensintensiver Prozesse im Kontext der MDA ist interessant, da sie in vielen Anwendungen auftreten, ihre Lösung aber nicht trivial ist. Im Bereich der KI existieren spezialisierte Sprachen, die sich für die Beschreibung und Lösung solcher Probleme bewährt haben, deren Anwendung bisher aber Expertenwissen erforderte. Das im Folgenden beschriebene MINT-Vorgehensmodell ermöglicht es Domänenexperten, Planungsprobleme fachlich zu beschreiben und diese für die Softwareentwicklung von spezialisierten planungsspezifischen Details zu abstrahieren.
Dieses MINT-Vorgehensmodell wird auf die Domäne Studienplanung angewandt. Als Ziel dient ein Softwaresystem, welches Studierenden entsprechend ihrer Studiensituation und ihren individuellen Anforderungen bei der Studienplanung unterstützt. Abbildung 1 zeigt eine Übersicht des intendierten Zielsystems und Abbildung 2 den gewünschten Geschäftsprozess als MINT XL[1]-Diagramm.

Zur Beschreibung der planungsspezifischen Prozesse auf CIM-Ebene werden Modell-

[1] MINT XL ist die im MINT-Projekt entwickelte domänenspezifische Sprache zur Modellierung von Geschäftsprozessen im MDA-Kontext (siehe `http://www.mint-projekt.de`)

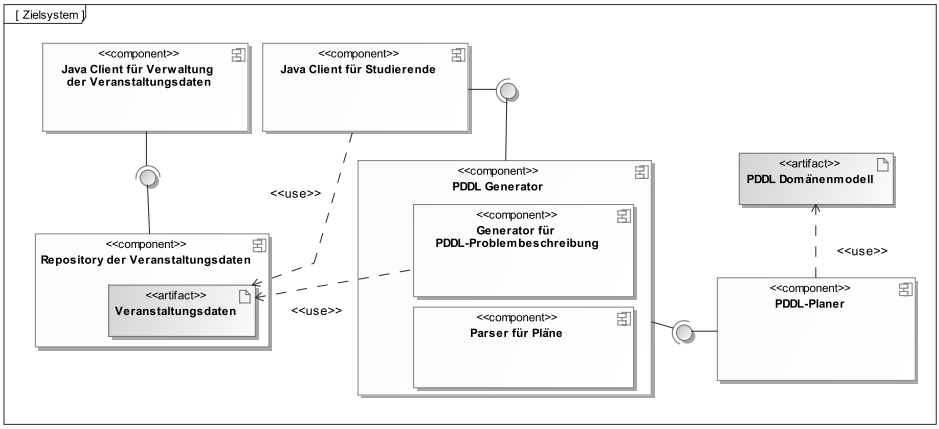


Abbildung 1: Intendiertes Zielsystem

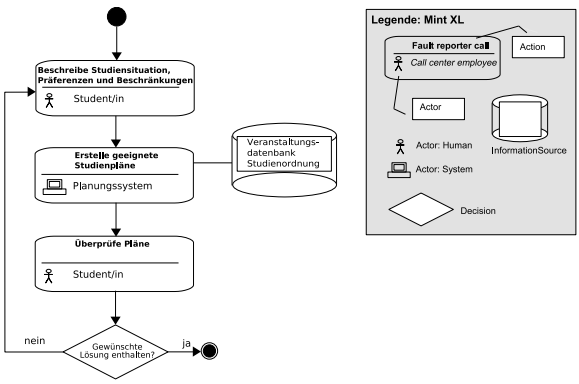


Abbildung 2: Geschäftsprozess der Studienplanung

muster für wissensintensive Prozesse (siehe [SAA+02, GRC98]) verwendet und angepasst. Zur Definition relevanter fachlicher Details werden die Musterelemente mit klassischen MDA-Methoden verfeinert. Die Studienplanung lässt sich auf ein Scheduling-Problem abbilden. Auf PIM-Ebene wird deshalb ein generisches Scheduling-Metamodell bereitgestellt, welches die fachlichen Anforderungen des CIM fasst und um Scheduling-Details erweitert. Das Studienplanungsproblem wird als partielle Instanz des Scheduling-Metamodells beschrieben. Das Problem wird auf plattformspezifischer Ebene als Planning Domain Definition Language (PDDL)-Planungsproblem (siehe [McD98, FL03]) oder als Constraint Satisfaction Problem (CSP) repräsentiert [RN04]. Somit ist das Modell deklarativ und maschinell lösbar.

2.1 CIM

CommonKADS ist eine Methodologie zur Erfassung, Strukturierung, Formalisierung und Operationalisierung von Wissen. Für die Modellierung wissensintensiver Prozesse stellt sie eine deklarative, für Domänenexperten verständliche Sprache bereit. Es werden vorgefertigte, generische Problemlösemuster, die „Template Knowledge Models“ (TKMs), für verschiedene wissensintensive Prozesse angeboten [SAA+02], welche von Experten für die Domänenmodellierung verwendet werden sollen. Da es sich bei den TKMs um generische Problemlösungen handelt, beschreiben sie den gewünschten Prozess in der Regel nicht genau den Anforderungen der Domäne entsprechend und es muss eine Adaption der Muster erfolgen. Das adaptierte instantiierte TKM *Planning*[2] (siehe Abbildung 3) eignet sich als Verfeinerung des in Abbildung 2 dargestellten Geschäftsprozesses, da dieser einen wissensintensiven Planungsprozess impliziert. CommonKADS definiert für alle Elemente des Musters Rollen, an denen erkennbar ist, welche Funktion die Daten bzw. Aktivitäten im Prozess übernehmen. Bei Daten wird zwischen dynamischen und statischen Rollen unterschieden: erstere bezeichnen Daten mit geringer und zweitere mit längerer Lebensdauer. Die Studienordnung hat in der Beispieldomäne mit einer Lebensdauer von mehreren Jahren die längste Gültigkeit. Für die Definition von Planungsproblemen wird im MINT-

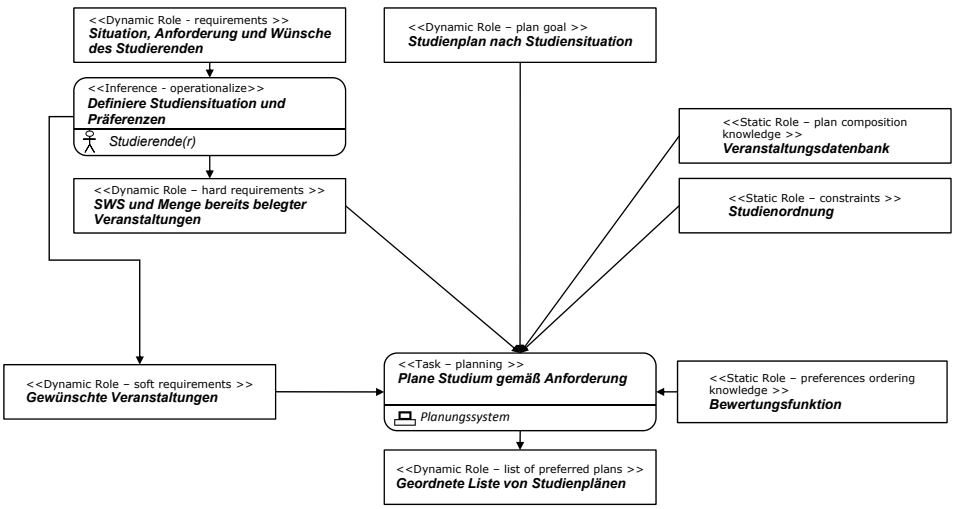


Abbildung 3: Planungsmuster

Vorgehensmodell die Domänenmodellierung über die Strukturierung der Domänenobjekte hinaus auf die Modellierung der statischen Struktur des Planungsproblems erweitert. Für die Domäne Studienplanung wird dafür das Studienordnungsmetamodell zur Verfügung gestellt.
Neben dem gezeigten Geschäftsprozess, dem Planungsmuster und dem Modell der Studienordnung, werden weitere Modelle erstellt, die die Struktur der Daten, sowie aller an-

[2]Das originale TKM *Planning* findet sich in der CommonKADS Methodologie[SAA+02].

deren Anforderungen an das restliche Softwaresystem enthalten. Diese werden gemäß des MDA-Ansatzes entwickelt, aber hier nicht weiter beschrieben.

2.2 PIM

Das gewählte Problem der Studienplanung lässt sich auf ein nicht-präemptives Scheduling-Problem mit harten und weichen Beschränkungen abbilden. Dabei wird das zeitliche Pensum des Studierenden als Kapazität einer Maschine im Sinne des Scheduling aufgefasst. Die in der Studienordnung definierten Module entsprechen Aktivitäten; mit Eigenschaften wie Dauer und Ressourcenverbrauch.
Für die plattformunabhängige Beschreibung von Scheduling-Problemen wurde ein generisches Metamodell entwickelt (siehe Abbildung 4), das sich an einer Scheduling-Ontologie orientiert [Raj04]. Dieses erlaubt die Abbildung der Domänenobjekte auf Jobs, Aktivitäten, Vorbedingungen und anderen Scheduling-spezifischen Eigenschaften, wie Fälligkeiten, Kapazitäten usw..
Mit diesem Scheduling-Metamodell wird gemäß der auf der CIM-Ebene definierten Anforderungen ein Modell erstellt, welches die berechnungsabhängigen, planungsrelevanten Aspekte der Domäne beschreibt. Dies wird über die Annotation des Studienordnungsmodells der CIM-Ebene anhand eines Scheduling-Profils erfolgen und der anschließenden automatisierten Transformation mittels noch dafür bereitzustellender Templates. Das Scheduling-Modell ist an dieser Stelle nur partiell instantiiert, da die dynamischen Anteile (Ressourcen) sich aus der Anfrage des Studierenden ergeben und erst zur Laufzeit bekannt sind. Zur Lösung des Scheduling-Problems müssen diese dem Planer zu Verfügung gestellt werden, dies erfolgt über den Einsatz eines Generators zur Laufzeit. Dieses Scheduling-Modell fügt sich zu anderen Modellen, die nach gewöhnlichen MDA Methoden entwickelt werden und zusammen alle Komponenten des Softwaresystems abbilden.

2.3 PSM

Als Plattform zur Lösung des Scheduling-Problems wird der PDDL-Planer *sgplan*[3] gewählt. Dieser wird zur Laufzeit in das Softwaresystem eingebunden. Als Eingabe benötigt der Planer die Studienordnungen, sowie die konkrete Anfrage des Studierenden (Studiensituation, gewünschte Veranstaltungen usw.). Das auf PIM-Ebene definierte Scheduling-Modell (Studienordnung) wird zu einer PDDL-Domänenbeschreibung transformiert.
Die dynamischen Anforderungen der Studierenden werden von einem Generator zur Laufzeit in eine PDDL-Problembeschreibung übersetzt und dem Planer zur Lösung übergeben. Die Ergebnisse werden wiederum übersetzt und dem umgebenen Softwaresystem zur wei-

[3] `http://manip.crhc.uiuc.edu/programs/SGPlan/index.html`

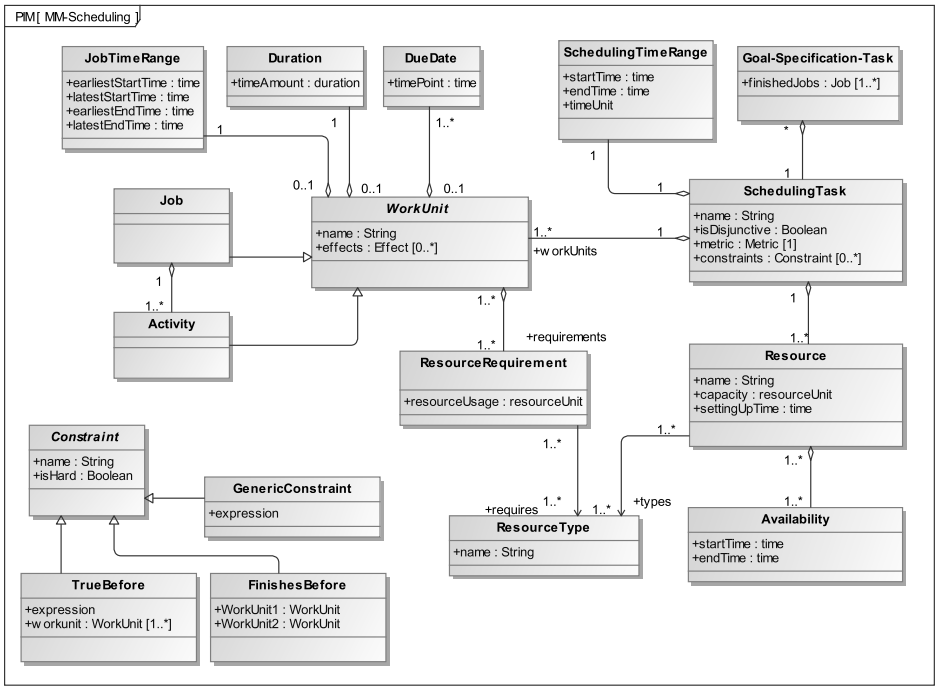


Abbildung 4: Scheduling Metamodell

teren Verarbeitung und schließlich dem Client zurückgegeben.
Als alternative Planungsplattform für Scheduling-Problem ermöglichen CSPs eine deklarative und maschinell lösbare Beschreibung. Entsprechende Modelle können mit ähnlichen Methoden aus dem generischen Scheduling-PIM transformiert werden. Eine Übersicht des resultierenden Zielsystems ist in Abbildung 1 dargestellt.

3 Zusammenfassung und Ausblick

Es wurde ein Verfahren vorgeschlagen mit dem wissensintensive Prozesse der Planungsdomäne modelliert und mittels MDA in ausführbaren PDDL-Code transformiert werden können. Es wurde ein Metamodell zur Definition von Studienordnungen auf CIM-Ebene entwickelt. Das erstellte, generische Metamodell auf PIM-Ebene erlaubt die Abbildung von Problemen unterschiedlicher Scheduling-Klassen. Bisher befindet sich eine Transformation zu einer PSM-Repräsentation in PDDL für nicht-präemptives Scheduling in der Konzeptionsphase. Eine Erweiterung auf andere Scheduling-Problemklassen ist geplant; dabei wird auch eine Überprüfung der Übertragbarkeit auf allgemeine Planungsprobleme langfristig angestrebt. Desweiteren soll das MINT-Vorgehensmodell an Planungsproblemen aus dem Energieversorgungssektor und damit an einer anderen Domäne evaluiert

werden.

Literatur

[FL03] Maria Fox und Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[GPR06] Volker Gruhn, Daniel Pieper und Carsten Röttgers. *MDA*. Springer, Heidelberg, 2006.

[GRC98] Karen Gardner, Alexander R. Rush und Michael Crist. *Cognitive Patterns: Problem-Solving Frameworks for Object Technology (Managing Object Technology Series)*. Cambridge University Press, 1998.

[McD98] D. McDermott. PDDL — the planning domain definition language, 1998.

[MLM+06] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, Rebekah Metz und Booz Allen Hamilton. *Reference Model for Service Oriented Architecture 1.0 Committee Specification 1*. OASIS, August 2006. Last seen: 25/09/2006.

[MM03] Joaquin Miller und Jishnu Mukerji. *MDA Guide Version 1.0.1*. Object Management Group (OMG), June 2003.

[Raj04] Dnyanesh Rajpathak. *A Generic Library of Problem-Solving Methods for Scheduling Application*. Dissertation, The Open University, 2004.

[RN04] Stuart Russell und Peter Norvig. *Künstliche Intelligenz. Ein moderner Ansatz*, Jgg. 2. Pearson Studium, 2004.

[SAA+02] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde und B. Wielinga. *Knowledge Engineering and Management*. The MIT Press, 2002.

Systematische Berücksichtigung von Abhängigkeitsbeziehungen bei Architekturentscheidungen

Sven Wohlfarth, Matthias Riebisch

TU Ilmenau, FG Prozessinformatik/Softwaresysteme, 98693 Ilmenau
{sven.wohlfarth|matthias.riebisch}@tu-ilmenau.de

Abstract: Architekturentscheidungen gehören zu den riskantesten Entscheidungen in großen Entwicklungsprojekten, da häufig nur unvollständige und unstrukturierte Informationen vorliegen. Gerade aufgrund der lückenhaft dokumentierten Abhängigkeitsbeziehungen steigen die mit einer Entscheidung verbundenen Risiken. Da ein Verfahren zur Erkennung und Berücksichtigung der Abhängigkeitsbeziehungen bei Architekturentscheidungen bislang fehlt, ist ein geeigneter Entscheidungsprozess zu entwickeln. Das Ziel ist es, die komplexen Abhängigkeitsbeziehungen trotz unvollständiger Informationsgrundlage sichtbar zu machen, im Detail zu analysieren und sie systematisch bei der Entscheidungsfindung zu berücksichtigen.

1 Notwendigkeit der Systematisierung der Entscheidungsfindung

In langfristigen Großprojekten sind Architektur- und Reengineering-Entscheidungen schwer zu treffen und mit erheblichen Risiken für den Auftraggeber verbunden, da Abhängigkeitsbeziehungen nur lückenhaft bekannt sind. Es ist daher ein Verfahren erforderlich, um die Abhängigkeitsbeziehungen sichtbar zu machen und sie systematisch und in der erforderlichen Tiefe bei der Entscheidungsfindung berücksichtigen zu können. Architekturentscheidungen in Großprojekten erfolgen unter speziellen Bedingungen, wie einer hohen Zahl von Entwicklern, räumlich getrennten Teilprojekten und paralleler Entwicklung von getrennten Versionen. Ein hoher Termindruck und eine Vielzahl an Veränderungen erschweren die Aufrechterhaltung einer vollständigen und konsistenten Dokumentation zur Ermittlung der Abhängigkeitsbeziehungen. Das Risiko von Fehlentscheidungen für das betroffene Unternehmen besteht in hohem Korrekturaufwand sowie hohen finanziellen Verlusten bei Ausfall eines geschäftskritischen Systems.

Ein strukturiertes, systematisches Vorgehen bei der Entscheidungsfindung hilft, die Komplexität zu beherrschen und Fehler zu vermeiden. Im ersten Schritt eines solchen Vorgehens werden die Ziele und Rahmenbedingungen erhoben und strukturiert. Dabei sind Widersprüche und Konkurrenz-Beziehungen zwischen den Zielen aufzulösen. Danach werden vom Architekten Lösungsansätze zur Veränderung der Architektur zwecks Erfüllung der Ziele entwickelt. Im Fall existierender Komponenten oder Softwaresysteme sind diese auf Abhängigkeitsbeziehungen zwischen den Komponenten hin zu analysieren. Dann sind die alternativen Lösungsansätze anhand der erwarteten Beiträge zur Zielerreichung und möglicher Seiteneffekte auf andere Systemteile zu bewerten, um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können. Diese Schritte (siehe

die Zusammenfassung in Abbildung 1) folgen den Vorschlägen des Vorgehens zur Entscheidungsfindung aus der präskriptiven Entscheidungstheorie [EW03].

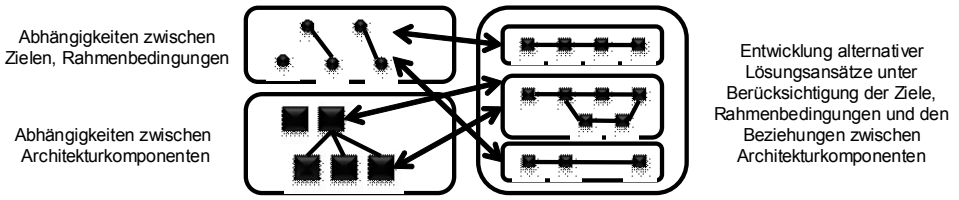


Abbildung 1: Berücksichtigung der Abhängigkeitsbeziehungen bei der Entscheidungsfindung

2 Typen von Abhängigkeitsbeziehungen

Abhängigkeitsbeziehungen spielen für das Verständnis von Zusammenhängen durch Architekten und Entwickler eine besonders wichtige Rolle. Sie werden für Aufstellung, Bewertung und Prüfung von Lösungsansätzen benötigt. Zu ihrer Modellierung und Beschreibung in verschiedenen Methoden der Softwaretechnik wurde eine Vielzahl verschiedener organisatorischen und technischen Abhängigkeitsbeziehungen verwendet, die teilweise unter verschiedenen Bezeichnungen geführt werden. Für ein einheitliches Verständnis wurde aus dieser Menge in Anlehnung an [KL05] eine für die Entscheidungsfindung relevante Auswahl von Typen getroffen (Tabelle 1). Eine Fokussierung auf wenige Beziehungstypen ist notwendig, um Abhängigkeiten besser analysieren und ihren Einfluss auf Entscheidungen bewerten zu können.

Tabelle 1: Typen von Abhängigkeitsbeziehungen

Beziehungen zwischen	**Komponenten**	**Maßn. zur Architektur-Veränderung**	**Ziele, Rahmenbedingungen**
Komponenten	‚bedingt durch' ‚benutzt von' ‚aggregiert mit' ‚spezialisiert zu'	‚Teil von'	‚in Konflikt mit' ‚erfüllen vollständig/ teilweise'
Maßnahmen zur Architektur-Veränderung		‚erzwingen' ‚in Konflikt mit' ‚zusammengefasst'	‚beeinflusst von'
Ziele, Rahmenbedingungen			‚sich ergänzend' ‚miteinander konkurrierend'

Die in Tabelle 1 aufgeführten Beziehungstypen beziehen sich auf folgende Elemente von Architekturentscheidungen (siehe Abbildung 1):

- **Ziele und Rahmenbedingungen** können sich ‚ergänzen', in einer Ziel-Mittel-Beziehung stehen, ‚konkurrieren' und sich gegenseitig ausschließen. In diesen Fällen muss ein Kompromiss gefunden werden. Beispiele: Budget-Restriktionen, Vorgaben hinsichtlich der zu verwendenden Frameworks.
- **Architekturkomponenten** können sich gegenseitig ‚benutzen' oder in einer ‚aggregiert mit' / ‚spezialisiert zu' Beziehung stehen. Diese Beziehungen geben wich-

tige Hinweise darauf, welche Komponenten unabhängig und welche Komponenten ausschließlich in Kombination verändert, ausgetauscht oder ergänzt werden können. Beispiele: Softwarekomponenten wie Broker, Event-Handler oder Datenbanken.

- **Maßnahmen zur Veränderung der Architektur** können miteinander ‚in Konflikt' stehen oder sich gegenseitig ‚erzwingen'. Gerade umfassende Architekturveränderungen müssen aus Risiko-, Ressourcen- und Steuerungsaspekten heraus in kleinen Schritten durchgeführt werden. Da zum Zeitpunkt der Entscheidung eine Sequenz oder eine Folge mehrerer Maßnahmen gleichzeitig zu berücksichtigen ist, ist die Kompatibilität der Maßnahmen zueinander sicherzustellen. Beispiele sind der Austausch, die Zusammenfassung oder die Teilung einer Komponente.

3 Systematische Berücksichtigung der Abhängigkeitsbeziehungen bei der Entscheidungsfindung

Um die in Tabelle 1 dargestellten Beziehungen bei der Entscheidungsfindung zu berücksichtigen sind zunächst die Beziehungen zwischen den **Zielen und Rahmenbedingungen** und zwischen den **Architekturkomponenten** zu identifizieren.

Zur Illustration werden hier Ausschnitte eines größeren Projektes angeführt, die eine Architektur-Veränderung am Web-Content-Management-System Typo3 behandeln. Typo3 ist eine Open Source PHP-Webapplikation zur Gestaltung und Konfiguration von komplexen Webseiten. Typo3 ist modular aufgebaut; neben einem Typo3-Kern existieren über 3.000 sogenannte Extensions, die einen Einsatz von Typo3 in vielen Anwendungsdomänen ermöglichen [FR05].

Widerspruchsfreie Strukturierung der Ziele und Rahmenbedingungen

Zur Strukturierung ‚konkurrierender' oder ‚sich ergänzender' Ziele wird in der ersten Phase ein Ziel-Mittel-Diagramm erstellt. Das Vorgehen hierzu entstammt aus der Entscheidungstheorie [EW03]. Dabei werden die Ziele zerlegt, verfeinert, nach Instrumental- und Fundamentalzielen klassifiziert und in eine baumartige Ziel-Mittel-Struktur eingeordnet. An der Spitze stehen die Fundamentalziele, welche durch die verschiedenen Instrumentalziele verfeinert werden. Diese Art der Strukturierung der Ziele erleichtert die anschließende systematische Priorisierung der Ziele. Die Zerlegung wird fortgesetzt, bis jedes Ziel in eindeutigen Ziel-Mittel-Beziehungen zu den anderen Zielen steht.

Im Beispiel Typo3 besteht das Ziel in einer Steigerung der Portabilität bezüglich des verwendeten Datenbank-Managementsystem DBMS. Ein zentraler Schwachpunkt von Version 3.6 ist die Bindung an das DBMS mySQL, was die Verwendung in einem Unternehmensumfeld stark einschränkt, weil hier häufig andere DBMS etabliert sind. Um als Fundamentalziel die Portabilität in der Version 4 zu verbessern, sind geeignete Lösungsansätze zur Restrukturierung des Datenbankzugriffs zu entwickeln und zu vergleichen (siehe Abbildung 2). Ein konkurrierendes Fundamentalziel sind die Antwortzeitverhalten. Da kurze Antwortzeiten für Typo3 ein kritischer Erfolgsfaktor sind, darf die Restrukturierung keinesfalls zu einer Verschlechterung führen. Eine Beeinträchtigung der Antwortzeiten ein hohes Risiko dar, da ein Wechsel zu einem anderen Content-Management-System mit erheblichem Aufwand verbunden wäre.

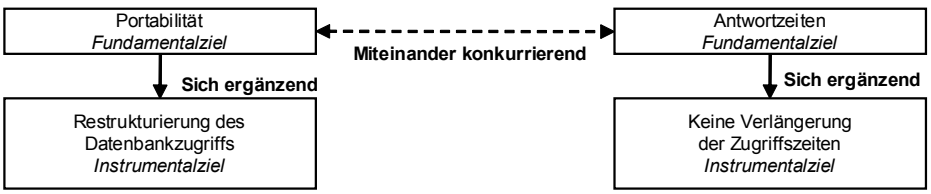


Abbildung 2: Erstellung eines widerspruchsfreien Ziel-Mittel-Diagramms

Um die Identifizierung und Strukturierung der Rahmenbedingungen, z.B. die Ermittlung des Budgets oder der technischen Bedingungen der Implementierungsplattform, systematisch durchzuführen, kann eine Checkliste eingesetzt werden. Sie umfasst einerseits die Rahmenbedingungen tabellarisch in gruppierter Form und andererseits die zugehörigen absoluten Werte oder Wertebereiche. Mit der Abarbeitung der Checkliste werden die Rahmenbedingungen zunächst vollständig erhoben; sich ergänzende Rahmenbedingungen werden dabei hervorgehoben. Die Auflösung von Konkurrenzbeziehungen erfolgt im zweiten Schritt durch eine Priorisierung der Rahmenbedingungen im Hinblick auf die Zielerreichung und die Anpassung der Rahmenbedingungen mit der geringeren Priorität.

Ermittlung der Abhängigkeitsbeziehungen zwischen Architekturkomponenten

Um die häufig nur lückenhaft bekannten Beziehungen zwischen den Architekturkomponente identifizieren zu können, ist eine Beschreibung der relevanten Teile der Architektur erforderlich. Um den Aufwand für die Beschreibung zu begrenzen, ist eine Abgrenzung des Betrachtungsraums anhand der Ziele und ggf. bereits bekannter Schwachstellen notwendig. Die Beschreibung des relevanten Teils als Modell erfolgt mittels üblicher Architektur-Beschreibungssprachen. Um die Abhängigkeitsbeziehungen zu ermitteln, wird ein Architektur-Review mittels einer szenariobasierte Architekturanalyse durchgeführt. Als Grundlage kann hierfür die Architecture Tradeoff Analysis Method ATAM [KK00] angewendet werden. Je nach Ziel werden Szenarien auf das Modell der Softwarearchitektur angewendet, um die Abhängigkeiten zwischen Komponenten erkennen zu können (siehe Abbildung 3). Zur Bewertung des Veränderungsaufwands kann die Architecture Level Modifiability Analysis ALMA [BL04] verwendet werden, die geeignete Szenarien für ATAM definiert. Sofern der Betrachtungsraum groß ist, können zur

Identifizierung von Schwachstellen zunächst Architekturmetriken ausgewertet werden, z.B. das Distanzmaß [SL99] oder Kopplungs-/Kohäsionsmetriken [SA01].

Beim Beispiel Typo3 mit seiner modularen Architektur aus einem Kern ‚Typo3-Core' und der Erweiterung durch ‚Extensions'(siehe Abbildung 3) ist eine Vielzahl von Abhängigkeitsbeziehungen zu berücksichtigen, da alle ‚Extensions' auf die Datenbank zugreifen. Der Fokus liegt jedoch auf den Beziehungen zwischen ‚Typo3-Core' und dem ‚Database-Abstraction-Layer', da die Programmierrichtlinien die Nutzung des Layers für den Zugriff auf die Datenbank vorschreiben. Dieser Layer stellt eine Vielzahl von Funktionen zur Verfügung, um Daten aus einer MySQL-Datenbank abzurufen, zu manipulieren und um neue Daten einzustellen. Bei der Restrukturierung müssen diese Funktionen in identischer Art und Weise benutzt werden können, um die Kompatibilität einer Vielzahl von ‚Extensions' zu gewährleisten.

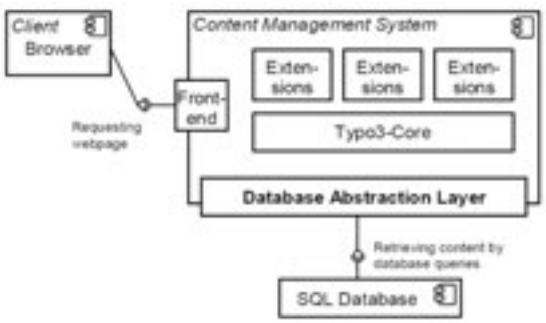


Abbildung 3: Modulare Architektur von Typo3 als UML-Komponentendiagramm

Systematische Entwicklung und Verfeinerung alternativer Lösungsansätze

In der dritten Phase erfolgt die Entwicklung und Verfeinerung der alternativen Lösungsansätze, mit denen die Architektur zur Erfüllung der Ziele verändert wird. Hierfür sind vom Architekten zunächst geeignete Lösungsansätze wie Best Practices, Muster, Stile und ähnliches auszuwählen. Zur Bewertung werden je alternativem Lösungsansatz die notwendigen Maßnahmen zur Architektur-Veränderung identifiziert. Dabei wird eine Reihenfolge der Maßnahmen festgelegt, wobei Abhängigkeiten zwischen den Maßnahmen berücksichtigt werden. Im Falle von Konflikten wie beispielsweise fehlenden Voraussetzungen muss die Reihenfolge verändert oder es müssen zusätzliche Maßnahmen eingefügt werden. Im Beispiel der der Typo3-Architekturveränderung muss vor Beginn der Restrukturierungen sichergestellt werden, dass alle Datenbankzugriffe über den ‚Database-Abstraction-Layer' erfolgen.

Zur Ermittlung der Reihenfolge wird die Architektur-Veränderung durch eine simulative oder tatsächliche Implementierung der Maßnahmen schrittweise untersucht. Dabei werden der Grad der Zielerreichung, die Einhaltung der Rahmenbedingungen und der Implementierungsaufwand ermittelt. Daraufhin können eine Bewertung der Lösungsansätze und eine rationale Entscheidung erfolgen.

Für die Restrukturierung des Datenbankzugriffs im Beispiel werden zwei Lösungsansätze entwickelt (siehe Abbildung 4). Der erste Ansatz ist die Ergänzung des ‚Database-Abstraction-Layer' (siehe Abbildung 3) um spezielle Bibliotheken, die angepasste Funktionen für den Zugriff auf DB2 oder Oracle beinhalten. Die existierenden Funktionen des Layers zur Datenmanipulation werden mit den Funktionen der Bibliotheken verknüpft, um einen Zugriff die verschiedenen Datenbank-Systeme zu gewährleisten. Der andere Lösungsansatz sieht die Ergänzung des ‚Database-Abstraction-Layers' durch die Komponente ADOdb [FR05] vor, die Funktionen zur Transformation von SQL-Statements bietet. Die generierten SQL-Statements werden durch ADOdb transformiert und so an die entsprechenden Datenbanken angepasst. Bei beiden Lösungsansätzen stehen die ursprünglichen Funktionen des Layers unverändert zur Verfügung.

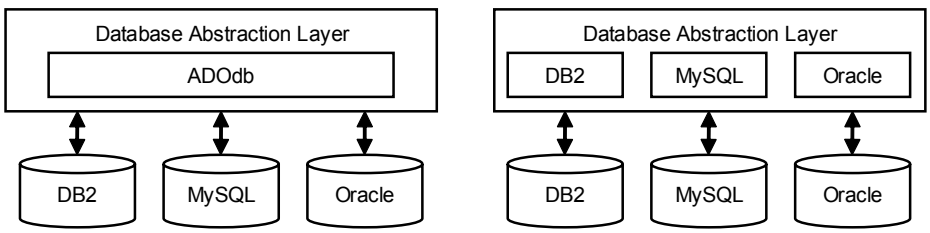


Abbildung 4: Gegenüberstellung der alternativen Lösungsansätze

Die Lösungsansätze unterscheiden sich im Hinblick auf den Implementierungsaufwand und der Antwortzeiten. Während ADOdb als Softwareprodukt direkt integriert werden kann, sind die Bibliotheken mit entsprechenden Zugriffsmethoden erst noch zu entwickeln. Andererseits führen die mit der ADOdb-Alternative verbundenen Transformationen zu einer spürbaren Erhöhung der Zugriffszeiten. Szenarioanalysen des prototypischen Architekturmodells zeigen, dass bei jedem Datenbankzugriff zwei Transformationen durchführt werden; bei einer Datenabfrage wird das SELECT-SQL-Statement *und* der Rückgabewert transformiert, um z.B. den Timestamp an das Oracle-Format anzupassen. Da Antwortzeiten ein kritischer Erfolgsfaktor für Typo3 sind, wird der erste Lösungsansatz der Entwicklung von datenbankspezifischen Bibliotheken besser bewertet und weiter verfolgt. Die weitere Entwicklung von Typo3 als Open-Source-Projekt wird von dieser projektbezogenen Entscheidung jedoch nicht berührt.

4 Reduzierung des Analyse- und Bewertungsaufwands

Das hier vorgestellte Vorgehen zur Entscheidungsfindung stellt in Bezug auf den Softwareentwicklungsprozess zusätzlichen Aufwand bei Architekturentwicklung dar, der eine Termin- und Budgeteinhaltung erschwert. Die Aktivitäten Modellerstellung, Ermittlung der Abhängigkeitsbeziehungen, szenariobasierte Bewertung von Lösungsalternativen und schrittweise Untersuchung von Veränderungen müssen deshalb jeweils in ihrem Aufwand minimal gestaltet werden. Während der Entscheidungsfindung muss dazu identifiziert werden, welche Elemente und Beziehungen direkt von der erforderlichen

Architekturveränderung betroffen sind und welche große Auswirkungen auf die restlichen Teile der Architektur haben. Nur diese Teile werden dann beschrieben, untersucht und bewertet. Die Entscheidung, was im Einzelfall relevant ist oder was weggelassen werden kann, erfordert einen Kompromiss, der Einfluss auf die Qualität der Entscheidung und auf die Wahrscheinlichkeit von Fehlentscheidungen hat. Je höher das Risiko einer Entscheidung ist, desto höherer Aufwand kann und sollte in eine systematische Entscheidungsfindung investiert werden.

Im Fall der Typo3-Architekturveränderung wurde zur Reduzierung des Analyse- und Bewertungsaufwands die Untersuchung auf die Abhängigkeitsbeziehungen zwischen dem ‚Typo3-Core' und des ‚Database-Abstraction-Layers' beschränkt. Die Beziehungen zwischen den ‚Extensions' und der Datenbank wurden nicht betrachtet, da die Programmierrichtlinien die Nutzung des Layers vorschreiben. Um Seiteneffekte aufgrund einer Verletzung dieser Richtlinie auszuschließen, wird jedoch vor Beginn der Restrukturierung deren Einhaltung überprüft. Eine grobe Abschätzung des Aufwands gegenüber dem Risiko im Anwendungsumfeld ließ sowohl den Zeit- als auch den Kostenaufwand gerechtfertigt erscheinen.

6 Resümee

Durch die konsistente Berücksichtigung der Abhängigkeitsbeziehungen wird es möglich, Architekturentscheidungen systematisch und nach rationalen Gesichtspunkten zu treffen, um Fehlentscheidungen und deren negativen Konsequenzen zu vermeiden. Die Fokussierung auf häufig verwendet Beziehungstypen stellt sicher, dass der Entscheidungsprozess in komplexen Großprojektumgebungen sinnvoll eingesetzt werden kann. Trotzdem sollte bei der Durchführung der Entscheidungsfindung jederzeit das Aufwand-Nutzen Verhältnis berücksichtigt werden.

Literaturverzeichnis

[BL04] Bengtsson, P.; Lassing, N.; Bosch, J.; van Vliet, H.: Architecture-level modifiability analysis (ALMA), Journal of Systems and Software - Volume 69 - Nr 1-2 - Januar 2004, Seiten 129 - 147.

[EW03] Eisenführ, F.; Weber, M.: Rationales Entscheiden, Springer, Berlin u.a., 2003.

[FR05] Fritz, R., Hinderink D. et al: Typo3: Enterprise Content Management, Packt, 2005.

[KK00] Kazman, R.; Klein, M.; Clements, P.: ATAM - Method for Architecture Evaluation, URL: http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf, Abruf 2008-10-01.

[KL05] Kruchten, P.; Lago, P.: Building Up and Reasoning About Architectural Knowledge. In: 2nd International Conference on the Quality of Software Architectures (QoSA 2006)

[SA01] Shereshevsky, M.; Ammari, H. et al: Information Theoretic Metrics for Software Architectures, In: '25th Annual International Computer Software and Applications Conference', IEEE Computer Society Los Alamitos (CA, USA), 2001, Seiten: 151 - 160.

[SL99] Simon, F.; Löffler,S. et al: Metrics Based Refactoring, In: 'Proceedings of the Fifth European Conference on Software Maintenance and Reengineering', IEEE Computer Society, Los Alamitos (CA, USA), 1999, Seiten: 30 - 38.

Grafische Validierungsregeln am Beispiel von EPKs

Sven Feja[1], Daniel Fötsch[1], Sebastian Stein[2]

[1] Institut für Informatik, Christian-Albrechts-Universität zu Kiel
24098 Kiel
svfe, daf@informatik.uni-kiel.de

[2] IDS Scheer AG
Altenkesseler Str. 17, 66115 Saarbrücken
sebastian.stein@ids-scheer.com

Abstract: Als Grundlage für die modellgetriebene Softwareentwicklung werden korrekte und qualitativ hochwertige Modelle bereits auf abstrakter fachlicher Ebene vorausgesetzt. Um die Qualität der Modelle zu sichern, existieren verschiedene Ansätze wie Modellprüfen (*Model Checking*). Modellprüfen ist eine formale Methode, mit der Modelle gegen eine Spezifikation validiert werden. Das Modell des Modellprüfers ist in der Regel ein mit Eigenschaften beschrifteter Zustandsautomat. Die Spezifikation der Prüfregeln wird textuell mit temporaler Logik auf Ebene des Zustandsautomaten definiert. Diese formale Definition ist für Fachmodellierer wenig geeignet.

In diesem Beitrag wird deshalb eine grafische Notation für Validierungsregeln am Beispiel der Geschäftsprozessnotation Ereignisgesteuerte Prozesskette (EPK) entwickelt. Dies ermöglicht dem Fachmodellierer, Anforderungen an die EPK zu formulieren. Die Anwendung der grafischen Notation wird anhand eines Fallbeispiels aus der E-Government-Domäne kurz demonstriert.

1 Einleitung

Die modellgetriebene Softwareentwicklung im Rahmen einer Model Driven Architecture [MM03] ist ein viel versprechender Ansatz, um die dringend benötigte Effizienzsteigerung in der Softwareentwicklung zu erreichen. Grundlage bilden dabei Modelle, die über Transformationen und Anreicherungen mit weiteren Details schrittweise bis hin zur ausführbaren Software verfeinert werden. Im Unternehmenskontext kommt häufig Standardssoftware zum Einsatz, die für den speziellen Einsatz angepasst werden muss. Anstatt die Software grundlegend neu zu entwerfen, werden bestehende Komponenten miteinander kombiniert und lediglich fehlende Komponenten neu entwickelt.

Aktuell ist zu beobachten, dass bestehende Komponenten im Rahmen einer serviceorientierten Architektur (SOA) [MSJL06, z. B.] als Web Services gekapselt werden und die Verknüpfung der einzelnen Komponenten über eine Orchestrierungssprache wie die Business Process Execution Language (BPEL[1]) erfolgt. Verschiedene Autoren [KTS05, DSS08]

[1] http://www-128.ibm.com/developerworks/library/ws-bpel/

zeigen, dass es sinnvoll ist die Anforderungen zunächst auf Ebene der Geschäftsprozesse durch Fachmodellierer zu erfassen und aus dieser fachlichen Prozessbeschreibung mittels einer wie in [SI07] vorgestellten Transformation die Web Service Orchestrierung in BPEL automatisch abzuleiten.

Voraussetzung für eine solche modellgetriebene Softwareentwicklung sind korrekte und qualitativ hochwertige fachliche Modelle. Zur Prüfung müssen sowohl syntaktische als auch semantische Prüfregeln formuliert werden. Während syntaktische Prüfregeln und deren Validierung von vielen Modellierungswerkzeugen bereits unterstützt werden, wird in diesem Beitrag gezeigt, wie semantische Prüfregeln auf der Fachebene definiert werden können und wie eine Validierung der Geschäftsprozessmodelle anhand dieser Prüfregeln möglich ist. Dabei ist es essentiell, dass die Definition der fachlichen Prüfregeln auf für Fachmodellierer verständliche Weise erfolgt. Bisherige Verfahren haben diese Forderung weitestgehend ignoriert. Bei unserem Ansatz nimmt diese Forderung eine zentrale Stellung ein. Unsere Lösung wurde prototypisch im ARIS SOA Architect umgesetzt und anhand eines Geschäftsprozesses aus der E-Government-Domäne [LRK06] erprobt.

2 Hintergrund

Für die Umsetzung unseres Validierungsansatzes wird die Modellprüfung verwendet. Die Modellprüfungstechnik hat ihre Ursprünge in der Hardware-Prüfung. In der aktuellen Forschung finden Modellprüfer immer stärker auch bei der Validierung von Software ihren Einsatz. Deutlich wird dies bspw. in der steigenden Anzahl von Beiträgen zum Thema Software-Modellprüfung in wissenschaftlichen Veranstaltungen wie der ECOOP Workshop Serie zum Thema Software-Modellierung und Validierung [PSD+01, SPC+02, VDSSP+03].

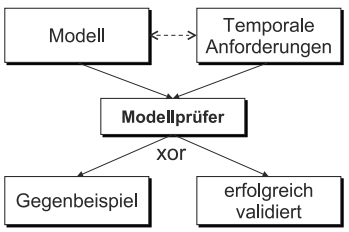


Abbildung 1: Grundprinzip des Modellprüfens.

Durch Modellprüfer können Abläufe modelliert und gegenüber Spezifikationen verglichen werden, die in temporaler Logik formuliert sind. Es wird die Erfüllbarkeit bestimmter temporaler Aussagen durch das vorliegende Modell bewiesen oder durch ein Gegenbeispiel widerlegt (siehe Abbildung 1). Damit lässt sich eine höhere Genauigkeit in Bezug auf die jeweilige Aussage erzielen als gegenüber einem manuellen Test. Trotzdem wird das modellierte System an sich nicht bewiesen, da nicht alle Anforderungen vollständig überprüft werden, bzw. es kann auch eine Prüfung über eine unvollständige Menge von Systemanforderungen erfolgen.

Überprüfbare Prozessmodelle oder Workflows sind bspw. UML Modelle oder Ereignisgesteuerte Prozessketten [KNS92]. Für UML existiert mit der Object Constraints Language (OCL) die Möglichkeit statische Abhängigkeiten in UML Diagrammen zu überprüfen. Dynamische Abhängikeiten lassen sich mit OCL nicht direkt prüfen. Dies wird erst durch die in [FM02], [DKR00] und [RM99] vorgestellten Erweiterungen ermöglicht. Für Geschäftsprozessmodelle sind bisher nur syntaktische Prüfungen durch die verwendeten Modellierungswerkzeuge möglich. Daher soll im Folgenden ein Verfahren der semantischen Überprüfung von Geschäftsprozessmodellen am Beispiel der EPK vorgestellt werden. Dennoch haben die folgenden Aussagen auch für andere Prozessmodelle wie UML ihre Gültigkeit.

Um die Modellprüfung für Geschäftsprozessmodelle nutzen zu können, müssen diese in das Modell eines Modellprüfers transformiert werden. Das Inputmodell ist typischerweise ein Zustandsautomat, dem je nach Modellprüfer ein linear diskretes (z. B. SPIN [Hol97]) oder verzweigt diskretes Zeitmodell (z. B. SMV [McM93]) zugrunde liegt. In diesem Beitrag wird das letztere Zeitmodell betrachtet, da es Aussagen über mehrere Pfade gleichzeitig ermöglicht. Eine Spezifikationssprache mit der sich temporale Aussage über verzweigte diskrete Zeitmodelle treffen lassen, ist *Computational Tree Logic* (CTL).

CTL ist eine temporale Logik, die von Clarke und Emerson entwickelt wurde [CE81]. CTL erweitert die Boolesche Logik, um verschiedene temporale Operatoren, die zeitliche Aussagen über Zustände erlauben. Dazu bietet CTL die Operatoren X (next), F (future) und G (globally), die spezifizieren, dass etwas im nächsten/irgendeinem/allen folgenden Zuständen gilt. Mit dem binäre Operator U kann ausgedrückt werden, dass etwas solange gilt, bis etwas anderes gilt. Diese temporalen Operatoren werden immer paarweise mit den Pfadquantoren A (auf allen Pfaden) und E (für mindestens ein Pfad) verwendet.

3 Grafische Modellierung von Prüfregeln

Zur Validierung von Geschäftsprozessen durch Modellprüfen bedarf es Validerungsregeln in einer geeigneten grafischen Notation auf der Abstraktionsebene des Geschäftsprozesses, wie z. B. die Ereignisgesteuerte Prozessketten. Die bisher notwendige textuelle Definition der temporalen Logik auf der Ebene des Modellprüfer-Modells ist für Fachmodellierer unbrauchbar, da es zum einen erheblichen Einarbeitungsaufwand erfordert und zum anderen die Kenntnis über das vom EPK überführte Modell des Modellprüfers verlangt. Daher muss eine grafische Notation an die im Geschäftsprozessmodell verwendete Notation angelehnt sein, um temporale Aussagen über die dort verwendeten Elemente zu möglichen. Die grafische Notation der von uns verwendeten temporalen Logik CTL, wird als *Graphical Computation Tree Logic* (kurz G-CTL) bezeichnet. Die Elemente der Notation sind in Abbildung 2 dargestellt.

Die G-CTL-Operatoren bestehen neben den Booleschen Operatoren aus zwei Komponenten, zum einen aus dem temporalen Operator und zum anderen aus einem Platzhalter. Die temporalen Operatoren entsprechen den in Abschnitt 2 vorgestellten. Die Platzhalter dienen zur Anlehnung an das Geschäftsprozessmodell. Dazu können die Felder a und b in

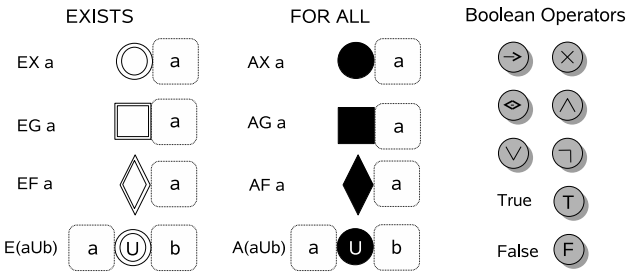


Abbildung 2: Die G-CTL-Operatoren.

Abbildung 2 mit den Elementen der EPK belegt werden. Der Name dieser Notation lautet daher EPK-G-CTL. Die Erstellung von grafischen Validierungsregeln mit den in Abbildung 2 vorgestellten Elementen wird im nächsten Abschnitt demonstriert.

4 Vorgehen und Anwendung

In Abbildung 3 ist das Gesamtvorgehen als vereinfachte EPK in fünf Schritten dargestellt. Neben den Einzelschritten ist weiterhin angegeben, in welchem Werkzeug im Rahmen unserer prototypischen Umsetzung jeder Schritt bearbeitet wird.

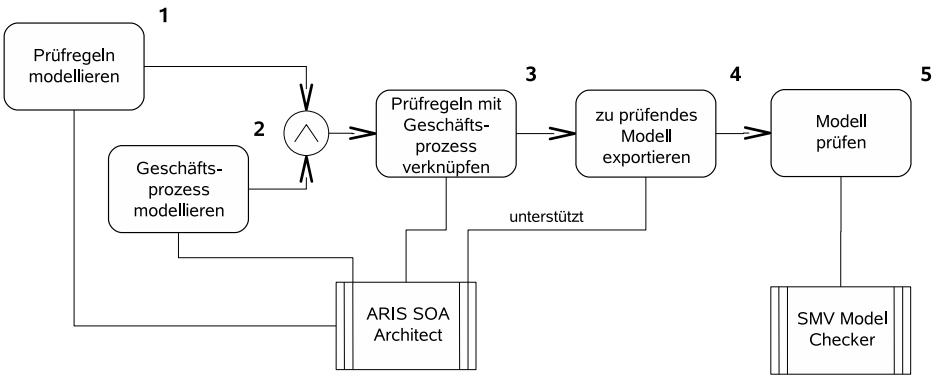


Abbildung 3: Übersicht des Gesamtvorgehens.

Schritt 1 und 2 des Vorgehens können dabei gleichzeitig ausgeführt werden, da sie zunächst voneinander unabhängig sind. In Schritt 1 erstellt der Fachmodellierer die Prüfregeln. Idealerweise handelt es sich dabei um allgemeingültige Regeln, die für alle Geschäftsprozesse des Unternehmens gelten, etwa dass neue Verträge immer von zwei unabhängigen Mitarbeitern autorisiert werden müssen. Parallel erfolgt in Schritt 2 die Modellierung der Geschäftsprozesse. In Schritt 3 wird angegeben, welche Prüfregel für welchen

Geschäftsprozess gilt. Dazu werden die grafischen Objekte, die die Regeln repräsentieren, in das Geschäftsprozessmodell eingebettet. In Schritt 4 wird das Modell und die Prüfregeln exportiert und mit Hilfe eines Transformators [FP07] in das für den Modellprüfer notwendige Format automatisch überführt. Im abschließenden Schritt 5 findet die Validierung statt. Im Folgenden wird das Vorgehen anhand eines Beispiels aus dem E-Government demonstriert.

Als fachlicher Hintergrund für die folgende Regel dient die *Einfache Melderegisterauskunft* (eMRA) [LRK06]. Diese ermöglicht es einem Benutzer über ein Portal des Einwohnermeldeamtes die Daten eines Bürgers zu überprüfen. Für die eMRA gilt in jedem Bundesland das Landesmeldegesetz (LMG), in welchem bspw. eine Protokollierung von versendeten Nachrichten an den anfragenden Benutzer festgelegt ist. Im konkreten Fall muss nach einer erfolgreichen Auskunftsberechtigungsprüfung das Absenden einer Antwort im Verlauf des modellierten Geschäftsprozesses protokolliert werden. In Form der temporalen Logik CTL würde dies bedeuten:

$$AG(Auskunftsberechtigung_liegt_vor \rightarrow AF(Protokollierung)). \quad (1)$$

Wobei `Auskunftsberechtigung_liegt_vor` dem Ereignis *Auskunftberechtigung liegt vor* und `Protokollierung` der Funktion *Protokollierung* in dem E-Government Geschäftsprozess entsprechen.

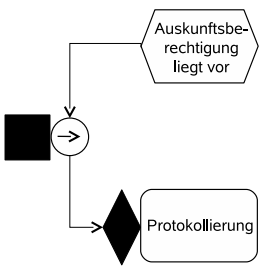


Abbildung 4: Beispiel einer EPK-G-CTL-Regel.

Die Modellierung mit EPK-G-CTL ist in Abbildung 4 dargestellt. Die Regel besteht aus einer Wenn-Dann-Beziehung, die man in der vorgegebenen Pfeilrichtung liest. Die den Implikationspfeil und der Funktion *Protokollierung* voran gestellten temporalen Operatoren gelten jeweils für das entsprechende Element. Das heißt konkret, immer wenn das Ereignis *Auskunftberechtigung liegt vor* im Prozess eintritt (`AG`), dann muss in der Zukunft auf allen Pfaden mindestens einmal (`AF`) die Funktion *Protokollierung* ausgeführt werden. Unter Anwendung dieser Regel kann nun ein vorhandenes Geschäftsprozessmodell auf die Einhaltung dieser Anforderung überprüft werden.

5 Zusammenfassung

Die syntaktische und semantische Validierung von fachlichen Anforderungsmodellen ist eine Grundvoraussetzung für die modellgetriebene Softwareentwicklung. Es existieren eine Reihe von Ansätzen auf Basis von Modellprüfen, die aber wenig praktikabel sind, da im Gegensatz zu den grafischen Fachmodellen die Formulierung der Prüfregeln lediglich textuell in Temporaler Logik erfolgt. Dieser Beitrag zeigt, wie Fachmodellierer sowohl Geschäftsprozessmodelle als auch die zugehörigen Prüfregeln modellieren können. Für die Darstellung der Geschäftsprozessmodelle werden Ereignisgesteuerte Prozessketten (EPK) verwendet und für die Darstellung der Prüfregeln wurde eine an die EPK angelehnte Notation entwickelt.

Eine erste Evaluierung unseres Vorgehens fand am Beispiel der *Einfachen Melderegisterauskunft* aus dem E-Government statt. Weiterhin existiert eine prototypische Werkzeugunterstützung auf Basis des ARIS SOA Architect. Unsere momentane Arbeit konzentriert sich auf der Evaluierung mit weiteren Beispielszenarien und einer Überarbeitung der Notation. Hier ist speziell von Interesse, wie die Modellierung der Prüfregeln sauber in die Gesamtmethodik zur Geschäftsprozessmodellierung integriert werden kann.

Literatur

[CE81] E. M. Clarke und E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, Hrsg., *Proceedings of Logic of Programs Workshop*, LNCS 131, Seiten 52 – 71. Springer Verlag, 1981.

[DKR00] Dino Distefano, Joost-Pieter Katoen und Arend Rensink. On a Temporal Logic for Object-Based Systems. In Scott F. Smith und Carolyn L. Talcott, Hrsg., *Formal Methods for Open Object-Based Distributed Systems IV - Proc. FMOODS'2000, September, 2000, Stanford, California, USA*. Kluwer Academic Publishers, 2000.

[DSS08] J. Drawehn, S. Stein und P. Schneider. Ein Verfahren zur Orchestrierung von IT Services. In *Modellgetriebene Softwarearchitektur – Evolution, Integration und Migration*, München, Februar 2008.

[FM02] Stephan Flake und Wolfgang Mueller. An OCL Extension for Real-Time Constraints. In *Object Modeling with the OCL*, Seiten 150–171, 2002.

[FP07] D. Fötsch und E. Pulvermüller. Constructing higher-level Transformation Languages based on XML. In H. Fujita und D. Pisanelli, Hrsg., *New Trends in Software Methodologies, Tools and Techniques*, Jgg. 161 of *Frontiers in Artificial Intelligence and Applications*, Seiten 269–284, Rome, Italy, November 2007. IOS Press.

[Hol97] G. J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279 – 295, May 1997.

[KNS92] G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". Arbeitsbericht Heft 89, Institut für Wirtschaftsinformatik Universität Saarbrücken, 1992.

[KTS05] S. Kühne, M. Thränert und A. Speck. Towards a methodology for orchestration and validation of cooperative e-business components. In M. J. Rutherford, Hrsg., *7th GPCE Young Researcher Workshop*, Seiten 29–34, 2005.

[LRK06] J. Lehmann, W. Rotzoll und S. Kühne. Analyse der E-Government-Domäne Meldewesen am Beispiel des Dienstes „einfache Melderegisterauskunft“. In K.-P. Fähnrich, S. Kühne, A. Speck und J. Wagner, Hrsg., *Integration betrieblicher Informationssysteme: Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering*, Jgg. IV of *Leipziger Beiträge zur Informatik*, Seiten 20–30. Eigenverlag Leipziger Informatik-Verbund (LIV), Leipzig, September 2006.

[McM93] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MM03] J. Miller und J. Mukerji. MDA Guide. Bericht omg/2003-06-01, Object Management Group (OMG), Juni 2003. Version 1.0.1.

[MSJL06] J. McGovern, O. Sims, A. Jain und M. Little. *Enterprise Service Oriented Architectures*. Springer, Dordrecht, The Netherlands, 2006.

[PSD+01] E. Pulvermüller, A. Speck, M. D'Hondt, W.D. De Meuter und J.O. Coplien. Proccedings of Workshop on Feature Interaction in Composed Systems, ECOOP 2001, Budapest, Hungary. Bericht No. 2001-14, Universität Karlsruhe, 2001.

[RM99] Sita Ramakrishnan und John McGregor. Extending OCL to Support Temporal Operators. In *Proceedings of the 21st International Conference on Software Engineering (ICSE99) Workshop on Testing Distributed Component-Based Systems, LA, May 16 - 22, 1999*, 1999.

[SI07] S. Stein und K. Ivanov. EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In Wolf-Gideon Bleek, Jörg Raasch und Heinz Züllighoven, Hrsg., *Software Engineering 2007*, Jgg. 105 of *Lecture Notes in Informatics (LNI)*, Seiten 75–80, Hamburg, Germany, März 2007. Gesellschaft für Informatik (GI).

[SPC+02] A. Speck, E. Pulvermüller, M. Clauß, R. Van Der Straeten und R. Reussner. Proccedings of Workshop on Model-based Software Reuse ECOOP 2002, Malaga, Spain. Bericht No. 2002-4, Universität Karlsruhe, 2002.

[VDSSP+03] R. Van Der Straeten, A. Speck, E. Pulvermüller, M. Clauß und A. Pleuß. Proccedings of Workshop on Correctness of Model-based Software Composition (CMC) ECOOP 2003, Darmstadt. Bericht No. 2003-13, Universität Karlsruhe, 2003.

Towards Evaluating Maintainability Within Model-Driven Environments*

Thomas Goldschmidt, Jens Kübler
FZI Forschungszentrum Informatik
Karlsruhe, Germany
goldschmidt, kuebler@fzi.de

Abstract: Model Driven Software Development (MDSD) has matured over the last few years and is now becoming an established technology. One advantage that is promoted by the MDSD community is the improved maintainability during the systems evolution over conventional development approaches. Compared to code-based development (meta-)models and transformations need to be handled differently when it comes to maintainability assessments. However, a comprehensive analysis of the impact of the model-driven development approach on the maintainability of a software system is still lacking. This paper presents work towards the finding of appropriate approaches and metrics for measuring the maintainability and evolution capabilities of artefacts within model-driven environments. We present our first steps and further ideas on how to tackle this problem.

1 Introduction

Maintainability metrics for object oriented (OO) systems have been around for several years now and are extensively explored [LH93]. However, with the advent of model-driven software development (MDSD)[VS06] a new area emerges where maintainability also needs to be assessed. Maintenance is one of the major cost factors in software development [Mas05]. MDSD claims to provide aid in reducing these costs. Though, currently it still needs to be validated whether MDSD really improves the maintainability of a system. One step that is needed to fulfil this issue is the identification of comprehensive, comparable and automatically collectable metrics for this paradigm.

For models and transformations the conventional code metrics can not be applied to obtain meaningful results. For example, it does not make sense to take the lines of code metric of generated code into account to measure its maintainability. First steps were made on how to assess maintainability within UML models by partly adopting metrics from object oriented development [MSZ+04] and employing new UML specific metrics [XKL04]. It remains unclear how these metrics can be generalized and also utilized to measure the maintainability of domain specific languages (either based on UML profiles

*This work is part of the research project *Model Driven Integration of Business Information Systems (MINT)* [SSMG06]. MINT is supported by the German *Federal Ministry of Education and Research* in the scope of the *Forschungsoffensive Software Engineering 2006*.

or own metamodels). Additionally changes that are made to the transformations in such an environment also need to be taken into account. Another important aspect that needs to be accounted for is the comparability of the metrics to those of non-model-driven approaches to decide if and in which scenarios such an approach is beneficial to maintainability costs.

Based on a case study that utilizes all different MDSD concepts, such as profiles, metamodels, model to model and model to code transformations we will set up a goal, question, metrics (GQM [BCR94]) plan to identify potential metrics.

The structure of this paper is as follows. First, we briefly discuss related work and indicate some shortcomings of existing approaches for maintainability assessment. In section 3, we present our ideas on how the maintainability can be measured. Section 4 describes the setup of our case study which will be used during further development of our approach. We finish with an overview on the current status of our efforts and a concluding summary in section 5 and 6.

2 Related Work

Maintainability of object oriented systems from an UML perspective with respect to metrics has been investigated in several works ([MP07],[MSZ+04],[XKL04] and evaluated in [SMK02] such as the number of classes, attributes or number of generalisations. However, the metrics are founded solely on models which are based on the UML metamodel and do not cover the application of profiles to these models, other metamodels and transformations. Preliminary work has been done on how complex semantic constraints formulated in OCL affect maintainability in [RGPM05]. As OCL is a normative reference for the Query/View/Transformation specification (QVT) [Obj] these findings may be included in analysis to transformations as well. In one of our previous case studies we already identified some scenario based metrics using the GQM approach that allow for the comparison of maintainability of model-driven and non-model-driven techniques [GWR07, GWR08]. Unfortunately these metrics are only partially generalizable for our purpose as they are specific to the given scenario.

3 Ideas for the Assessment of Maintainability in a Model-driven Environment

Several definitions of maintainability have been proposed i.e. in IEEE 610.12 or ISO/IEC 12207 and 14764. The work of [OH92] splits software maintainability into three main aspects : (1) The management practices, (2) the operational hardware and software environment that is involved with the software system under maintenance, and (3) the target software system itself. Instead of tampering with a precise definition we tend to use a constructive approach that focuses on actions necessary for maintenance. In the broader sense of software quality ISO 9126 defines a quality model which is therfore our source of inspiration for a GQM plan. To address maintainability in this context we analyse the

general structure of the model-driven paradigm which consists of (meta-)models and transformations. Therefore we propose to establish detailed metrics on these two components through our GQM plan.

3.1 Metrics on Models

Starting with models we recognize a separation of domain aspects from functional aspects which can be found i.e. in [BGeA]. This can be implemented in various ways such as applying different profiles (that is stereotypes and tagged values) to annotate functional information to the platform independent models or keeping two separate models which are woven together. On one hand structural metrics for both aspects should be straight forward to define on the other hand we regard semantics like OCL expressions more complicated to measure. Suppose a functional aspect "persistent" on class "A" which would imply the semantic constraint, that all properties of that class are to be persistent. At first glance it is unclear up to which degree of abstraction semantic constraints are more maintainable than explicit syntactic constructs. In this case verbosity conflicts with conciseness as described in [MVG05]. We consider an unified model of metrics for domain and functional aspects useful, if it is parametrizable so that for each aspect classification of good and bad design is possible.

3.2 Metrics on Transformations

To cover maintainability on transformations two approaches are considered. The first approach is based upon model comparison and similarity between original and transformed models which is suitable for M2M transformations. Similarity is an indicator for complexity of transformations and hence it relates to the maintainability of the transformation. [Ruf03] gives a good overview and classification how model similarity can be addressed.

The second approach focuses on transformations themselves. [CH03] outlines several transformation features that may be used to identify components of a transformation that are crucial to its maintainability. Regardless of the transformation technique used, i.e. declarative or imperative, transformations can be treated as rule systems with rules consisting of a left hand side and a right hand side . Traceability of rules is considered an important feature of transformations as it connects source and target models which offers impact analysis that is analysis of subsequent changes caused by an initial change. [CH06]. Therefore, we think support for traceability by the transformation engine influences the maintainability of a transformation substantially. Further transformation features are considered feasible such as rule organisation which covers reuse and modularity mechanisms that are outlined in [CH06]. [KvdBJ06] introduces the notion of patterns with respect to transformations. Such patterns may affect maintainability metrics both in a harmful and leveraging way and should be carefully inspected. Suppose a façade pattern in a conventional code centric software project that accumulates several methods from various parts

of the application. Although more maintainable this pattern may result in poor cohesion values which indicate bad maintainability.

3.3 Measurement Process

To extract meaningful metrics for maintainability from the numerous metrics available we aim to setup a GQM plan [BCR94]. Through the goal driven definition of the metrics according to this plan we are able to directly draw conclusions from our gathered metrics. A question we consider relevant is which changes in the source or target model cause what amount of adoption in the transformation. Switching from one platform specific model to another one is often stated as a feature of model driven software development so we plan to analyse the effort of switching models.

Integrating the measurement process into the transformation is a significant issue to tackle as it supports the engineer with useful information during the development process. Therefore we need to rely on openness and extendabilty of the tools we consider to use.

4 The Case Study

To be able to ensure the comparability of our metrics and also to identify the possible interdependencies between the different aspects in a model-driven environment we are currently developing a case study based on the domain of persistency frameworks. Within this case study we will use a standard Model Driven Architecture (MDA) process as proposed by the Object Management Group (OMG) in [Obj03]. This process includes the definition of platform independent and platform specific models as well as transformations to models and code.

On the one hand we chose the domain of persistency frameworks because there are many different frameworks for this purpose that can serve as platform specific target model for the model transformation and on the other hand because there are several platform independent functional aspects that are to be handled by a model transformation to a PSM. The transformation rules should reveal themselves as basic which enables easy analysis of frequently arising transformation structures.

In our concrete study we use UML to model the platform independent model, which allows us to re-use, or at least directly compare existing model metrics and decide whether they are usable in a broader sense. Furthermore we use the profiling mechanism of UML to create profiles for important aspects within our target domain, such as object relational mapping, object identity or validation. This profile will be kept platform independent. As an example for the platform specific model we will use the Java Persistence API (JPA) part of the EJB3 specification [jsr].

To define the profiles and create the necessary models we plan to use MagicDraw UML [mag] as it provides extended support for both UML models and profiles. We intend to

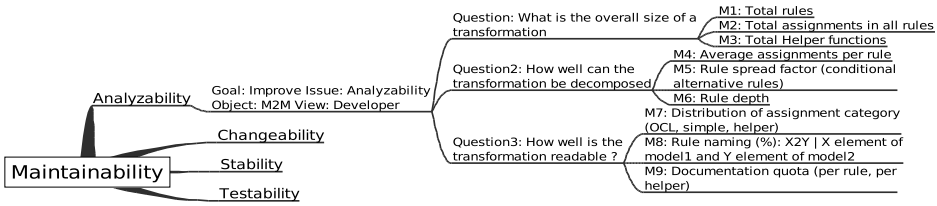


Figure 1: Snapshot of Goal Question Metric Plan for Analysability in M2M Transformations

focus our efforts for M2M transformations on the relational part of the QVT specification where mediniQVT [med] recently became available as an implementation. For M2T transformations we intend to use openArchitectureWare (OAW) [oaw] as it is a quite mature and well established framework with templating, modularization and aspect weaving capabilities.

Once usable metrics are found applying those to legacy software systems will be subject to our research so that a real world example is provided.

5 Current Status

We defined a PIM profile for persistence, a PSM profile for JPA and implemented a model to text transformation which generates the appropriate Java classes. We are yet to implement a M2M transformation which should provide an "easy" example for a transformation. As measurement process plan we set up a GQM plan which currently covers analysability. We identified metrics for models, stereotypes, differences between models as well as metrics for M2M transformations which are partly shown in figure 1. M2T metrics are not considered at this time but may be transferred from our M2M findings in future research. Our metrics application is implemented as a Eclipse plugin using OCL as query language for models. As many constituents of the MDA approach are available as Ecore models such as OCL, QVT or UML, we have flexible measurement capabilities at hand. Upcoming investigation will tackle classification and aggregation of measures and the issue of how to establish validity of our findings.

6 Conclusion

In this paper we presented our ideas on how to approach the problem of evaluating maintainability within a model-driven development environment. We identified that not only the maintainability of the developed domain models but also the metamodels, profiles and transformations that are used have a great influence on the effort of maintaining such a system. We will use a GQM plan applied within a case study to find these metrics.

References

[BCR94] V. Basili, G. Caldeira, and H. D. Rombach. *Encyclopedia of Software Engineering*, chapter The Goal Question Metric Approach. Wiley, 1994.

[BGeA] Krishnakumar Balasubraman, Aniruddha Gokhale, and et Al. Weaving Deployment Aspects Into Domain-Specific Models. last access: 20.12.2007.

[CH03] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[CH06] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, July 2006.

[GWR07] T. Goldschmidt, J. Winzen, and R. Reussner. Evaluation of Maintainability of Model-driven Persistency Techniques. In *IEEE CSMR 07 - Workshop on Model-Driven Software Evolution (MoDSE2007)*, 2007.

[GWR08] T. Goldschmidt, J. Winzen, and R. Reussner. A Case Study Evaluation of Maintainability and Performance of Persistency Techniques. In *Proceedings of the 30th international conference on Software engineering ICSE '08*, 2008. to appear.

[jsr] JSR 220: Enterprise JavaBeans 3.0. last access: 20-12-2007.

[KvdBJ06] Ivan Kurtev, Klaas van den Berg, and Frédéric Jouault. Evaluation of Rule-based Modularization in Model Transformation Languages illustrated with ATL. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06)*, pages 1202–1209, Dijon, France, 2006. ACM Press.

[LH93] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *J. Syst. Softw.*, 23(2):111–122, 1993.

[mag] MagicDraw UML. `http://www.magicdraw.com/`. Last Access: 20-12-2007.

[Mas05] Dieter Masak. *Legacysoftware*. Springer, 2005.

[med] ikv++ mediniQVT. last access: 20-12-2007.

[MP07] Jacqueline A. McQuillan and James F. Power. On the Application of Software Metrics to UML Models. In *Models in Software Engineering*, Lecture Notes in Computer Science, pages 217–226. Springer, 2007.

[MSZ+04] Haohai Ma, Weizhong Shao, Lu Zhang, Zhiyi Ma, and Yanbing Jiang. Applying OO Metrics to Assess UML Meta-models. In *UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings*, volume 3273 of *Lecture Notes in Computer Science*. Springer, 2004.

[MVG05] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. In *Proc. Int'l Workshop on Graph and Model Transformation*, 2005.

[oaw] openArchitectureWare. `http://www.openarchitectureware.org/`. Last Access: 20-12-2007.

[Obj] Object Management Group. MOF QVT Final Adopted Specification. http://www.omg.org/cgi-bin/doc?ptc/05-11-01.pdf.

[Obj03] Object Management Group, Framingham, Massachusetts. *MDA Guide Version 1.0.1*, June 2003.

[OH92] P. Oamn and J. Hagemeister. Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on*, 1992.

[RGPM05] L. Reynoso, M. Genero, M. Piattini, and E. Manso. Assessing the impact of coupling on the understandability and modifiability of OCL expressions within UML/OCL combined models. *Software Metrics, 2005. 11th IEEE International Symposium*, pages 10 pp.–, 19-22 Sept. 2005.

[Ruf03] Raimi A. Rufai. New Structural Similarity Metrics for UML Models. Master's thesis, King Fahd University of Petroleum and Minerals, Dharan 31261, Saudi Arabia, January 2003.

[SMK02] Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors. *Conceptual Modeling - ER 2002, 21st International Conference on Conceptual Modeling, Tampere, Finland, October 7-11, 2002, Proceedings*, volume 2503 of *Lecture Notes in Computer Science*. Springer, 2002.

[SSMG06] Niels Streekmann, Ulrike Steffens, Claus Möbus, and Hilke Garbe. Model-Driven Integration of Business Information Systems. In *Softwaretechnik-Trends*, volume 26, pages 9–13, November 2006.

[VS06] Markus Völter and Thomas Stahl. *Model-Driven Software Development*. Wiley, 2006.

[XKL04] Baowen Xu, Dazhou Kang, and Jianjiang Lu. A Structural Complexity Measure for UML Class Diagrams. In *Computational Science - ICCS 2004, 4th International Conference, Kraków, Poland, June 6-9, 2004. Proceedings*, volume 3036 of *Lecture Notes in Computer Science*. Springer, 2004.

UPPAAL-Modelle als ausführbare Spezifikation in Java

Michael Goedicke, Moritz Balz, Michael Striewe
Universität Duisburg-Essen
{michael.goedicke, moritz.balz, michael.striewe}@s3.uni-due.de

Abstract: Zustandsautomaten können als Modelle umfassend spezifiziert, simuliert und validiert werden. Werden sie im Programmcode umgesetzt, bleibt der formale Bezug zu diesen Modellen jedoch in der Regel nicht vollständig erhalten. Dieser Beitrag untersucht, wie Modellentwicklung am Beispiel von UPPAAL und die Umsetzung im Java-Programmcode gleichzeitig geschehen können, indem eine Methode zur Spezifikation von Zustandsautomaten im Programmcode skizziert wird. Dieser Programmcode kann sowohl als Zustandsautomat ausgeführt, als auch ad hoc in ein UPPAAL-Modell transformiert werden.

1 Einleitung

Modellgetriebene Technologien und Werkzeuge unterstützen die abstrakte Spezifikation von Zustandsautomaten von der Modellierung über Simulation bis zum Model Checking. So können bereits zum Entwicklungszeitpunkt der Modelle Fehleranalysen erfolgen und eine korrekte Funktionsweise sichergestellt werden. Durch die Hierarchisierung von Automaten kann zudem eine Strukturierung der Softwarearchitektur erreicht werden.

Auf einer weniger hohen Abstraktionsebene erfolgt allerdings die Umsetzung im Programmcode in Programmiersprachen wie etwa Java. Gleich, ob der Code von Hand geschrieben oder auf Basis des Modells generiert wird, bleibt die Semantik des Modells im ausführbaren Code nicht erhalten. Das Wissen über den Bezug zwischen Modell und Code ist nur indirekt vorhanden, im günstigen Fall über eine (oft nicht-formale) Dokumentation, im schlechteren Fall nur über das implizite Wissen der Entwickler [TG03]. Diese Problematik setzt der Verwendung von Modellen in der Softwareentwicklung enge Grenzen: Der häufig verwendete Ansatz der Code-Generierung aus Modellen macht eine Rückkopplung ebenso unmöglich wie die manuelle Entwicklung. Bekannte Techniken des Model Round-Trip Engineering [SK04] versuchen, Inkonsistenzen zu benennen und durch geeignete Manipulationen zu beheben und damit bereits mehr zu leisten als reine Codegenerierung und Reverse-Engineering. Auch dann sind allerdings Inkonsistenzen aufgrund der unterschiedlichen Abstraktionsebenen von Code und Modellen nicht vollständig zu vermeiden.

Ausgangspunkt für diesen Beitrag ist ein in einem Industrieprojekt entwickelter Lastgenerator [BSGT03], dessen Steuerung der Lasterzeugung über Zustandsautomaten realisiert und zu dem Zweck in UPPAAL [LPY97] modelliert und dokumentiert wurde. Um den Zusammenhang zwischen Programmcode und Modell zu erhalten stellt sich hier die Auf-

gabe, eine Umsetzung von Zustandsautomaten im Programmcode zu finden, die sowohl direkt ausführbar ist als auch als formale Spezifikation dient. Diese soll alle Informationen enthalten, die zur Laufzeit-Interpretation und gleichzeitig zu Modellierung und Model Checking zur Entwicklungszeit notwendig sind: Informationen über Abläufe, Variablenbelegungen sowie Vor- und Nachbedingungen sollen permanent explizit und konsistent verfügbar sein, so dass semantische Inkonsistenzen nicht entstehen können.

2 Ansatz

Da mit vorhandenen Mitteln der modellbasierten Entwicklung das Problem nicht zu beheben ist, soll versucht werden, die Beziehung zwischen Modell und Programmcode langfristig zu erhalten. Eine parallele aber ansonsten unabhängige Pflege beider Ebenen ist nicht realistisch. Daher soll eine Struktur für den in jedem Fall zur Ausführung benötigten Programmcode gefunden werden, die zur Laufzeit als Zustandsautomat interpretiert wird und gleichzeitig eine Rückkopplung ins Modell erlaubt. Die statischen Bestandteile des Modells – Zustände, Übergänge und Bedingungen, die für das Schalten einer Transition erfüllt sein müssen [PGS01, Kap. 4] – sollen dazu als Klassenstrukturen definiert und die Klassen mit Metadaten über Nachfolgezustände und Bedingungen angereichert werden. In den Transitionen wird Java-Code ausgeführt, der anwendungsspezifisch und beliebig komplex ist, so dass er als Black Box nicht betrachtet werden soll. Nicht umsetzbar ist daher die statische Prüfung von Nachbedingungen, die das Setzen und Verändern von Variablen abbilden. Diese sollen allerdings trotzdem definiert werden, so dass zur Entwicklungszeit die Validerung des Modells und zur Laufzeit eine dynamische Überprüfung zum Debugging des Modells möglich ist.

Es wird also das Ziel verfolgt, das UPPAAL-Modell so vollständig im Java-Code abzubilden, dass es semantisch interpretiert werden kann. So wird erreicht, dass Modell und Programmcode nicht getrennt gepflegt werden und der Zusammenhang erhalten bleibt. Das Modell soll aus dem Java-Code ausgelesen und in UPPAAL eingespeist werden können, um dort auf einer höheren Abstraktionsebene validiert zu werden. Entwickelt werden sollen für diesen Zweck neben der erforderlichen Spezifikation drei Werkzeuge:

- Eine Ausführungskomponente, die zur Laufzeit die Strukturen interpretiert und Zustandsübergänge entsprechend erfüllter Bedingungen aktiviert oder untergeordnete Automaten aufruft.
- Ein Java-nach-UPPAAL-Konverter, der die Klassenstrukturen und Metadaten des Java-Codes interpretiert und das UPPAAL-Modell aufbaut.
- Ein Konverter, der UPPAAL-Modelle in Java transformiert. Die Herausforderung ist hierbei die Validierung, dass nur unterstützte Funktionalität im Modell verwendet wird und somit keine Informationen verloren gehen. Diese Komponente ist optional, da sie für den Erhalt des Zusammenhangs zwischen Programmcode und Modell nicht benötigt wird. Sie dient lediglich der Vereinfachung der erstmaligen Erzeugung von Programmcode-Strukturen aus einem gegebenen Modell.

3 Umsetzung

3.1 Relevante UPPAAL-Funktionalität

UPPAAL wurde für das genannte Projekt aufgrund der Fähigkeiten zur Erstellung zeitbehafteter Modelle gewählt. Dieser Aspekt bleibt für die konzeptionelle Vorstellung in diesem Beitrag zunächst unberücksichtigt, eine spätere Ergänzung ist allerdings sinnvoll. Daher wurde darauf verzichtet, für diesen Beitrag eine einfachere Modellsprache zu verwenden. Zudem eignen sich UPPAAL-Modelle aufgrund der übersichtlichen Darstellung der ausgereiften Werkzeuge zur Dokumentation und Kommunikation von Modelle. Verwendet wird in diesem Ansatz zunächst die Grundfunktionalität von UPPAAL, also die Definition von Automaten mit Zuständen und Transitionen [LPY97, Kap. 4.1] sowie die Verwaltung von Variablen. Letztere werden in UPPAAL für einen Automaten global definiert. Auf die gleiche Art stehen dabei global definierte Funktionen zum Aufruf an den Transitionen zur Verfügung, die komplexere Variablenänderungen bewirken. Das Senden von Signalen über Kanäle bleibt ebenfalls zunächst unberücksichtigt.

Aus der Vernachlässigung zeitbehafteter Modelle ergibt sich implizit die weitere Einschränkung, dass die Aufenthaltsdauer in einem Zustand 0 ist, so dass innerhalb eines Zustandes keine Veränderung von Variablen vorgenommen werden kann. Eine Prüfung der Invarianten eines Zustandes ist in diesem vereinfachten Modell daher hinfällig. Im Model Checking im engeren Sinne, in dem der Zustandsraum durch die Belegung der Variablen aufgespannt wird, ist eine Veränderung der Variablenbelegung innerhalb eines Zustandes aber ohnehin nicht möglich, da jede Änderung einer Transition und damit einem Wechsel des Zustandes entspricht, während ein einzelner Zustand stets nur einem Zeitpunkt des Systems und niemals einer Zeitspanne entspricht ([PGS01, Kap. 4.2]). Als weitere Einschränkung wird aus Gründen der Übersicht zunächst auf die Umsetzung der in UPPAAL definierbaren komplexen Datentypen verzichtet und die Vorgehensweise anhand der vordefinierten primitiven Datentypen für Wahrheitswerte, Ganzzahlen und Fließkommazahlen erläutert.

3.2 Umsetzung und Ausführung im Java-Code

Die entsprechend strukturierten UPPAAL-Modelle sollen im Java-Code so repräsentiert werden, dass der Bezug zum Modell erhalten bleibt. Erforderlich sind dafür deklarative Strukturen, hier in Form von Klassen, Methoden und Java-Annotationen [GJSB05, Kap. 9.7]. Dabei wurde Wert auf die Beibehaltung der Typsicherheit gelegt, so dass die Programmierung ohne Kompromisse für die Angabe der Spezifikation erfolgen kann. Die Strukturen sind im Einzelnen:

- Aus obiger Beschreibung der Struktur von Zustandsautomaten in UPPAAL ergibt sich, dass der Aufruf von Java-Code, der eine Zustandsänderung bewirken soll, ausschließlich über die Formulierung entsprechender Methodenaufrufe für das Schalten von Transitionen erfolgen kann. Es wird davon ausgegangen, dass die Schnittstel-

le zu diesem als Black Box betrachteten Code in einer Klasse gekapselt und eine Instanz davon – bezeichnet als **Actor** – an die Ausführungskomponente übergeben wird. Dadurch wird gleichzeitig vermieden, dass versehentlich Programmaufrufe über die Grenzen verschiedener Automaten hinweg getätigt werden, da jeder Automat einen ihm zugeordneten Actor haben kann und eine saubere Trennung gemäß der Architektur damit erzwungen wird.

- UPPAAL-**Zustände** werden als Klassen abgebildet, die das vorgegebene State-Interface implementieren. State-Klassen müssen über das Interface einen generischen Typ definieren, der dem Typ der Actor-Implementierung entspricht.
- **Transitionen** werden als Methoden innerhalb des States abgebildet, von dem sie ausgehen. Jeder Transitions-Methode wird als Parameter die Actor-Instanz übergeben, so dass im Methodenrumpf Methoden im Actor aufgerufen werden können. Über Annotationen werden die Klasse des Ziel-States sowie Mengen von Vor- und Nachbedingungen angegeben. Letztere können sowohl konstante Werte als auch Wertebereiche für bestimmte Variablen erwarten. Nachbedingungen werden zur Laufzeit nicht zur Steuerung des Ablaufs genutzt, allerdings kann die Ausführungskomponente mit den Angaben die vom Actor gesetzten Variablen nach dem Feuern der Transition validieren.
- Aus UPPAAL wird die Verwaltung der **Variablen** adaptiert: Diese werden nicht von der Ausführungskomponente verwaltet, sondern in einem separaten Variablencontainer, das für den jeweiligen Anwendungsfall spezifisch implementiert wird und die Variablen über get-Methoden zur Verfügung stellt. Für das Setzen der Variablen ist der Actor verantwortlich, da die Wertezuweisung im Programmcode erfolgen muss, der nicht interpretiert werden kann.

In der Ausführungskomponente wird die Abarbeitung des Automaten über eine Methode angestoßen, die die zu verwendenden Instanzen von Actor und Variablencontainer sowie den Startknoten als Parameter übernimmt. Diese wird in Java über folgende Signatur definiert, wobei <A> der generische Typ des Actors und <V> der generische Typ des Variablencontainers ist:

```
void <A, V> start(Class<State<A>> start, A actor, V vars)
```

Die Ausführungskomponente baut aus diesen Informationen einen Zustandsautomat mit Instanzen der State-Klassen auf. Dabei wird die Gültigkeit der Transitionsmethoden im Bezug auf die Parameter (Actor-Typ) validiert. Dann beginnt im Startzustand die Prüfung der Vorbedingungen und das Feuern der Transitionen.

4 Beispiel

Gegeben sei der Zustandsautomat aus Abbildung 1 mit fünf Zuständen. Für die Umsetzung in Java sollen der Actor als Objekt vom Typ MyActor und der Variablencontainer

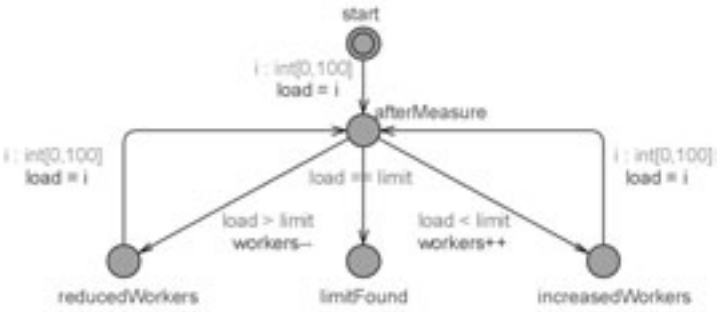


Abbildung 1: Einfacher Zustandsautomat mit fünf Zuständen.

repräsentiert durch ein Objekt vom Typ MyVariables als gegeben angenommen werden. Die States müssen demnach wie in folgendem Codeausschnitt definiert werden:

```
public class StartState implements State<MyActor> {
   @Transition(AfterMeasure.class)
   @Updates(ranges = {
      @RangeUpdate(var="load", min=0, max=100)
   })
   public void startToAfterMeasure(MyActor a) {
      a.doMeasure();
   }
}

public class AfterMeasure implements State<MyActor> {
   @Transition(ReducedWorkers.class)
   @Condition("load > limit")
   @Updates(constants = {
      @ConstantUpdate(var="workers", value="workers-1")
   })
   public void reduceWorkers(MyActor a) {
      a.reduceWorkers();
   }
   // . . .
}
```

Die im Modell eingetragene nicht-deterministische Auswahl einer Ganzzahl im Intervall von 0 bis 100 bildet die Durchführung einer Messung ab, die durch den Actor vollständig gekapselt und deren Ergebnis erst zur Laufzeit bestimmbar ist. Die Spezifikation der gültigen Spanne für Messwerte erlaubt es, das Modell sowohl statisch als auch dynamisch zu überprüfen, ohne die Implementierung der Messung zu kennen.

Der Start der Ausführungskomponente in Java entspricht folgenden Zeilen, wobei die Parameter der Konstruktoren nur beispielhaft sind:

```
MyVariables v = new MyVariables();
MyActor a = new MyActor(v);
StateMachine.start(StartState.class, a, v);
```

5 Fazit und Ausblick

Als Fazit kann festgestellt werden, dass für den gewählten Teilbereich der Funktionalität von UPPAAL alle Modell-relevanten Informationen als Klassenstrukturen und Annotationen im Programmcode abgelegt werden können. Eine Extraktion und Weiterverwendung als Modell ist jederzeit möglich, so dass dieses Ziel vollständig erreicht wurde.

Ein Großteil der Informationen wird auch verwendet, um das Modell auszuführen, so dass die Angaben obligatorisch sind und die Weiterentwicklung des Modells nicht vernachlässigt werden kann. Eine Ausnahme sind die Updates, deren Angabe freiwillig ist und vernachlässigt werden kann, solange das Modell nicht in UPPAAL verifiziert wird. Somit ist dieses Ziel größtenteils, aber nicht vollständig erreicht worden. Fehlerhafte Updates werden beim Simulieren in UPPAAL allerdings erkannt und Inkonsistenzen aufgedeckt. Da die Einstiegspunkte in den anwendungsspezifischen Programmcode bekannt sind, können die fehlenden Informationen durch Analyse dieses im Ansatz als Black Box definierten Codes mit geeigneten Methoden extrahiert werden.

Die gewonnenen Erkenntnisse werden verwendet, um ein Werkzeug zu entwickeln, das in Echtzeit die parallele Entwicklung an Code und Modell unterstützt und dabei trotz unterschiedlicher Repräsentationen die Synchronisierung erhält, ohne automatische Transformationen oder Manipulationen vornehmen zu müssen.

Nach der Umsetzung soll zudem die Möglichkeit der Übertragung des Ansatzes auf weitere Problemfelder, in denen Modelle im Programmcode abgelegt werden können, untersucht werden.

6 Related Work

Auf formaler Ebene kann die Konsistenzsicherung zwischen Modellen durch Triple-Graph-Grammatiken [Kö05] ausgedrückt werden. Wird der Programmcode in Form seines Syntaxbaumes als Graph aufgefasst, definieren die in Abschnitt 3.2 Umsetzungen das Mapping zwischen den beiden Graphen.

Durch die Erhaltung der permanenten Konsistenz zwischen Code und Modell lässt sich der gewählte Ansatz auch gegenüber anderen Ansätzen zu ausführbaren Spezifikationen abgrenzen. Anders als im Konzept der “executable UML” [MB02], bei dem Inkonsistenzen zwischen verschiedenen Repräsentationen durch toolbasierte automatische Transformation oder die Definition einer primären Repräsentation, aus der andere Darstellungeformen – inklusive Programmcode – generiert werden, ausgeglichen werden sollen, soll in diesem Ansatz die Vorrangstellung einer Repräsentation genau vermieden werden.

Anders als in der Java Modeling Language (JML) [LBR99] wird mit dem gewählten Ansatz nicht versucht, eine Spezifikationsnotation für alle denkbaren Systeme zu finden, sondern nur für das ausgewählte Teilproblem der Zustandsautomaten. Dadurch kann für diesen Teilbereich weitgehende Vollständigkeit erreicht werden.

Literatur

[BSGT03] Reinhard Bordewisch, Bärbel Schwärmer, Michael Goedicke und Peter Tröpfner. Lastsimulation für Anwendungsumgebungen in vernetzten IT-Architekturen. *Mitteilungen der GI-Fachgruppe MMB*, (43), 2003.

[GJSB05] James Gosling, Bill Joy, Guy Steele und Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[Kö05] Alexander Königs. Model Transformation with Triple Graph Grammars. In *Model Transformations in Practice Satellite Workshop of MODELS 2005*, Montego Bay, Jamaica, 2005.

[LBR99] Gary T. Leavens, Albert L. Baker und Clyde Ruby. JML: A Notation for Detailed Design. In Haim Kilov, Bernhard Rumpe und Ian Simmonds, Hrsg., *BehavioralSpecifications of Businesses and Systems*, Seiten 175–188. Kluwer, 1999.

[LPY97] Kim G. Larsen, Paul Pettersson und Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, Oct 1997.

[MB02] Stephan J. Mellor und Marc J. Balcer. *Executable UML*. Addison-Wesley, 2002.

[PGS01] Doron A. Peled, David Gries und Fred B. Schneider, Hrsg. *Software reliability methods*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[SK04] Shane Sendall und Jochen Küster. Taming Model Round-Trip Engineering. In *Proceedings of Workshop on Best Practices for Model-Driven Software Development*, 2004.

[TG03] Matthias Tichy und Holger Giese. Seamless UML Support for Service-based Software Architectures. In N Guefi, E Artesiano und G Reggio, Hrsg., *Proceedings of the International Workshop on scientiFic engIneering of Distributed Java applIcations (FIDJI) 2003, Luxembourg*, Jgg. 2952 of *Lecture Notes in Computer Science*, Seiten 128–138. Springer-Verlag, November 2003.

Modellgetriebene Integration von Altsystemen

Dieter Hildebrandt[1] Michael Gründler[2] Heiner Feislachen[2]

[1]Betriebliches Informationsmanagement, OFFIS Institut für Informatik
Escherweg 2
D-26121 Oldenburg
dieter.hildebrandt@offis.de

[2]Softwarelösungen, BTC AG
Escherweg 5
D-26121 Oldenburg
michael.gruendler@btc-ag.com, heiner.feislachen@btc-ag.com

Abstract: Modellgetriebene Entwicklungsverfahren zur Integration bestehender heterogener betrieblicher Informationssysteme können insbesondere für KMU eine wesentliche Unterstützung darstellen, da sie den flexiblen Umgang mit sich rasch ändernden Anforderungen vor dem Hintergrund gewachsener Anwendungslandschaften erlauben.

In dem vorliegenden Beitrag wird ein Vorgehen zur modellgetriebenen Integration von Altsystemen beschrieben. Bei diesem Vorgehen werden Altsysteme bereits auf der Ebene von Geschäftsprozessen integriert, wobei die Geschäftsprozesse mit einer für diesen Zweck geschaffenen domänenspezifischen Sprache modelliert werden. Neben der Darstellung des Vorgehens liegt der Fokus dieses Beitrages auf der Beschreibung von Aspekten der Modellierung eines Altsystems und der Art und Weise der Verwendung von Altsystemmodellen bei der modellgetriebenen Integration von Altsystemen.

1 Einleitung

Das vom BMBF geförderte Projekt *Modellgetriebene Integration von Informationssystemen* (MINT) [SSMG06] befasst sich unter anderem mit der Entwicklung eines modellgetriebenen Verfahrens für die Integration bestehender Systeme, im Folgenden auch als Altsysteme bezeichnet. Der Fokus des Projekts liegt dabei auf betrieblichen Informationssystemen für kleine und mittlere Unternehmen (KMU). Das entwickelte Verfahren soll KMU die Möglichkeit geben, ihre Softwaresysteme kosteneffizient an sich ändernde Geschäftsprozesse und neue Anforderungen anpassen zu können. Ein Schwerpunkt des Projektes ist die Vereinfachung der Beschreibung von Softwaresystemen unter Einbindung von Fachexperten durch den Einsatz domänenspezifischer Sprachen (Domain Specific Languages, DSL [MHS05]).

Der hier vorgestellte Ansatz grenzt sich zu anderen Ansätzen der modellgetriebenen Entwicklung durch die Integration bestehender Systeme in den Entwicklungsprozess ab. Dieser Aspekt wird weder in der von OMG vorgeschlagenen *Model Driven Architecture*

(MDA) [MM03], noch in den bisherigen praxisorientierten Ansätzen zur modellgetriebenen Entwicklung, wie z. B. [SV05], berücksichtigt. Zudem integriert der vorgestellte Ansatz bereits auf der Ebene von Geschäftsprozessen, und nicht wie häufig anzutreffen auf der Ebene von Datenmodellen oder Anwendungen.

In dem Projekt MINT wird u. a. der Frage nachgegangen, ob sich die üblicherweise mit der modellgetriebenen Softwareentwicklung verbundenen Vorteile (vgl. z. B. [SV05]) auch auf Integrationsvorhaben übertragen lassen, und wie eine modellgetriebene Integration von Altsystemen konkret umgesetzt werden kann. Der Fokus dieses Beitrages liegt auf der Beschreibung von Aspekten der Modellierung von Altsystemen für den Zweck der modellgetriebenen Integration (Abschnitt 2) und der Darstellung eines Vorgehens zur modellgetriebenen Integration (Abschnitt 3). In Abschnitt 4 wird kurz auf die verwendete technische Plattform eingegangen. Der Beitrag schließt mit einer Zusammenfassung in Abschnitt 5.

2 Modellierung von Altsystemen

Viele der älteren, in Unternehmen häufig noch im Einsatz befindlichen Anwendungssysteme sind monolithisch aufgebaut. Die Herausforderung der Integration neuer Software mit diesen Anwendungssystemen besteht darin, dass die dort bereits implementierten Fachverfahren sowie der mitunter über Jahrzehnte aufgebaute Datenbestand in den neu zu realisierenden Geschäftsprozessen aufgehen müssen. Da in der Regel für die bestehenden Anwendungen keine Modelle existieren, müssen ihre Daten, Verfahren und mögliche Schnittstellen zunächst identifiziert und modelliert werden.

Informationen über bestehende Dienste der zu integrierenden Anwendungssysteme sind aus unterschiedlichen Quellen verfügbar:

- Technische und fachliche Dokumentation
- Anwender und Entwickler
- Quelltexte und Datenbanken
- Repositories, wie z. B. UDDI (Universal Description, Discovery and Integration)
- Explizite Beschreibungen von Funktionalitäten, z. B. in der Form von WSDL (Web Services Description Language)

Im Kontext des Projektes MINT ist es notwendig, dass die gesammelten Informationen über die bestehenden Anwendungssysteme bzw. Altsysteme in einer Sprache vorgehalten werden, die sowohl von Softwarearchitekten, als auch von den im weiteren Integrationsprozess verwendeten Generatoren interpretierbar ist. Zur Modellierung des Altsystems wurde als Modellierungssprache die UML2-Notation [Obj07] gewählt. Eine Evaluation hat ergeben, dass die UML die an eine Modellierungssprache von Altsystemen im Kontext des Projektes gestellten Anforderungen am besten erfüllt:

- Angemessenheit: Die Modellierungssprache muss sinnvoll zur Modellierung von Altsystemen anwendbar sein und alle relevanten Eigenschaften dieser Systeme modellieren können.
- Verbreitung: Die Modellierungssprache muss sowohl in aktuell vorhandenen Werkzeugen unterstützt als auch bei Softwarearchitekten verbreitet sein.
- Gute Werkzeugunterstützung: Die Modellierungssprache muss durch bestehende Werkzeuge gut unterstützt werden. In der UML repräsentierte Modelle können beispielsweise im Generatorframework *openArchitectureWare* (oAW) [ope] bei den Modell-zu-Modell-Transformationen direkt verarbeitet werden.

Modelle von Altsystemen bestehen in erster Linie aus Komponentendiagrammen. Diese Komponentendiagramme modellieren eine Zerlegung der zu integrierenden, aus Altsystemen bestehenden Anwendungslandschaft in Komponenten und Schnittstellen dieser Komponenten. Die Modellierung beschränkt sich dabei allerdings auf die für das jeweilige Ziel der Integration notwendige, öffentliche Funktionalität der Altsysteme. Weitere relevante Informationen, die nicht über die vorgegebenen Sprachmittel der UML abgebildet werden können, werden über ein für diesen Zweck erstelltes UML-Profil mit Stereotypen und *tagged values* aufgenommen.

In der derzeitigen, ersten Realisierung des Vorgehens sind Altsystemmodelle manuell durch einen Softwarearchitekten zu erstellen. Diese Vorgehensweise ist aufwendig aber für weniger komplexe Systeme vertretbar, da nur die für die jeweilige Integration notwendigen, öffentlichen Funktionalitäten modelliert werden müssen. Das erstellte Altsystemmodell kann dabei in aufeinanderfolgenden Integrationsprojekten wiederverwendet und inkrementell angepasst und erweitert werden. Es ist geplant zu einem späteren Zeitpunkt die Altsystemmodelle automatisiert durch geeignete Reengineering Methoden zu generieren. Wenn die Altsysteme beispielsweise im günstigsten Fall bereits Funktionalität in der Form von Web Services anbieten, ist eine automatisierte Erzeugung eines UML-Altsystemmodells aus WSDL-Beschreibungen mittels einer WSDL-zu-UML-Transformation möglich.

3 Das MINT-Vorgehensmodell zur Integration von Altsystemen

Das MINT-Vorgehensmodell zur modellgetriebenen Integration von Altsystemen basiert auf der MDA von OMG [MM03] und unterstützt alle vier dort vorgeschlagenen Ebenen: das *Computation Independent Model* (CIM), das *Platform Independent Model* (PIM), das *Platform Specific Model* (PSM) und die Code-Ebene. Abbildung 3.1 zeigt das MINT-Vorgehensmodell des Integrationsprozesses im Überblick.

Das im vorherigen Abschnitt dargestellte Altsystemmodell ist sowohl der CIM-, PIM- als auch PSM-Ebene zuzuordnen. Diese Zuordnung reflektiert, dass (a) das Altsystemmodell primär geschäftsbezogene Funktionalität modelliert, die als Teil von Gechäftsprozessen verwendet wird (CIM), (b) bestimmte Aspekte der Architektur der Altsysteme modelliert (PIM) und (c) zusätzlich plattformspezifische Angaben enthält (PSM), wie beispielsweise

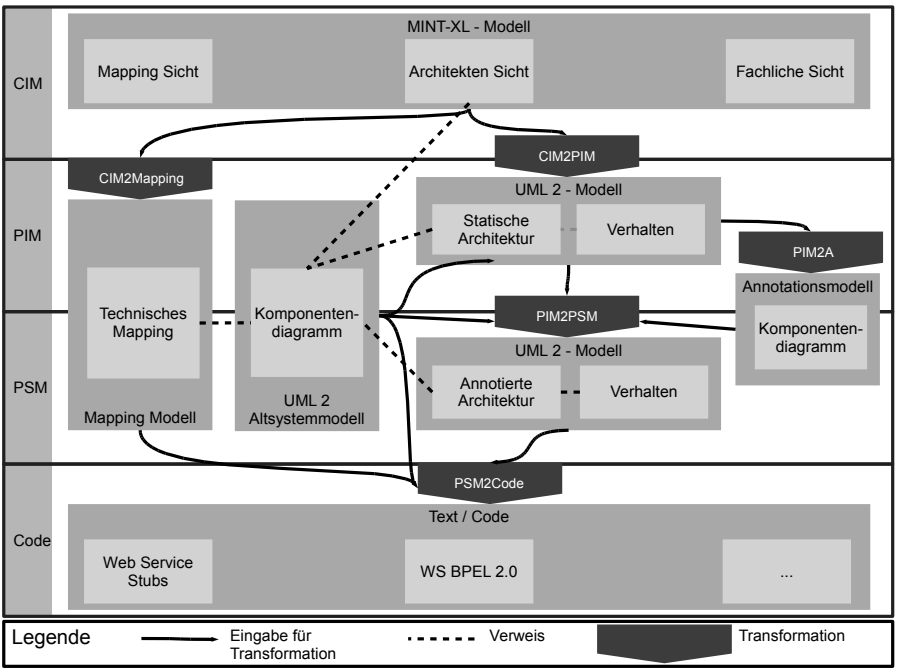


Abbildung 3.1: Das MINT-Vorgehensmodell zur Integration von Altsystemen

Hostinformationen von Web Services. Das Vorgehensmodell setzt voraus, dass die Funktionalitäten der Altsysteme durch Web Services gekapselt sind. Für Altsysteme, deren Funktionalität noch nicht auf diese Weise gekapselt ist, werden durch das Vorgehen im weiteren Verlauf automatisch Web Service Stubs bzw. Adapter generiert (s. u.).

Auf der CIM-Ebene sind unter Verwendung der im Projektverlauf entwickelten domänenspezifischen Sprache *MINT-XL* drei Modelle zu erstellen. Den Ausgangspunkt stellt die von einem Fachexperten zu erstellende *fachliche Sicht* dar. Sie definiert einen Geschäftsprozess, der das Zusammenspiel der zu integrierenden Altsysteme beschreibt. Die *Architekten Sicht* wird von einem Softwarearchitekten erstellt und besteht aus einer Menge von Annotationen, die sich auf die fachliche Sicht beziehen. Durch diese Annotationen können u. a. Beziehungen zwischen dem Altsystemmodell und der fachlichen Sicht hergestellt werden, indem angegeben wird, durch welche Elemente des Altsystems einzelne Prozessschritte realisiert werden. Neben den Prozessmodellen ist von einem Softwarearchitekten eine *Mapping Sicht* zu erstellen, die definiert, wie Daten beim Austausch zwischen den beteiligten Altsystemen zu transformieren sind. Mit der Mapping Sicht wird dem Umstand Rechnung getragen, dass bestehende Altsysteme häufig unterschiedliche Datenmodelle verwenden und auch die Durchsetzung eines unternehmensweit einheitlichen Datenmodells [Sch88] als nicht realistisch gilt [KBS04]. Die Erstellung der drei genannten Modelle wird durch drei spezialisierte, domänenspezifische *MINT-XL Editoren* unterstützt.

Bei der Transformation des CIM in ein PIM wird die fachliche Sicht, die Architekten

Sicht, die Mapping Sicht und das Altsystemmodell verarbeitet und ein PIM in der UML2 Notation erzeugt. Das PIM beschreibt sowohl *Statische Struktur* in der Form eines Komponentendiagramms, als auch *Verhalten* in der Form eines Aktivitätsdiagramms des durch die Integration erzeugten Systems. Dieses System besteht aus den Altsystemen, die gemäß des auf der CIM-Ebene spezifizierten Geschäftsprozesses verknüpft worden sind.

Für die Transformation des PIM in ein PSM sind neben dem PIM zusätzliche Informationen bzgl. der gewünschten Plattform notwendig, auf der der integrierte Geschäftsprozess später ausgeführt werden soll. Diese Informationen werden, wie bereits analog auf der CIM-Ebene, in Form von Annotationen hinterlegt, die in diesem Fall das PIM mit Informationen anreichern. Um zu verhindern, dass Annotationen an einem generierten PIM bei einer erneuten Generierung überschrieben werden, werden diese Informationen in dem separaten *Annotationsmodell* abgelegt. In die Transformation des PIM in das PSM geht neben dem PIM und dem Annotationsmodell auch das Altsystemmodell ein.

Bei der letzten Transformation geht neben dem PSM auch ein *technisches Mapping* genanntes Modell ein. Dieses ergänzt die bereits eingeführte Mapping Sicht, in der im Wesentlichen lediglich deklarativ Zuordnungen von Elementen unterschiedlicher Datenmodelle zueinander vorgenommen werden, um detaillierte Spezifikationen, wie die Abbildungen algorithmisch umgesetzt werden sollen. Die Transformation von PSM und technisches Mapping erzeugt ausführbaren Code in unterschiedlichen Zielsprachen. Der auf der CIM-Ebene modellierte Geschäftsprozess und Teile des technischen Mappings werden in die Business Process Execution Language (BPEL, [OAS]) transformiert. Die erzeugten BPEL-Skripte können direkt in entsprechenden BPEL-Engines ausgeführt werden.

4 Technische Plattform

Technische Plattform des MINT-Integrationsprozesses ist die Open Source Entwicklungsumgebung *Eclipse* [Ecla]. Eclipse erlaubt das Laden unterschiedlicher Plugins, die u. a. das Erstellen von domänenspezifischen Sprachen, Editoren und Generatoren ermöglichen. Bei einer zum Projektstart durchgeführten Evaluation von unterschiedlichen MDA-Werkzeugen (*openArchitectureWare, AndroMDA, Software Factories*) stellte sich heraus, dass das Framework *openArchitectureWare* (oAW) [ope] die im Projekt MINT bestehenden Anforderungen am besten erfüllt [FM07]. Ein wesentliches Kriterium war neben der Unterstützung von Modell-zu-Code-Transformationen die Unterstützung von Modell-zu-Modell-Transformationen, die einzig von oAW in ausreichendem Maße geboten wird. Aus diesem Grund bietet oAW als einziges der untersuchten Werkzeuge ausreichend Unterstützung für alle vier Ebenen der MDA. In oAW kann zur Erstellung des Meta-Metamodells das *Eclipse Modeling Framework* (EMF) [Eclb] verwendet werden. EMF nutzt zur Modellierung *Ecore*, das weitestgehend kompatibel zu *EMOF* [Obj06] ist.

5 Zusammenfassung

In diesem Beitrag wurden Aspekte der Modellierung von Altsystemen zur Verwendung in der modellgetriebenen Integration dargestellt. Zudem wurde ein Vorgehen zur modellgetriebenen Integration vorgeschlagen, das Altsysteme bereits auf der Ebene von Geschäftsprozessen integriert. Der beschriebene Ansatz basiert dabei auf der von der OMG vorgeschlagenen MDA. Bei dem Vorgehen sind neben den Geschäftsprozessmodellen das Altsystemmodell und die Mappings zwischen den Datenmodellen der Altsysteme wichtige Eingaben und Ausgangspunkte für eine Kette von Transformationen, an deren Ende ein ausführbares, integriertes Teilsystem steht.

Das Projekt befindet sich derzeit noch in der Durchführung. Zukünftige Arbeiten richten sich insbesondere auf die Generierung von Adaptern für Altsysteme und die automatisierte Erzeugung des Altsystemmodells durch Methoden des Reengineerings. Evaluationsergebnisse des Ansatzes liegen derzeit noch nicht vor.

Literatur

[Ecla] Eclipse Foundation. Eclipse. `http://www.eclipse.org`.

[Eclb] Eclipse Foundation. Eclipse Modeling Framework. `http://www.eclipse.org/emf/`.

[FM07] Heiner Feislachen und Jürgen Meister. Evaluation von MDA-Werkzeugen. Bericht, BTC-AG, 2007. `http://bi.offis.de/publicProjectDocument/Bericht_Evaluation_MDA-Werkzeugauswahl.pdf?docId=nbqy8mgsvk42\&groupId=awb0lzr338sz`.

[KBS04] Dirk Krafzig, Karl Banke und Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[MHS05] Marjan Mernik, Jan Heering und Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[MM03] Joaquin Miller und Jishnu Mukerji. *MDA Guide Version 1.0.1*. Object Management Group (OMG), June 2003.

[OAS] OASIS. *BPEL*. OASIS. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[Obj06] Object Management Group (OMG). *Meta Object Facility (MOF) Core Specification Version 2.0*. Object Management Group (OMG), January 2006.

[Obj07] Object Management Group (OMG). Unified Modeling Language: Superstructure Version 2.1.1 (formal/2007-02-05), 2007.

[ope] openArchitectureWare.org. openArchitectureWare. `http://www.openarchitectureware.org`.

[Sch88] August-Wilhelm Scheer. *Wirtschaftsinformatik*. Springer Verlag, Berlin, Heidelberg, New York, 1988.

[SSMG06] Niels Streekmann, Ulrike Steffens, Claus Möbus und Hilke Garbe. Model-Driven Integration of Business Information Systems. *Softwaretechnik-Trends*, 26(4):9–13, November 2006.

[SV05] Thomas Stahl und Markus Völter. *Modellgetriebene Softwareentwicklung*. Dpunkt Verlag, 2005.

Inkrementelle Transformation einer monolithischen Geschäftssoftware

Sascha Hunold[1], Matthias Korch[1], Björn Krellner[2],
Thomas Rauber[1], Thomas Reichel[2], Gudula Rünger[2]

[1] Lehrstuhl für Angewandte Informatik II, Universität Bayreuth
Universitätsstraße 30, 95447 Bayreuth
hunold, matthias.korch, rauber@uni-bayreuth.de

[2] Professur Praktische Informatik, Technische Universität Chemnitz
Straße der Nationen 62, 09111 Chemnitz
bjoern.krellner, thomas.reichel, ruenger@informatik.tu-chemnitz.de

Abstract: Pflege und Wartung von Altsoftware (*legacy software*) sowie die Erweiterung und Anpassung dieser Applikationen an neue Märkte machen einen immensen Kostenanteil von vielen Softwareunternehmen aus. Die manuelle Migration von Software ist durch zunehmend kürzer werdende Abstände der Entwicklungsstadien (release cycles) nur durch immensen personellen Aufwand zu leisten. Die Entwicklung von Werkzeugen und Verfahren zur Unterstützung dieses Migrationsprozesses ist deshalb für die Unternehmen von entscheidender Bedeutung. In diesem Artikel erarbeiten wir einen inkrementellen Transformationsprozess für Geschäftssoftware, der unter Zuhilfenahme von Verfahren des Compilerbaus und des Software-Engineerings eine Kette von Transformationsschritten definiert, mit deren Hilfe eine monolithisch strukturierte Altsoftware in eine modulare Client-Server-Applikation überführt werden kann.

1 Einleitung

Softwaresysteme zur Realisierung von Geschäftsprozessen werden in vielen Unternehmen eingesetzt und tragen oft entscheidend zum Erfolg des Unternehmens bei. Die schnelle Entwicklung der Hardware- und Software-Technologien, aber auch Veränderungen innerhalb von Unternehmen, z. B. die Anpassung an neue Geschäftsfelder, erfordern jedoch eine stetige Wartung, Anpassung und Erweiterung der Softwaresysteme. Ist die Pflege einer Software mit vertretbarem Aufwand nicht mehr möglich, spricht man von *Altsoftware* [Ulr02].

An eine moderne Geschäftssoftware werden Anforderungen gestellt, die eine solche monolithische Architektur nicht erfüllen kann. Diese sollte auf einer Vielzahl von Hardware- und Software-Plattformen einsetzbar sein. Sie sollte modular aufgebaut und flexibel konfigurierbar sein, um an spezifische Kundenanforderungen und neue Geschäftsstrategien angepasst werden zu können. Eine schichtenbasierte, in Objekte und Module gegliederte Softwarearchitektur ermöglicht darüber hinaus den Austausch von Programmteilen und erleichtert somit eine spätere Migration zu anderen Plattformen und Technologien. Die

Anwender sollten unterschiedliche Thin- und Rich-Clients nutzen können; insbesondere sollte die Nutzung einer Web-Schnittstelle möglich sein. Die Geschäftslogik sollte von den Clients abgekoppelt und in Form von verteilten Diensten über ein Netzwerk nutzbar sein.

Der Herausforderung der Umstrukturierung ihrer Software stellen sich viele Firmen, die ihre Softwareprodukte Ende der 80er oder Anfang der 90er Jahre im Hinblick auf den damaligen Standard der Modularität und Flexibilität entworfen haben, aber die den heutigen technologischen Anforderungen nicht mehr genügen [CMER+06]. Zu einem großen Teil wurden diese Anwendungen in einer objektorientierten Programmiersprache implementiert. Zwar besitzen sie eine interne Aufteilung in Unterprogramme und Klassen sowie Gruppierungen dieser Programmelementen zu Quelltextpaketen, es fehlt jedoch eine klare Aufteilung in Schichten und eigenständige Module. Vielmehr sind die Programmelemente so eng miteinander verwoben, dass man von einer monolithischen Struktur spricht. Oftmals sind diese Business-Applikationen dadurch gekennzeichnet, dass mehrere Benutzer über ihre grafische Benutzerschnittstelle (engl.: *graphical user interface*, GUI) Daten auf einem gemeinsamen Datenbankserver lesen oder modifizieren.

Das Ziel des Projekts TransBS[1] ist es, erprobte monolithische Unternehmenssoftware zu modularisieren und diese Module in einen verteilten Programmkontext einzubetten. Im Rahmen des Projekts betrachten wir vor allem Geschäftssoftware, die in der Programmiersprache Delphi implementiert wurde. Das hat die Konsequenz, dass sich der logische Aufbau der Software stark am Rapid-Application-Development-(RAD)-Ansatz von Borland Delphi orientiert. Da eine Softwaretransformation facettenreich auf verschiedenen Abstraktionsschichten arbeitet (Codeebene, Modellebene) [KWC98] wird innerhalb des Projekts versucht, verwandte Verfahren und Werkzeuge in den einzelnen Teilschritten wiederzuverwenden. Der inkrementelle Transformationsprozess gliedert sich in drei Schritte: (1) Generierung des Softwaremodells aus der Ursprungssoftware, (2) Modifizierung des Modells mittels Model-Driven-Architecture-(MDA)-Ansätzen [VS06] und (3) Erzeugen der Zielsoftware.

2 Inkrementeller Transformationsprozess

Wie bereits beschrieben, ist die Transformation von Geschäftssoftware eine vielschichtige Problemstellung, für die sich folgende wichtige Aspekte ergeben:

- Trennung von Nutzerschnittstelle, Geschäftslogik und Daten (Model-View-Controller-(MVC)- oder Presentation-Abstraction-Control-(PAC)-Architektur),
- Identifikation von Geschäftsobjekten und Geschäftsprozessen der Geschäftslogik,
- Modularisierung und Umsetzung in Dienste und Workflows (explizite Modellierung der Geschäftsprozesse auf der Basis von Workflows),
- Wiederverwendung des Codes der Altsoftware und Integration in das Zielsystem (z. B. zur Anbindung unterschiedlicher Thin- und Rich-Clients),
- Validierung der Zielsoftware und

[1] `http://www.transbs.de`

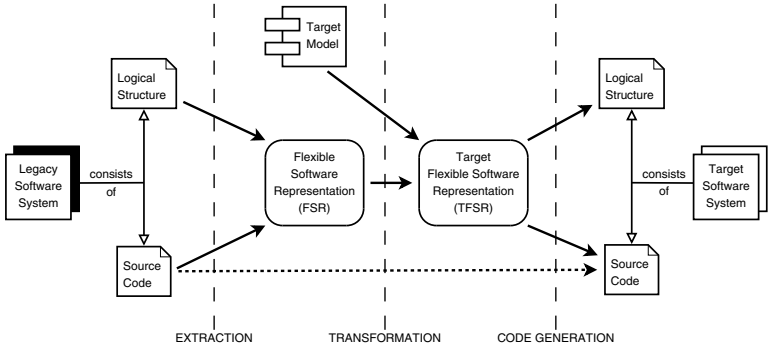


Abbildung 1: Ablauf der Transformation der Altsoftware mit `TransFormr`.

- Nutzung von Metainformationen über das Altsystem (Laufzeitverhalten, fehleranfällige Funktionen).

Zur Lösung der genannten Problemstellungen schlagen wir einen inkrementellen Transformationsprozess für die Transformation eines Altsystems in ein verteiltes, workflowbasiertes System vor. Unser Transformationssystem bietet ein Toolset zur interaktiven Unterstützung des Reengineeringprozesses. Wie in Abbildung 1 dargestellt, werden im ersten Schritt (Extraktion) des Prozesses Informationen über Quellcode und Struktur des Altsystems extrahiert und analysiert. Das Ergebnis dieses Schritts ist die Erstellung der flexiblen Sofwarerepräsentation (FSR) [RR07]. Die FSR ist eine mit Meta-Informationen angereicherte, sprachunabhängige Darstellung des Sourcecodes der Altsoftware. Sprachunabhängige Transformationen der Applikation können im nächsten Schritt (Transformation) auf die FSR angewendet werden. Im abschließenden Schritt (Generierung) wird aus dem transformierten Softwaremodell der Zielcode des Systems erzeugt.

Im Analyseschritt (Extraktion) können verschiedene Verfahren zum Reverse-Engineering zum Einsatz kommen [MJS+00], z. B. das Identifizieren von Geschäftsobjekten durch das Verfolgen des Datenwegs im Programm bis hin zu Formularfeldern. Ein wichtiges Ziel ist die Extraktion des Workflows, wobei auch die Identifikation von Business-Logik und die Gewinnung der Softwarearchitektur eine wesentliche Rolle spielt [ZH06]. Die Korrektheit der Transformation ist für eine erfolgreiche Migration von großer Bedeutung. Da die FSR eine Baumdarstellung der Software enthält, müssen u. a. Verfahren zur Validierung der Graphtransformationen berücksichtigt werden [GGZ+05].

Im Rahmen unseres Forschungsprojektes wird das Werkzeug `TransFormr` entwickelt, mit dem die Transformation einer monolithischen Applikation in den drei definierten Schritten möglich ist. Die folgenden Absätze beschreiben die Implementierungsdetails der einzelnen Schritte.

Extraktion Zur Analyse der Altsoftware nutzen wir eine Grammatik der Ausgangssprache und verschiedene Extraktionsregeln, um mit Hilfe von TXL [CCH05] die FSR zu erzeugen. Die FSR enthält die Struktur des Systems (package, module, etc.) und Beziehungen (Generalisierungen, Realisierungen, Methodenaufrufe, etc.) innerhalb des Systems

mit ihren jeweiligen Eigenschaften und Referenzen zum Quellcode. Aus der FSR können UML-Klassendiagramme, Paketabhängigkeitsdiagramme und Call-Dependency-Diagramme für die gesamte Software, für Packages oder einzelne Klassen erzeugt werden, die bei der Einschätzung und Bewertung von Transformationen hilfreich sein können.

Transformation Für eine Systemtransformation ist es nötig, ein Zielmodell (target model) zu definieren, welches Metainformationen über die Anforderungen an das Zielsystem und dessen Eigenschaften enthält. Diese Informationen über das Zielsystem müssen vom Nutzer in einzelne Transformationsschritte zerlegt werden, die mit `TransFormr` angewendet werden können. Notwendige Transformationsschritte sind das Verschieben von Funktionalität in separate Subsysteme (oder auch Dienste) und das Erzeugen von Client- und Serverklassen und verteilter Objekte, welche die Funktionalitäten der Kommunikationspartner kapseln. Zur Durchführung solch komplexer Transformationen definieren wir eine Reihe von Basistransformationen (Refactorings, z. B. `rename`, `move`, `split`) aus denen sich ein vollständiger Reengineeringprozess zusammensetzt. Die Transformationen der FSR und ihre Abhängigkeiten können mit `TransFormr` visualisiert werden. Eine Auswahl der aktuell verfügbaren Refactorings innerhalb der Transformationskomponente von `TransFormr` ist in Abbildung 2 dargestellt.

Abbildung 2: Refactoring mit `TransFormr`

Generierung Die Ziel-FSR ist durch eine Reihe von Transformationen der FSR definiert. Der Zielcode kann durch die Anwendung der Transformationen auf Codeebene erzeugt werden, wobei Teile des Originalsourcecodes wiederverwendet werden können. Die aktuelle Version von `TransFormr` unterstützt die Erzeugung von Quellcode der Zielsoftware in derselben Programmiersprache wie die Altsoftware. Für die definierten Basistransformationen sind sprachabhängige Regeln definiert, die diese umsetzen. Der erzeugte Quellcode kann entsprechend in Client und Server oder andere definierte Subsysteme/Dienste aufgeteilt sein.

3 Konfigurationsvarianten zur Generierung des Zielsystems

Die umfangreichen Steuerungsmöglichkeiten der Schritte des inkrementellen Transformationsprozesses ermöglichen eine flexible Konfiguration des Zielsystems. Das Zielsystem kann sich aus Subsystemen der Altsoftware, neu erzeugtem Programmcode, abstrakten, modellbasierten Beschreibungen von Teilen der Software (z. B. explizite Workflows oder Teile der Nutzerschnittstelle) sowie einem statischen Framework, das die Ausführung der übrigen Komponenten durch die Bereitstellung einer Laufzeitumgebung unterstützt, zusammensetzen. Eine mögliche Zielarchitektur wäre die Verwendung eines Java-Enterprise-

Servers[2] (Java EE). Datenbankobjekte müssten dafür von `TransFormr` in Entity-Beans transformiert und erforderliche Schnittstellen können dann als Web-Services für andere Systeme zugänglich gemacht werden.

Mit entsprechenden Transformationsregeln ist es auch möglich, Code zur Anbindung verschiedener Datenbanksysteme sowie die Bereitstellung und Anbindung von Diensten über unterschiedliche Middleware-Lösungen und Protokolle (z. B. CORBA, RMI, SOAP) zu generieren, wobei auch spezifische Erweiterungen des Applikationsservers berücksichtigt werden können. Für die Integration von Subsystemen der Altsoftware sind Wrapping-Ansätze geeignet, die z. B. das Subsystem in einem verteilten Objekt kapseln und auf diese Weise dessen Funktionalität im Netzwerk zur Verfügung stellen. Außerdem lassen sich damit Codeblöcke aus der Legacy-Anwendung in die neue Software-Plattform integrieren, auch ohne eine komplette Sprachmigration. Muss neuer Programmcode (z. B. aus modellbasierten Zwischendarstellungen) erzeugt werden, können mit spezifischen Codegeneratoren verschiedene Programmiersprachen unterstützt werden. Ein Ändern der Programmiersprache kann eine besseren Plattformunabhängigkeit oder Wartbarkeit ermöglichen.

Oft ist es notwendig, dass verschiedene Teile der Alt- und Neusoftware koexistieren müssen. Als Kommunikationsprotokolle zur Vermittlung zwischen Subsystemen der Altsoftware und neu erzeugten Teilen des Zielsystems (auch in anderer Programmiersprache) kommen u. a. CORBA und SOAP in Frage. Der CORBA-Standard beinhaltet jedoch für einige relevante Programmiersprachen, u. a. auch Delphi, keine standardisierte Abbildung der Schnittstellenbeschreibungssprache (IDL). Die aktuellen Delphi-Entwicklungsumgebungen bieten aber eine umfangreiche Unterstützung für SOAP. Aufgrund des einfachen, XML-basierten Protokolls können SOAP-Implementierungen bei Bedarf auch für bisher nicht unterstützte Programmiersprachen mit überschaubarem Aufwand erstellt werden.

Für die Realisierung von Client-Anwendungen können durch das Transformationssystem unterschiedliche GUI-Frameworks unterstützt werden (z. B. Delphi VCL, Java Swing). Rich-Clients vereinen Präsentationsschicht und Steuerungslogik innerhalb eines einzelnen Anwendungsprogramms; Netzwerkkommunikation führen sie nur zum Zugriff auf Dienste der Geschäftsschicht durch. Thin-Clients, wie z. B. Web-Anwendungen, erfordern Netzwerkkommunikation auch zwischen Präsentationsschicht und Controller. Speziell für die Realisierung von Web-Anwendungen eignen sich hierfür die unter dem Begriff „AJAX“ (*Asynchronous JavaScript and XML*) zusammengefassten Technologien und Protokolle.

Die Verwendung expliziter Workflows im Transformationsergebnis kann oftmals viele Vorteile mit sich bringen. Zum einen müssen Geschäftsprozesse explizit definiert werden, und zum anderen wird durch den Einsatz einer Workflow-Engine (z. B. jBPM, Enhydra Shark, WfmOpen) die Veränderung und das Hinzufügen einer Prozesskette deutlich erleichtert. Auch die Reduzierung der möglichen Fehlerquellen im Programmcode ist ein nicht zu unterschätzender Vorteil. Als Beschreibungssprache für Geschäftsprozesse bietet sich die von der Workflow Management Coalition (WfMC) standardisierte Sprache XPDL an. Zusammengesetzte Dienste, die sich mit Hilfe eines Workflows modellieren lassen, können mit einer Workflow-Beschreibungssprache (im Fall von Web-Diensten z. B. BPEL) beschrieben und in einer Workflow-Engine (z. B. jBPM) ausgeführt werden.

[2] `http://java.sun.com/javaee/`

4 Zusammenfassung

In dieser Arbeit wurde eine Methode zur Transformation monolithischer Geschäftssoftwaresysteme in eine modulare Softwarearchitektur entwickelt. Es ist eines der Ziele, dass die entstandenen Module in einen verteilten Kontext eingebettet werden können. Der zugehörige Transformationsprozess arbeitet in drei Schritten: Extraktion, Transformation und Generierung. Eine Anpassung der Schnittstellen unseres Transformationsvorschlags an die ADM-Standardspezifikation [ADM07] der OMG wird zurzeit diskutiert. Die Anforderungen des Unternehmens und dessen Kunden sowie die Eigenschaften der Altsoftware legen die Zielarchitektur fest. Aus diesem Grund unterstützt das Transformationssystem verschiedene Konfigurationsmöglichkeiten zur Auswahl der Zielarchitektur (Workflow-Extraktion, Abbildung in verteilte Komponenten, Code-Restrukturierung in MVC-Architektur). Exemplarisch werden wir im Verlauf des Projekts verschiedene Konfigurationen für eine gegebene monolithische Delphi-Anwendung hinsichtlich ihrer Anwendbarkeit evaluieren und implementieren.

Literatur

[ADM07] ADM task force. Architecture-Driven Modernization: Transforming the Enterprise. http://adm.omg.org, 2007.

[CCH05] James R. Cordy, Ian H. Carmichael und Russell Halliday. The TXL Programming Language, Version 10.4, January 2005. `http://www.txl.ca/docs/TXL104ProgLang.pdf`.

[CMER+06] R. Correia, C. Matos, M. El-Ramly, R. Heckel, G. Koutsoukos und L. Andrade. Software Reengineering at the Architectural Level: Transformation of Legacy Systems. Bericht, Department of Computer Science, University of Leicester, U.K., 2006.

[GGZ+05] Lars Grunske, Leif Geiger, Albert Zündorf, Niels Van Eetvelde, Pieter Van Gorp und Daniel Varro. Using Graph Transformation for Practical Model Driven Software Engineering. In Sami Beydeda, Matthias Book und Volker Gruhn, Hrsg., *Model-driven Software Development*, Seiten 91–118. Springer, 2005.

[KWC98] R. Kazman, S. G. Woods und S. J. Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In *Proceedings of WCRE 98, (Honolulu, HI)*, Oktober 1998.

[MJS+00] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley und Kenny Wong. Reverse Engineering: A Roadmap. In *Proc. of the Conf. on The Future of Software Engineering (ICSE '00)*, Seiten 47–60. ACM Press, 2000.

[RR07] Thomas Rauber und Gudula Rünger. Transformation of Legacy Business Software into Client-Server Architectures. In *Proc. of the 9th ICEIS*, 2007.

[Ulr02] William M. Ulrich. *Legacy Systems: Transformation Strategies*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[VS06] Markus Völter und Thomas Stahl. *Model-Driven Software Development*. Wiley, 2006.

[ZH06] Ying Zou und Maokeng Hung. An Approach for Extracting Workflows from E-Commerce Applications. In *Proc. of the 14th IEEE Int. Conf. on Program Comprehension (ICPC'06)*, Seiten 127–136. IEEE Computer Society, 2006.

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation

(PIK2008)

Inhaltsverzeichnis

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation
PIK2008 Workshop-Einleitung und Überblick 235
Andreas Birk, Ludger Fiege, Klaus Schmid

Relating Product Line Context to Requirements Engineering Processes
Using Design Rationale 240
Samuel Fricker, Reinhard Stoiber

Varianten in der Automobilelektronikentwicklung Herausforderungen und
Lösungsansätze 252
Martin Becker, Matthias Weber, Thomas Wierczoch

Multiple software product lines in automotive software development 261
Uwe Beher, Günter Bönisch, Mike Heidrich

The Sales Scenario: A Model-Driven Software Product Line 273
Henrik Lochmann, Birgit Grammel

Modellierung von Variabilität in der Geschäftsanalyse – eine industrielle Fallstudie 285
Thomas von der Maßen, Andreas Wübbeke

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation

PIK2008 Workshop-Einleitung und Überblick

Andreas Birk[1], Ludger Fiege[2], Klaus Schmid[3]

[1] Software.Process.Management, Gutenbergstraße 99, 70197 Stuttgart
andreas.birk@swpm.de

[2] Siemens AG, CT SE 2, Otto-Hahn-Ring 6, 81730 München
ludger.fiege@siemens.com

[3] Universität Hildesheim, Institut für Informatik
Marienburger Platz 22, 31141 Hildesheim
schmid@sse.uni-hildesheim.de

Abstract: Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten. Dennoch bergen sie für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Der Workshop beleuchtet aktuelle Erfahrungen mit Produktlinien und bietet ein Forum für den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt steht das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten.

1 Produktlinien

Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten. Die technischen Grundlagen sind im Wesentlichen verstanden. Erfahrungen aus der Industrie belegen das Potenzial von Produktlinien: Sie ermöglichen höhere Produktivität, steigern die Qualität und verbessern die strategischen Positionen der Unternehmen.

Dennoch bergen Produktlinien für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Selbst Unternehmen, die bereits erfolgreich Produktlinienentwicklung durchführen, sehen sich im Laufe der Zeit mitunter neuen Schwierigkeiten gegenüber. Die Gründe liegen teilweise im technischen Bereich. So sind viele Technologien für den breiten Einsatz in der Praxis noch nicht genügend ausgereift und miteinander integriert. Die größten Herausforderungen stellen sich jedoch in Bezug auf die Wechselwirkungen zwischen den technischen Verfahren mit den Prozessen sowie in Bezug auf die Gestaltung des organisatorischen und geschäftlichen Kontexts der Produktlinienentwicklung.—Wie müssen die technologischen Ansätze auf diese Wechselwirkungen ausgerichtet sein? Welche Potenziale bieten neue technologische Entwicklungen in unterschiedlichen Einsatzfeldern?

Der Workshop „Produktlinien im Kontext“ [PIK08a][PIK08b] beleuchtet aktuelle Erfahrungen mit Produktlinien und unterstützt den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt steht das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten. Daneben sollen auch neue technologische Entwicklungen vorgestellt und diskutiert werden.

2 Themengebiete

Die angenommenen Workshop-Beiträge und die Diskussionen behandeln die folgenden Themengebiete:

- Die Besonderheiten von Produktlinien in verschiedenen Geschäftsdomänen (z.B. technische Software, IT-Anwendungssysteme, sicherheitskritische Anwendungen)
- Fallstudien und Untersuchungen über die technische Umsetzung von Produktlinien (z.B. Komponenten-Frameworks, MDSD, DSL, standardisierte Architekturen)
- Produktlinien und die Prozesse des Software-Lebenszyklus (z.B. Produktmanagement, Requirements-Engineering, Testen)
- Projekt- und Programmmanagement für Produktlinien
- Initiierung, Einführung und Evolution von Produktlinien sowie Kosten/Nutzen-Betrachtungen

3 Beiträge und Diskussionen

Dieses Kapitel stellt die Workshop-Beiträge kurz vor und präsentiert die vorgesehenen Diskussionsblöcke. Die Diskussionen werden nach dem Workshop zusammengefasst und auf der Workshop-Homepage veröffentlicht [PIK08a][PIK08b].

In ihrer Keynote benennt Christa Schwanninger, Senior Research Scientist Architecture bei der Siemens AG, die wesentlichen Gefahren und Erfolgsfaktoren für Produktlinien [Sc08]. Sie betont, dass die Rolle eines Software-Architekten etabliert sein muss, in der die architektonische Verantwortung für die Produktlinie gebündelt ist. Außerdem müssen Produktmanagement und Software-Entwicklung stets eng zusammen arbeiten.

Der Beitrag von Fricker und Stoiber [FS08] berichtet von einer empirischen Studie (Action Research) über die Anpassung eines Requirements-Engineering-Prozesses an das Umfeld einer Produktlinien-Entwicklung. Er zeigt, wie eine solche Anpassung systematisch erfolgen kann, und welche Kontextfaktoren bei der Prozessgestaltung beachtet werden müssen. In der Studie waren Kontextfaktoren aus den Bereichen Technologie, Business, Organisation, geografische Verteilung, und Historie relevant.

Die Untersuchung zeigt auch, wie das betreffende Projekt mit Produktlinien-Anforderungen umging: Anstelle eines umfassenden formalen Variabilitätsmodells wurden Variabilitätsaspekte teils in textuellen Anforderungen beschrieben und teils durch tabellarische Darstellungen von Varianten und ihren Feature-Ausprägungen definiert. Wissen über Feature-Abhängigkeiten blieb weitgehend implizit, wurde aber durch Reviews und Klärungen während der Requirements-Analyse systematisch im Prozess behandelt.

Die Beiträge von Becker, Weber und Wierczoch [BWW08] sowie von Beher, Bönisch und Heidrich [BBH08] beschäftigen sich mit Software-Varianten und -Produktlinien in der Automobilindustrie. Software-Produktlinien in der Automobilindustrie sind sehr komplex. Für jede Elektronikkomponente gibt es eine Vielzahl unterschiedlicher Varianten. Ihre Variabilität wird von vielen Faktoren bestimmt. Zudem sind an der Entwicklung viele Parteien (mitunter mehrere Original Equipment Manufacturer, OEM, und fast immer mehrere Zulieferer) beteiligt.

Ein wesentlicher Bestandteil der Lösung von Becker et al. ist die explizite Modellierung von Variabilität über alle Produkte, Komponenten und wiederverwendbaren Artefakte hinweg. Dabei sollen die Modellierungsverfahren standardisiert werden. Je nach Anwendungsbereich müssen verschiedene Variabilitätsmechanismen vorgesehen sein. Teils sind aber auch übergreifende Vereinheitlichungen möglich. Weitere Vorkehrungen sind die umfassende Dokumentation von Konfigurationsentscheidungen und die Modellierung mehrstufiger Produktlinien nach dem Multi-Level-Konzept.

Auch Beher et al. beschäftigen sich mit dem Problem mehrstufiger Produktlinien. Sie schlagen eine Schnittstellenbeschreibung von Produktlinien vor, mit der die Wechselbeziehungen zwischen den an einem Entwicklungsvorhaben beteiligten Varianten erfasst und abgeglichen werden können. Außerdem beschreiben sie eine Vorgehensweise, um diesen Abgleich schrittweise vorzunehmen.

Um Produktlinien für geschäftliche IT-Anwendungen geht es in den Beiträgen von Lochmann und Grammel [LG08] sowie von von der Maßen und Wübbeke [MW08]. Einen umfassenden Ansatz, der Model-Driven Development (MDD), Aspect-Oriented Software Development (AOSD) und Software Product Line Engineering (SPLE) verbindet, streben Lochmann und Grammel an. Ihre Arbeit ist in zwei großen Forschungsprojekten angesiedelt, aus denen sie erste Erfahrungen berichten. Sie stellen ihr Anwendungsbeispiel vor, die exemplarische Produktlinie *Sales Scenario*, und berichte über drei wesentliche Herausforderungen bei der Etablierung dieser Produktlinie: Der Ausgleich von Stakeholder-Interessen, Anforderungen an die Entwicklung von Modellierungs- und Architekturbeschreibungssprachen und der gleichzeitige, koordinierte Einsatz mehrerer Sprachen und Modelle.

Von der Maßen und Wübbeke stellen ein Verfahren vor, mit dem die Artefakte der Anforderungsdefinition bei Produktlinien beschrieben werden können. Bei der Produktlinie handelt es sich um kundenspezifische CRM-Systeme (*Customer Relationship Managmenent*), die bei allen kundenspezifischen Ausprägungen auch viele Gemeinsamkeiten aufweisen. Ihr Ansatz verfolgt eine Bottom-Up-Stategie: Er geht aus

von einer im Unternehmen etablierten Vorgehensweise, die aber noch keine Produktlinienverfahren nutzt. Für die verwendeten Geschäftsskizzen und Geschäfts-Anwendungsfälle schlagen sie Erweiterungen zur expliziten Modellierung von Variabilität vor.

Drei Diskussionsblöcke im Workshop vertiefen die Beiträge und betrachten die Querbeziehungen zwischen ihnen. Die erste Diskussion zum Thema *Produktlinien in komplexen Organisationen am Beispiel der Automobilindustrie* geht aus von den Beiträgen von Becker et al. und Beher et al. Sie setzt auch die Ergebnisse der Studie von Fricker und Stoiber dazu in Beziehung.

Die zweite Diskussion, *Aktuelle Herausforderungen bei Produktlinien*, spannt einen Bogen über den gesamten Workshop. Schwerpunktthemen sind: Domänenspezifika (z.B. technische Software vs. IT-Systeme), organisatorische Maßnahmen und Prozesse, wirtschaftliche Motivation und Nutzen von Produktlinien sowie Einführung und Erhalten von Produktlinien.

Die dritte Diskussion trägt den Titel *Auf der Suche nach Lösungen: Was leisten Technologien wie Variantenmanagement und MDSD?* Sie betrachtet Schlüsseltechnologien von Produktlinien und versucht einen Ausblick künftige technologische Entwicklungen.

Literaturverzeichnis

[BWW08] Becker, M.; Weber, M; Wierczoch, T.: Varianten in der Automobilelektronikentwicklung: Herausforderungen und Lösungsansätze. In: Proceedings of Software Engineering 2008 (SE 2008), Munich, 2008.

[BBH08] Beher, U.; Bönisch, G.; Heidrich, M.: Multiple Software Product Lines in Automotive Software Development. In: Proceedings of Software Engineering 2008 (SE 2008), Munich, 2008.

[FS08] Fricker, S.; Stoiber, R.: Relating Product Line Context to Requirements Engineering Processes Using Design Rationale. In: Proceedings of Software Engineering 2008 (SE 2008), Munich, 2008.

[LG08] Lochmann, H.; Grammel, B.: The Sales Scenario: A Model-Driven Software Product Line. In: Proceedings of Software Engineering 2008 (SE 2008), Munich, 2008.

[MW08] von der Maßen, T.; Wübbeke, A.: Modellierung von Variabilität in der Geschäftsanalyse: Eine industrielle Fallstudie. In: Proceedings of Software Engineering 2008 (SE 2008), Munich, 2008.

[PIK08a] Produktlinien im Kontext 2008. Workshop-Homepage und -Archiv: http://www.sse.uni-hildesheim.de/pik2008/

[PIK08b] Produktlinien im Kontext 2008. Workshop-Bericht: http://pik2008.swpm.de

[Sc08] Schwanninger, Ch.: Product Lines: Technological or Organizational Challenge? Keynote Talk, PIK 2008 Workshop, Munich, 2008. (http://www.sse.uni-hildesheim.de/pik2008/)

PIK 2008 Workshop-Organisation

Organisatoren

Andreas Birk, SWPM

Ludger Fiege, Siemens

Klaus Schmid, Universität Hildesheim

Programmkomitee

Danilo Beuche, pure-systems

Ulrich Eisenecker, Univ. Leipzig

Samuel Fricker, ABB und Universität Zürich

Eva Geisberger, TU München

Birgit Geppert, Avaya

Lars Geyer, Robert Bosch

Paul Grünbacher, Univ. Linz

Andreas Helferich, Universität Stuttgart

Gerald Heller, Hewlett-Packard

Frank Houdek, Daimler

Dirk Janzen, Harman/Becker

Olaf Kath, ikv++

Michael Kircher, Siemens

Peter Knauber, FH Mannheim

Thomas von der Maßen, Bertelsmann

Dirk Muthig, Fraunhofer IESE

Klaus Pohl, Universität Duisburg-Essen

Mark-Oliver Reiser, TU Berlin

Matthias Riebisch, TU Ilmenau

Reinhard Stoiber, Univ. Zürich

Ramin Tavakoli, TU Berlin

Axel Uhl, SAP

Martin Verlage, Vereinigte Wirtschaftsdienste

Matthias Weber, Carmeq

Relating Product Line Context to Requirements Engineering Processes Using Design Rationale

Samuel Fricker[1,2], Reinhard Stoiber[2]

[1]ABB Switzerland Ltd.
Bruggerstrasse 72, 5400 Baden, Switzerland
samuel.fricker@ch.abb.com

[2]University of Zurich, Department of Informatics
Binzmuehlestrasse 14, 8057 Zurich, Switzerland
fricker@ifi.uzh.ch, stoiber@ifi.uzh.ch

Abstract: The design of engineering processes is dependent on the context they are designed for. This paper presents the results of an action research study that used design rationale to discover a number of contextual factors that provide criteria for designing a requirements engineering process in a given product line environment. The presented context dimensions and process design argumentation provide a starting point for understanding product line context and how supporting engineering processes are designed, communicated, and evaluated.

Kewords: product lines, context, requirements engineering, process design, design rationale, action research

1 Introduction

Product line engineering is adopted by a number of companies as a means to increase product individualization, to reduce development costs, to enhance product quality, and to reduce time-to-market of products. These benefits are achieved by reusing domain artifacts and exploiting product line variability [PBL05]. The use of a product line approach, however, comes with a number of challenges. Compared to single-product development, product line engineering introduces a new level of complexity in designing artifacts and engineering processes.

The product line approach adds additional challenges when used in large and distributed organizations. Coordination and communication problems will occur, as distance plays a role [GF07]. Here, requirements engineering is in the focal point, as requirements are a key instrument for achieving coordination. Requirements specifications define agreements between stakeholders. Therein contained requirements capture objectives and expectations of these stakeholders.

Without requirements engineering processes that integrate product line development, complexity can hardly be mastered. Communication breakdowns happen and islands of development emerge that lead to a product line that consists of heterogeneous, rather than homogeneous, parts. Quality of the resulting product line is disastrous, as short-sighted decisions are taken to meet reasonable delivery dates with non-fitting parts. Ultimately, the company risks loss of customers and revenue.

We used action research [DMK04] to better understand the factors that influence the design of such requirements engineering processes. The research was carried out parallel to the development of a concrete process in an industrial environment. Design rationale [MC96] was used to relate objectives and product line context to the design of the requirements engineering process that was adopted by the company. Design rationale helped to recognize and express the interdependencies of context and design. The conceptualization of context influences the shape of the process, and questions in the process design were used to discover the facets of the context that were relevant.

The paper describes those facets of product line context that matter in the design of a supporting requirements engineering process. It shows how design decisions can be justified by a specific constellation of product line context. The results provide a starting point for moving from tacit, experience-based, and person-dependent process design towards a model for using concrete characteristics of product line context for guided design and evaluation of engineering processes.

The paper is structured as follows. Section 2 describes background and related work. Section 3 characterizes the case organization and outlines the research method. Section 4 describes the conceptualization of context that emerged from the research study and the use of design rationale for the design of a requirements engineering process. Section 5 discusses the research results. Section 6 concludes.

2 Background and Related Work

The behaviour of development organizations is often characterized by *process reference models*. A process reference model describes a number of process areas in terms of the goals that are pursued and the practices that are implemented to achieve these goals. A widely spread reference model of product line organizations is the framework for software product line engineering [PBL05]. The process areas primarily related to requirements engineering are product management [WBN06], domain requirements engineering, and application requirements engineering.

In addition to the pursued goals, a number of *contextual factors* influence the design of engineering processes. For example, organizational structure [Ha03], social and political concerns [ORS05], and distance [FG07] have an effect on how requirements engineering is performed. To construct requirements engineering processes for specific product line contexts, process engineers would greatly benefit from a framework for conceptualizing the relevant context and identifying optimal design decisions. The lack of such a framework forces such engineers to adopt a costly trial and error approach.

The process reference models and a specific organization were taken as a starting point towards establishing a *joint model* of product line context and requirements engineering processes. *Design rationale* was used to relate product line context to the design of a requirements engineering process. Design rationale captures the motivations for initiating design, the objectives and conditions that give rise to shape [MC96]. In the context of process design, such rationale is also called method rationale [RTRL00].

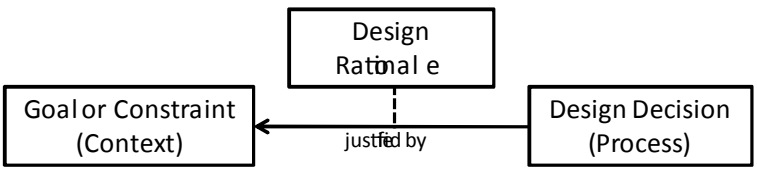


Figure 1. Design rationale relates process design to objectives and context.

Figure 1 describes the kind of design rationale used in this research. Process design is concerned of a number of design decisions such as which methods are used, how requirements are structured, and how stakeholders are involved. The process is designed to address a number of objectives and to fit contextual constraints. A specific combination of one design decision with its justifying goals and constraints is captured by a design rationale.

Design rationale captures design knowledge that emerges as a result of reflections during process design, independent of the process design method applied. It is useful to explaining the process, to support process evolution, and to advise other process designers in conceptualizing context and reusing design knowledge. The design rationale for a specific process can be generalized into a process pattern [HL05].

3 Research Method and Process

3.1 Studied Organization and Product Line

The research has been performed at ABB. ABB is a leader in power and automation technologies that help enable utility and industry customers to improve their performance while lowering environmental impact. The ABB group of companies operates in about 100 countries and employs more than 110,000 people. The research project took place in one of ABB's global organizations. The overall product line effort aimed at reducing hardware and software cost and at implementing new domain-specific standards. The organization had a tradition in product line engineering and performed at the product line maturity level [Bo02].

The first author of this paper was employed by ABB at the time of the study. Of the global organization a product management unit and a development unit, both located at two European sites, were involved in the project. The project client was senior management of the product management organization. A number of employees

participated in the project: one process champion, two process developers, one information system engineer, two global product managers, half a dozen of local product managers from Europe and Asia, four product and domain architects, representatives from marketing, and three line managers. The researcher participated as a leader of the process development effort, as a requirements engineering expert, and as a facilitator. The project regularly reported to a steering committee that consisted of senior management from the product management and the development units.

3.2 Research Question and Method

Canonical action research [DMK04] was used as a research methodology to be able to contribute to knowledge, while providing results of immediate practical value. The general goal was to understand the context that is relevant for the design of requirements engineering processes in a product line environment. The project client was interested in establishing a company-wide homogeneous requirements engineering process. The success of the project was measured by the degree of process adoption, satisfaction of process performers and reported evidence of economic impact.

The launch-point of this research was the observation that it was hard to transfer requirements engineering processes from one organization to another. The aim of the research project, hence, was to explore the relationships between context and the design of suited requirements engineering processes. Of particular interest was understanding the kinds of contextual factors that influence process design decisions. The following research question was formulated: *which facets of product line context are relevant for the design of requirements engineering processes?*

The project followed the *cyclical process model* of canonical action research, which consists of diagnosis, action planning, intervention, evaluation, and reflection phases. The here reported research is based on one iteration in this cycle, which lasted one year. Subsequent iterations were started to adjust the requirements engineering process to other situations that can be described using the presented model of product line context. A description of these following iterations is out of scope of this paper, however.

The *diagnosis* phase established the objectives of process development and a first understanding of the product line context that was relevant for process development. Senior managers, a process champion, and a number of process performers were interviewed. Senior management appointed representative parts of the product line for process development, piloting and evaluation.

During *Action planning* a tentative requirements engineering process was designed. Requirements engineering concepts and methods that seemed to optimally suit given objectives and context were selected. The proposed process was then analyzed, debated, and adjusted by all involved members of the organizations. This negotiation ensured understanding and sharing of the rationale behind the process design, relevance of the process for the concerns of the process performers, feasibility with given tool support, practicability within time and resource constraints, and commitment to piloting the process.

During the *intervention* phase the agreed requirements engineering process was piloted. The IT infrastructure was adjusted to support the process. Training was provided to the process performers, by involving them into the definition of the requirements engineering process during the action planning phase.

Evaluation included monitoring the process and was performed concurrently to intervention. For this purpose, requirements and requirements specifications were regularly reviewed and experiences of performing the process discussed. Identified inconsistencies between process specification and actual practice got resolved as rapidly as possible. Hence the process was continuously adjusted to the context.

Reflection was performed both by the research and the practitioner side. The process champion, a product manager, wrote an experience report and recommended the adoption of the piloted process in the product management unit on a wider scale. One architect became a champion for adopting relevant parts of the process in the development unit. Global senior management decided to roll the piloted process out for the complete product line and to support further process development. This marked the start of a second action research iteration.

4 PL-RE Context

4.1 Product Line Context

Major design decisions for requirements engineering processes in a commercial product line company concern the structure, flow, lifecycle and storage of requirements and the definition of how and when different requirements engineering activities are performed. During the action research project the following major categories of product line context have emerged and affected the design of the requirements engineering process:

- **Technical context**: requirements provide the rationale for the design of the various parts of the product line; hence the structure of the product line influences the requirements engineering process.
- **Business context**: the people and organizations benefitting from the product line justify the effort and resources that are put into its development; hence the argumentation of benefits influences the requirements engineering process.
- **Organizational context**: the people and organizations managing and developing the product line aim at pursuing their objectives and interests; hence the organizational structure and behavior influences the requirements engineering process.
- **Geographical context**: different means for communication and coordination are used, depending on the distance between collaborators; hence geographical separation influences the requirements engineering process.
- **Historical context**: development is organized differently, depending on the lifecycle state of the product line and its various parts; hence the evolution of the product line influences the requirements engineering processes.

The following subsections discuss these categories in more detail.

4.1.1 Technical Context

The *technical context* concerned the structure of *engineering artifacts*. Such artifacts are the result of development projects and are recognizable stable entities in the overall product line. Every such project was a consumer of requirements and produced requirements for other artifacts as a result of negotiations with other projects. Two kinds of relationships between artifacts were of particular importance: the use hierarchy and the composition hierarchy.

The *use hierarchy*, Table 1, had the following implications on the requirements engineering process: an engineering artifact implements requirements posed by the next higher-level entity and provides interfaces for its use. It also uses interfaces and controls features of the lower-level artifacts, hence is subject to constraints imposed by these entities. The design of engineering artifacts requires understanding and negotiation of expectations, constraints, and interfaces along this use hierarchy.

Table 1. Use hierarchy. A higher-level entity acts on lower-level artifacts.

Entities	Definition
Users	Humans configure, monitor, and control lower-level artifacts through the use of software tools.
Software Tools	Software tools store, process, and present data and provide access to lower-level artifacts.
Secondary Equipment	Software-intensive equipment automates the control and protection of lower-level artifacts.
Primary Equipment	Electrical-mechanical equipment acts on the electrical grid and take measurements.

The *composition hierarchy*, Table 2, had the following implications on the requirements engineering process: requirements for contained artifacts are derived from design decisions for a containing artifact. The reuse of lower-level artifacts provides the efficiency and quality advantages that are expected from the product-line approach, but introduces constraints for higher-level engineering artifacts.

Table 2. Composition hierarchy. A higher-level artifact is composed of lower-level artifacts.

Entities	Definition
Systems	Specific arrangements of products to provide services and applications.
Products	Stable arrangements of components that provide a number of related functions. A product appears on a price list.
Components	Hardware or software that is systematically reused in the development of products.

4.1.2 Business Context

The *business context* concerned the objectives to be achieved by engineering artifacts. A number of abstraction levels were identified [GW06], with different kinds of benefits to

be reached with the engineering artifact. Table 3 characterizes the four levels of the *benefit hierarchy*: strategy, economic factors, usage, and engineering artifact.

The business context provided the argumentation for justifying the concrete realization of the engineering artifact. The interactions between two abstraction levels were used for prioritizing and selecting requirements for releases of the artifact by exploring requirements refinement choices and evaluating requirements pertinence [HC88, Yue87, In04].

Table 3. Benefit hierarchy. Higher abstraction levels describe objectives of the company and lower levels how these objectives can be reached.

Abstraction Level	Definition
Strategy	Long and short-term company vision and goals pertaining to an engineering artifact and the market it addresses.
Economic Factors	Features that influence buying decisions and cost of the engineering artifact.
Usage	Requirements related to the usage of the engineering artifact, including functions and quality properties.
Engineering Artifact	Design decisions related to structure and behavior of the engineering artifact and utilized technologies.

4.1.3 Organizational Context

The *organizational context* concerned a number of *company functions* that acted on the engineering artifacts [Po85]. Every function has specific responsibilities and manages staff, knowledge, assets, and capabilities. The steering committee, consisting of senior managers, defined and managed the product line company. Product management steered the evolution of the engineering artifacts by coordinating marketing & sales, research & development, and production.

Negotiations between organizational units lead to a number of *contracts*. Every role had responsibilities and pursued objectives, which needed to be considered in the development of the engineering artifacts and balanced against each other. To speed up negotiation, product management and architects acted as principal agents and prepared contract proposals. A typical contract was a requirements specification for a release of an engineering artifact, which included expected benefits [GW06] and implementation proposals [FGM07].

4.1.4 Geographical Context

The *geographical context* concerned collaboration over distance. English was the language used for documentation as the organization was spread over the world. Instead of working in a document-based manner, requirements were stored in one common repository that was accessible from everywhere.

Specific tactics were used to adjust to local or remote collaboration. For example, while communication of requirements and domain knowledge are supported by frequent informal meetings in a collocated situation, a more structured, tool-based approach was needed to ensure requirements understanding with few meetings [FGM07].

4.1.5 Historical Context

The *historical context* concerned the evolution of engineering artifacts and the product line as a whole. Novelty of requirements played an important role; new requirements needed more effort and care to be analyzed, documented and communicated than requirements that were well known and understood. Compatibility of engineering artifacts over a number of versions yielded a number of constraints for development. Finally, the order of implementation defined the order of how requirements were fixed; early developed artifacts provided constraints for later artifacts.

4.2 Requirements Engineering Process and Its Design Rationale

The company decided to scope the process development project so that the requirements engineering process could be developed, piloted, and evaluated with given resources within one year. The process was developed according to the procedure described in section 3.2 by considering requirements engineering literature and by actively involving the process owners, participants and supporters. When consensus was reached, the process was piloted for two mission-critical products to gather lessons-learned.

The requirements engineering process addressed three *primary objectives*: continuous collection of needs, selection of requirements for a product releases, and communication of requirements to the development project team.

The *product line context* was fixed as follows:

- **Technical context**: requirements are engineered for one single control product (secondary equipment, product) that is part of a standardized control system and based on recently developed components. The product is operated through existing tools and controls well-established primary technology.
- **Business context**: product design needs to be justified by objectives from the whole benefit hierarchy.
- **Organizational context**: the requirements engineering process needs to be defined for the product management function. Product management acts towards research & development as a principal agent representing the steering committee, customers, marketing & sales and production.
- **Geographical context**: Product management is located at two sites in Europe, customers, marketing & sales, and production at various sites distributed over the whole world, and research & development at one of the two European sites.
- **Historical context**: the development adjusts an existing product to a new market and ports it on recently developed components. Hence, the majority of features

were well-known with a few quality attributes to be adjusted to the flavor of the new market. Compatibility concerns were defined as requirements.

Methods to address the three objectives in the given context were integrated into the overall *requirements engineering process* by basing the process on the lifecycle of market requirements [CR00]. To address need collection, the flow of needs towards product management was defined and responsibilities assigned to various roles working at a number of company sites. A globally accessible need database was established. Needs were collected from company management, marketing & sales, customers, and production. Requirements that were derived from the collected needs were structured according to the Requirements Abstraction Model [GW06]. Requirements selection was supported by informal goal [HC88, Yue87] and cost/value [KR97] analysis. The selected requirements were stored as a requirements specification in a globally shared requirements database. New requirements were specified in detail. Requirements related to reuse of features were specified by referring to the product line domain documentation. The requirements specifications for product releases were reviewed by important company-internal stakeholders of the product. Requirements communication was supported by handshaking with implementation proposals [FGM07]. Finally, a change management process was set up to manage change requests from product development.

A number of *design rationales*, summarized in Table 4, motivated the design of the requirements engineering process. These design rationales capture the reflections behind the process design and provide the motivation for both why the described contextual factors matter and why the process fits the specific context.

Table 4. Design decisions for the requirements engineering process and their rationales.

Design Decision	Rationale	Context
Backbone of process: market requirement lifecycle.	Engineering artifact: secondary equipment product.	Technical
	Process owner: product management.	Organizational
Need providers: steering committee, marketing & sales, customers, and production.	Product management acts as principal agent for these company functions.	Organizational
Globally accessible need database.	Globally distributed company.	Geographic
Distributed need collection responsibilities.	Globally distributed company.	Geographic
Requirements structure: Requirements Abstraction Model.	Justification of product design: argumentation using objectives from the whole benefit hierarchy.	Business
Goal analysis for requirements selection.	Justification of product design: argumentation using objectives from the whole benefit hierarchy.	Business
Cost/Value analysis for requirements selection.	Economic impact of development (implicit strategic objective).	Business

Design Decision	Rationale	Context
Varying requirements specification style.	Varied novelty of requirements.	Historical
Requirements specification reviewers: steering committee, marketing & sales, customers, and production.	Product management acts as principal agent for these company functions.	Organizational
Globally accessible requirements database.	Product management and research & development collaborate over distance.	Geographic
Requirements communication method: handshaking with implementation proposals.	Product management and research & development collaborate over distance.	Geographic
Change management process.	Stakeholders agreement required for changes to artifacts.	Organizational

5 Discussion

The requirements engineering process was developed and is in use in the described organization. Product management was satisfied with the *practical impact* of the process: "We get structure and quality in the work results and improve the communication about requirements. […] Because of the handshaking concept, research & development had significantly adjusted the scope of the development project and reconsidered one major decision about the components to be used in the product". Expressed was also, however, the early stage of process adoption: "Need collection is under start-up. Still to be addressed is that people assigned to the process also get the time needed for the work."

The positive results encouraged the continued use and further spread of the requirements engineering process. A second iteration of the research project was launched to extend the requirements engineering process for a more comprehensive coverage of the technical product line context.

The *research contribution* of the presented study is a conceptualization of product line context for the design of supporting requirements engineering processes. The role of design rationale for motivating relevance of contextual factors for process design was exemplified. The relationships between the requirements engineering process, the supported objectives, and the conceptualization of the contextual constraints provide the reasons that justify the process. With this understanding, process goals such as those of process reference frameworks [PBL05] can be achieved in a traceable manner.

Design rationale helps to split process development into manageable and foreseeable junks. Reusing design rationale helps to limit the scope of process development by limiting the context in which process design takes place. Further, design rationale can be used for designing a process in a constructive manner, rather than employing pure trial

and error. Documented design rationale helps to share design knowledge for advising process developers confronted with contexts new to them with already-made experience.

Design rationale helps to communicate and train a requirements engineering process. Without design rationale, the process appears as a number of steps and records stored in databases. With design rationale, process stakeholders start to see interdependent pieces of the process that together make meaning in the context they are applied.

Other observations concern the *role of product line variability in process design*. The acquisition, analysis, explicit representation, and use of variability are differentiating factors of product line engineering. In the design of the requirements engineering process, however, these practices have played a less dominant role than expected. They were supported to some degree by the chosen requirements structuring mechanism, the Requirements Abstraction Model [GW06], and by spreadsheets that characterized product variants in the product requirements specifications.

The product line was not managed from a central authority using a comprehensive variability model. Instead, catalogues of available engineering artifacts were used at the different levels of the composition hierarchy. Constraints between the artifacts were negotiated between the organizational units as part of the requirements reviews and requirements communication activities. Hence, much of the understanding of variability and commonality of products was distributed in the organization and the knowledge of people that participated in product line engineering.

6 Summary and Conclusions

Product line engineering has great potential for enabling companies to increase individualization and quality of products, while at the same time increasing development efficiency. The adoption of a product line engineering approach, however, leads to a number of challenges related to the size and complexity of the product line. As a consequence, supporting engineering processes must be carefully designed to achieve their objectives in their given environment. Overcoming costly trial and error in constructing such a process requires access to knowledge of successful design.

This paper describes a successful action research study performed in a large global product line organization. Design rationale was used to capture how the product line context influences the design of requirements engineering processes. Five categories of context were of particular importance: technical, business, organizational, geographic, and historical context. The paper characterizes these categories and shows their use for explaining the design of a specific requirements engineering process.

The research results provide a starting point towards a general design framework for engineering processes in a product line context. Such a framework will help to reduce some of the challenges of process design, hence reducing process improvement cost and increasing the yield and acceptance of engineering processes. Future work includes studying how changes in product line context affect the design of supporting requirements engineering processes.

References

[Bo02] Bosch, J.: Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. 2nd International Conference on Software Product Lines (SPLC2), 2002.

[CR00] Carlshamre, P.; Regnell, B.: Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes. International Workshop on the Requirements Engineering Process: Innovative Techniques, Models and Tools to support the RE Process (REP2000), 2000.

[DMK04] Davison, R.; Martinsons, M.; Kock, N.: Principles of Canonical Action Research. In Information Systems Journal 14(1), 2004; pp. 65-86.

[FGM07] Fricker, S.; Gorschek, T.; Myllyperkiö, P.: Handshaking Between Projects and Stakeholders Using Implementation Proposals. In (Sawyer, P.; Paech, B.; Heymans, P., eds.). 13th International Working Conference on Requirements Engineering: Foundations for Software Quality (RefsQ'07), Lecture Notes in Computer Science (LNCS) 4542. Springer, Berlin, 2007; pp. 144-159.

[GF07] Gorschek, T.; Fricker, S. (eds.): Proceedings of the 1st International Global Requirements Engineering Workshop (GREW'07), 2007.

[GW06] Gorschek, T.; Wohlin, C.: Requirements Abstraction Model. In Requirements Engineering 11, 2006; pp. 79-101.

[Ha03] Harmon, P.: Business Process Change – A Manager's Guide to Improving, Redesigning, and Automating Processes, Morgan Kaufmann, 2007.

[HC88] Hauser, J.; Clausing D.: The House of Quality. In Harvard Business Review, May-June 1988; pp. 63-73.

[HL05] Hagge, L.; Lappe, K.: Sharing Requirements Engineering Experience Using Patterns. In IEEE Software, Jan/Feb 2005; pp. 24-31.

[In04] In, H.: Requirements Negotiation Using Multi-Criteria Preference Analysis. In Journal of Universal Computer Science 10(4), 2004; pp. 306-325.

[KR97] Karlsson, J.; Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. In IEEE Software, Sept/Oct 1997; pp. 67-74.

[MC96] Moran, T.; Carroll, J., eds.: Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum Associates, 1996.

[ORS05] Ovaska, P.; Rossi, M.; Smolander, K.: Filtering, Negotiating and Shifting in the Understanding of Information System Requirements. In Scandinavian Journal of Information Systems 17(1), 2005; pp. 31-66.

[PBL05] Pohl, K.; Böckle, G.; van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005.

[Po85] Porter, M.: Competitive Advantage: Creating and Sustaining Superior Performance, The Free Press, 1985.

[RTRL00] Rossi, M.; Tolvanen, J.; Ramesh, B.; Lyytinen, K.; Kaipala, J.: Method Rationale in Method Engineering. 33rd Hawaii International Conference on System Sciences, 2000.

[WBN06] van de Weerd, I.; Brinkkemper, S.; Nieuwenhuis, R.; Versendaal, J.; Bijlsma, L.: Towards a Reference Framework for Software Product Management. 14th IEEE International Requirements Engineering Conference (RE'06), 2006.

[Yu87] Yue, K.: What Does It Mean To Say That A Specification is Complete?. Fourth International Workshop on Software Specification and Design (IWSSD-4), 1987.

Varianten in der Automobilelektronikentwicklung

Herausforderungen und Lösungsansätze

Martin Becker, Matthias Weber, Thomas Wierczoch

Team Prozessmanagement
Carmeq GmbH
Carnotstr. 4
10587 Berlin

{martin.becker, matthias.weber, thomas.wierczoch}@carmeq.com

Abstract: Der Beitrag beschreibt die gegenwärtigen Probleme im Umgang mit Varianten in der Automobilelektronik und leitet daraus Anforderungen an eine Lösung ab. Danach werden die Kernelemente eines bei der Carmeq GmbH entwickelten Lösungsansatzes zum systematischen, werkzeuggestützten Variantenmanagement in der Automobilelektronik vorgestellt.

1 Herausforderungen

Martin Winterkorn (Volkswagen Vorstand):
"Die Palette wird weiter die klassischen Segmente umfassen,… Aber wenn man wachsen will, muss man intelligent weitere Fahrzeuge hinzufügen und attraktive Nischen besetzen."

In unserer Projektarbeit haben wir zunehmend Probleme im Umgang mit Variabilität in der Automobilelektronikentwicklung festgestellt. Wachsende Produktpaletten mit verschiedenen Ausstattungsvarianten bei gleichzeitig steigendem, softwarebasiertem Funktionsumfang stellen die Entwickler in der Automobilelektronik zunehmend vor eine Vielzahl von Problemen.

In den letzten 20 Jahren hat eine erhebliche Segmentierung des Automobilmarkts stattgefunden. Grund dafür ist, dass die Kunden heute sehr viel differenziertere Wünsche an ihr Auto stellen und die Autohersteller ihre Produkte immer genauer auf diese Wünsche ihrer Zielgruppen zuschneiden. Dadurch nimmt die Zahl der Segmente innerhalb der Produktpalette zu, gleichzeitig aber werden die Segmente selbst immer kleiner. Darüber hinaus müssen sich die einzelnen Modelle klar von einander abgrenzen und von den Fahrzeugen der Mitbewerber unterscheiden. Es wird deshalb immer mehr Fahrzeuge mit immer mehr Varianten in der Ausstattung geben. Die Erschließung neuer Märkte und die dort geltenden regionalen und gesetzlichen Bedingungen führen zu einer zusätzlichen Variabilität.

Die genannten Aspekte und die stetig steigende Anzahl von in den Fahrzeugen integrierten Funktionen führen zwangsweise zu immer vielschichtigeren Systemstrukturen in den Fahrzeugen. Gleichzeitig besteht aus Kosten- und Qualitätsgründen das eigentlich gegenläufige Bestreben, in den Systemstrukturen strategisch ein steigendes Maß an Invariabilität, d.h. Verwendung von Gleichteilen, zu erreichen.

Diese Herausforderungen bezüglich Variabilität und Invariabilität stellen sich nicht nur für die Endprodukte, die Fahrzeugsteuergeräte, sondern auch für alle vorgelagerten Entwicklungsartefakte der Automobilelektronik – Lasten- und Pflichtenhefte, Architekturen, Funktionsmodelle, Implementierungsmodelle, Testmodelle etc. Es gibt zurzeit keine Methoden, um diese Herausforderungen bezüglich Variabilität und Invariabilität zufrieden stellend zu beherrschen. Die naive Übertragung von Produktlinienansätzen aus der Softwaretechnik berücksichtigt nicht in erforderlichem Maße die Besonderheiten der Automobilindustrie. Beispielsweise findet in der Automobilelektronik ein langfristiger, kontinuierlicher Rückfluss aus dem Application Engineering in das Domain Engineering statt. Aufgrund der Komplexität dieser Rückflüsse in Verbindung mit der Umsetzung strategischer Vorgaben aus dem Domain Engineering ist in der Regel eine mehrstufige Organisation der Produktlinien notwendig. Das Management solcher, im Kontext langfristiger Elektronikplattformen angelegter, mehrstufiger Produktlinien mit komplexen Rückflüssen wird heute nicht hinreichend beherrscht.

Die Heterogenität der Produktlandschaft ist aber nur ein Aspekt, der die Herausforderung durch Komplexität in der Automobilbranche ausmacht. Die komplexen Organisationsstrukturen in großen Automobilkonzernen führen zu langen Entscheidungsprozessen und die Auswirkungen der Entscheidungen auf die gesamte Wertschöpfungskette bleiben häufig undurchschaubar.

2 Anforderungen an eine Lösung

Die wichtigste Anforderung an einen Lösungsansatz zur Bewältigung der genannten Herausforderungen besteht in der Bereitstellung einer Modellierungstechnik, die es ermöglicht, sämtliche Strukturen und Entscheidungen, die Auswirkungen auf die Systemvariabilität haben, über die gesamte Produktlebensdauer, d.h. von Fahrzeugsteckbrief bis zum Auslaufen des Werkstattservice, explizit zu beschreiben. So entsteht eine einheitliche Basis zum Management laufender sowie zum Aufsetzen für zukünftige Entwicklungsprojekte. Außerdem sollte diese Modellierung als Grundlage für (teil-)automatisierte Verfolgung von Auswirkungen von Änderungsentscheidungen geeignet sein. Zusätzlich wird auf Basis der Modellierungstechnik eine Methodik und entsprechende Werkzeugunterstützung zum Management der Variabilität benötigt. Schließlich muss diese Methodik auf die vielfältigen und sehr unterschiedlich ausgeprägten Prozessschritte in der Automobilelektronik anpassbar sein.

3 Lösungsansatz: systematisches, werkzeuggestütztes Variantenmanagement

Die oben genannten Herausforderungen müssen auf vielen Ebenen angegangen werden. Mittel- und langfristig spielt dabei die Standardisierung im Bereich Elektroniksoftware, die in der Automobilindustrie derzeit durch die AUTOSAR Initiative vorangetrieben wird, eine sehr wichtige Rolle ([AS]). Die zentrale Herausforderung sehen wir jedoch im methodischen Bereich. Das Business Team Prozessmanagement der Carmeq hat deshalb auf Basis langjähriger Projekterfahrungen einen methodischen Ansatz zum systematischen Management von Varianten in der Automobilelektronik entwickelt und prototypisch evaluiert. Der Ansatz ist durch folgende Grundelemente gekennzeichnet

⇒ Die Modellierung der „internen" Variabilität der Entwicklungsartefakte durch spezifische, auf die jeweiligen Artefakte optimierte Mechanismen. Damit können spezifische Gegebenheiten von Anforderungs-, Entwurfs- oder Testnotationen sowie der dafür eingesetzten Tools optimal berücksichtigt werden.

⇒ Die einheitliche Modellierung der „externen" Variabilität der Entwicklungsartefakte, d.h. der verschiedenen Möglichkeiten der (stufenweisen) Konfiguration dieser Artefakte sowie der dabei zu beachtenden Abhängigkeiten durch Featuremodelle ([Ka90]).

⇒ Die Modellierung von strategischen Konfigurationsentscheidungen für Entwicklungsartefakte durch Product-Sets ([RW05]). Damit wird ein strategisches Modell zur systematischen Dokumentation strategischer Entscheidungen im Unternehmen definiert. Dieses strategische Modell dient als einheitlicher Ausgangspunkt für alle Entwicklungsprojekte innerhalb eines Unternehmens.

⇒ Die Modellierung von Konfigurationsentscheidungen innerhalb eines Entwicklungsprojektes auf Basis des strategischen Modells. Auf dem strategischen Modell aufsetzende Entwicklungsprojekte übernehmen initial die im strategischen Modell getroffenen Entwicklungsentscheidungen. Im strategischen Modell offen gebliebene Entscheidungen müssen im Entwicklungsprojekt getroffen und übernommene strategische Entscheidungen müssen im Kontext des Entwicklungsprojekts überprüft werden.

⇒ Die Modellierung und das Management von mehrstufig organisierten Produktlinien mit komplexen Rückflüssen vom Domain Engineering in das Application Engineering durch das Multi-Level Konzept ([RW07]).

3.1 Modellierung der „internen“ Variabilität mit artefaktspezifischen Techniken

In der Welt der Entwicklungsartefakte gibt es leider eine Reihe von Schwierigkeiten bei der Einführung von Mechanismen zur Darstellung von Variabilität:

⇒ Aufgrund der Heterogenität der Entwicklungsartefakte, beispielsweise aufgrund der Verwendung sehr unterschiedlicher Spezifikationstechniken in der Automobilelektronik, erscheint es fast unmöglich einen gemeinsamen Mechanismus zur Beschreibung von Variabilität einzuführen, der in allen Entwicklungsartefakten reibungslos funktioniert.

⇒ Darüber hinaus dominieren in vielen Anwendungsbereichen in der heutigen Praxis kommerzielle Werkzeuge mit proprietären Spezifikationstechniken. Zum gegenwärtigen Zeitpunkt bieten diese Werkzeuge keine umfassende Lösung zur Behandlung von Variabilität an. Daher müssen aktuelle Entwicklungsprojekte im Rahmen der Anpassungsmöglichkeiten dieser Werkzeuge pragmatische Lösungen finden. Leider führt dies zu einer Vielzahl verschiedener Darstellungen von Variabilität, selbst für den Fall das nur genau ein Artefakt und genau ein Werkzeug betrachtet werden.

Um dies etwas besser zu verdeutlichen betrachten wir die Phasen der textuellen Spezifikation sowie der modellbasierten Entwicklung von Steuergerätesoftware. In diesen Phasen kommen häufig Werkzeuge wie Requirements Management Werkzeuge wie Doors und Simulationswerkzeuge wie ML/SL/SF zum Einsatz ([Doors], [ML]). Da beispielsweise das Werkzeug ML/SL/SF keine explizite Darstellung von Variantionspunkten unterstützt, werden in der Praxis häufig Workarounds benutzt:

⇒ Ein Ansatz ist die Erstellung eines Modells in dem alle Varianten parallel modelliert werden und durch „An- bzw. Abschalten“ von Blöcken konfiguriert werden. Der Hauptnachteil bei diesem Ansatz ist die Tatsache, dass so ein „Gesamtmodell“ als solches kein sinnvolles System darstellt und daher schwierig zu verstehen und weiterzuentwickeln ist.

⇒ Ein anderer Ansatz ist etwa die „kreative“ Verwendung eines existierenden Konzepts, z.B. das Konzept konfigurierbarer Teilsysteme, um zumindest einige einfache Fälle von Variabilität beschreiben zu können.

Eine ähnliche Situation findet sich heute im Bereich des Requirements Management: verbreitete Werkzeuge bieten keine Konzepte zur expliziten Modellierung von Variabilität. Daher sind eine Reihe unterschiedlicher pragmatischer Lösungen und entsprechende Werkzeuganpassungen im Gebrauch ([So06]).

Aufgrund dieser Betrachtungen schlagen wir vor, im Bereich der „internen“ Variabilität von Artefakten wie folgt vorzugehen:

⇒ Da jedes Artefakt seine eigenen Charakteristika und pragmatischen Einschränkungen hat, müssen verschiedene Variabilitätsmechanismen unterstützt werden.

⇒ Um die Vielfalt der pragmatisch gewachsenen Variabilitätsmechanismen eines Artefakts zu beherrschen, muss für jedes Artefakt eine eindeutige und standardisierte Darstellung definiert werden, auf die alle existierenden Mechanismen abgebildet werden können. Dies bedeutet nicht, dass der Endanwender zwingend mit dieser Standarddarstellung arbeiten muss, sie dient lediglich als eine interne „Zwischensprache“.

⇒ Diese Standarddarstellung dient dann als Basis zur Verknüpfung der Variabilität verschiedener Artefakte. Dies wird im Folgenden beschrieben.

3.2 Modellierung der „externen“ (Konfigurations-)Variabilität mit Feature-Modellen

Von außen betrachtet möchte man im Rahmen der Produktlinienentwicklung ein oder mehrere „variable“ Entwicklungsartefakte, in denen Variationspunkte beschrieben sind, möglichst flexibel „konfigurieren“ können, d.h. einige der Variationspunkte auflösen und dabei jeweils eine der möglichen Varianten wählen. Die Konfigurationsabhängigkeiten können bei einem solchen Vorgehen schnell komplex und unübersichtlich werden. Daher hat sich in diesen Bereich eine spezielle Technik hierfür etabliert, die Featuremodellierung.

Feature-Modelle beschreiben abstrakte Eigenschaften von technischen Systemen sowie deren Beziehungen untereinander. Entgegen klassischen Definitionen des Begriffs Feature verstehen wir in diesem Zusammenhang unter einem Feature nicht nur benutzerwahrnehmbare Eigenschaften, sondern auch interne technische Funktionen und Eigenschaften. Bei jedem Automobilhersteller gibt es neben den vom Kunden konfigurierbaren Eigenschaften eines Automobils mehrere tausend solche Features, die bestimmten internen Funktionen und Eigenschaften entsprechen. Heutzutage sind diese Features selbst z.T. bereits gut dokumentiert. Dagegen werden die Beziehungen zwischen diesen zahlreichen Features nur sehr selten festgehalten. Dieses Defizit erschwert Entscheidungen und führt bspw. dazu, dass Entscheidungen mehrfach getroffen werden.

3.3 Modellierung von strategischen Konfigurationsentscheidungen

In der Praxis besteht die Konfiguration von Artefakten (z.B. im Falle eines neuen Elektronikbaukastens für eine bestimmte Menge von Baureihenprojekten, das Aufsetzen von Funktionslisten und die Festlegung für welche Fahrzeugmengen welche Funktionen umzusetzen sind) häufig aus einer Vielzahl von Einzelentscheidungen, in die z.B. vertriebsstrategische Überlegungen, rechtliche Gegebenheiten bzw. erwartete Entwicklungen, sowie funktionale Abhängigkeiten und Übernahmestrategien einfließen. Es ist offensichtlich, dass Mechanismen notwendig sind, diese Konfigurationsentscheidungen feingranular beschreiben zu können, um jederzeit nachvollziehen zu können, aus welchem Grund (bzw. aus welcher Verbindung von Gründen) eine bestimmte Funktion für eine bestimmte Menge von Fahrzeugen innerhalb des Baukastens ausgewählt wird.

Um dies zu erreichen, muss neben dem Featuremodell zunächst ein Modell der Fahrzeuge des Baukastens erstellt werden. Diese Modell wird im folgenden Produktlinienmodell genannt. Die Basis für das Produktlinienmodell eines Herstellers bilden klassifizierbare Eigenschaften der Produkte (z.B. Fahrzeugklasse, Markt, Ausstattungsvariante, etc.) und die möglichen Ausprägungen dieser Eigenschaften. Ein spezifisches Produktlinienmodell ergibt sich für einen Hersteller aus den gültigen Kombinationen der Ausprägungen der Eigenschaften. Eine Kombination dieser Ausprägungen wird als Product-Set bezeichnet. Ein Product-Set bildet eine Teilmenge innerhalb des Produktangebots. Das Product-Set {Markt=USA} umfasst zum Beispiel alle Fahrzeuge für den US-amerikanischen Markt. Dabei werden sowohl aktuelle als auch zukünftige Produkte erfasst.

Das *Strategische Modell* entsteht durch die Verknüpfung von Elementen des Feature-Modells mit Product-Sets aus dem Produktlinienmodell. Dabei stellt jede dieser Verknüpfungen eine strategische (Konfigurations-)Entscheidung dar. Die strategische Entscheidung „keine Sprachbedienung in den Fahrzeugen der Ausstattungsvariante Low“ wird z.B. durch eine Relation zwischen einem Feature Sprachbedienung im Feature-Modell und dem Product Set {Ausstattungsvariante=Low} abgebildet.

Die Gesamtheit aller dokumentierten strategischen Entscheidungen bildet das Strategische Modell eines Unternehmens. Dieses Strategische Modell dient als Basis für alle konkreten Entwicklungsprojekte. Weitere durch das Featuremodell geforderte Entscheidungen, die innerhalb des Strategischen Modells offen gelassen wurden, müssen durch das jeweilige Entwicklungsprojekt getroffen werden.

3.4 Die Modellierung von Konfigurationsentscheidungen in Entwicklungsprojekten

Ein Entwicklungsprojekt übernimmt initial die Konfigurationsentscheidungen aus dem Strategischen Modell, welche für dieses konkrete Entwicklungsprojekt relevant sind. Die Relevanz von Entscheidungen für ein Entwicklungsprojekt ergibt sich aus den für das Entwicklungsprojekt gegenüber dem Produktlinienmodell festgeschriebenen Ausprägungen der Produkteigenschaften. So wird z.B. für ein Fahrzeugprojekt, welches ein Fahrzeug für die USA entwickeln soll die Produkteigenschaft {Markt=USA} festgelegt. Damit sind Entscheidungen des strategischen Modells, welche sich beispielsweise auf den europäischen Markt beschränken ({Markt=EU}) für dieses Entwicklungsprojekt nicht relevant.

Auf der Basis der aus dem Strategischen Modell übernommenen Konfigurationsentscheidungen müssen während des Entwicklungsprojekts sowohl diese übernommenen Entscheidungen im Kontext des aktuellen Entwicklungsprojekts überprüft als auch weitere durch das Featuremodell geforderte Entscheidungen, die innerhalb des Strategischen Modells offen gelassen wurden, getroffen. Wird eine aus dem Strategischen Modell übernommene Entscheidung im Entwicklungsprojekt anders entschieden, so muss dies begründet werden. Durch diesen Ansatz wird eine durchgängige Transparenz in den Entwicklungsentscheidungen von den strategischen Ebene bis hin zu konkreten Projektentscheidungen sichergestellt.

Entwicklungsprojekte können grundsätzlich auf verschiedenen Ebenen aufgesetzt werden. So gibt es Entwicklungsprojekte, welche ein gesamtes Fahrzeug als Entwicklungsgegenstand haben als auch Entwicklungsprojekte, deren Ziel die Entwicklung einer einzelnen Komponente (z.B. Radio) umfassen.

3.5 Die Modellierung von mehrstufig organisierten Produktlinien mit dem Multi-Level Konzept

Die bisher beschriebenen Mechanismen basieren alle auf der Annahme einer „idealen“ produktlinienorientierten Entwicklung, d.h. spezifische Produkte entstehen durch nichts anderes als die bloße Konfiguration von variablen Artefakten des Domain Engineering. In der Praxis ist diese Situation jedoch eher die Ausnahme als die Regel: üblicherweise entstehen durch die Konfiguration lediglich die Ausgangsartefakte von spezifischen Produktentwicklungen. Im Laufe eines konkreten Entwicklungsprojektes werden dann diese Ausgangsartefakte projektspezifisch modifiziert. Beispielsweise ist zu erwarten, dass bei bestimmten Funktionen eines neu aufgesetzten Baureihenbaukastens im Rahmen der Kostendiskussionen mit dem Zulieferer Erweiterungen bzw. Reduktionen zum strategischen Referenzbaukasten vorgenommen werden. Manche dieser Änderungen (z.B. bestimmte funktionale Erweiterungen) können explizit erwünscht sein, andere hingegen sind explizit unerwünscht (z.B. nicht abwärtskompatible Änderungen an der Signalschnittstelle), da dadurch strategische Ziele gefährdet werden.

Es ist von entscheidender Bedeutung, diese Art von Bewertungen von Abweichungen spezifizieren zu können, sowie sicherzustellen dass bei solchen „Nachbearbeitungen“ des konfigurierten Artefakts die Beziehungen zum Ursprungsartefakt nicht verloren gehen. Um dies zu unterstützen verwenden wir das Multi-Level Konzept ([RW07]). Mit dieser Technik können mehrstufig organisierte Produktlinien unterstützt werden. Im wesentlichen definiert das Multi-Level Konzept die Beziehungen zwischen Referenzartefakten, bspw. einer Referenzbibliothek von Funktionsspezifikationen und referenzierenden Artefakten, bspw. den Funktionsspezifikationen eines aus der Referenzbibliothek abgeleiteten Elektronikbaukastens für eine bestimmte Menge von Fahrzeugprojekten. Für jede Funktion aus der Referenzbibliothek kann festgelegt werden, ob und an welchen Stellen welche Arten von Abweichungen (bspw. zusätzliche Ein-/ Ausgabesignale, oder zusätzliche Teilfunktionen) erlaubt sind. Damit kann jederzeit analysiert werden, welche Abweichungen vorhanden sind und welche davon erlaubt bzw. unerlaubt sind.

Durch die Multi-Level Technik sind also Abweichungen vom Strategischen Modell in Entwicklungsprojekten möglich, werden aber erst durch die gezeigte Herangehensweise transparent.

Darüberhinaus bietet die Multi-Level Technik die Basis um kontrolliert und rechnergestützt Änderungen in beide Richtungen, d.h. vom Referenzmodell zu den Projektmodellen und umgekehrt, zu übernehmen ([Ne08]). Der letztere Fall, d.h. der Rückfluss vom Application Engineering in das Domain Engineering, tritt in der Praxis regelmäßig auf, da die referenzierenden Artefakte im Rahmen der Entwicklungsprojekte, z.B. im Rahmen der aufwendigen Entwicklungsreviews mit den Projektzulieferern, an manchen Stellen eine sehr hohe technische Qualität erreichen, die man natürlich für Folgeprojekte nutzen will. Auch werden bestimmte, z.B. aus Kostendruck resultierende, projektspezifische Lösungsideen, häufig als Referenzlösung für neue Projekte übernommen.

3.6 Zusammenfassung der Methodik

Durch diese Methodik des Anforderungs-/Variantenmanagement wird ein explizites Aufzeigen komplexer Variabilitätsstrukturen und Zusammenhänge möglich und es wird ein bewusster Entscheidungsprozess bezüglich des Aufbaus und der Pflege von mehrstufig organisierten Produktlinien gefordert und gefördert. Aufgrund der Komplexität der zu dokumentierenden Strukturen ist eine entsprechende Werkzeugunterstützung unerlässlich.

4 Werkzeugunterstützung

Eine prototypische Umsetzung des vorgestellten Ansatzes im Bereich Anforderungsmanagement hat die Carmeq GmbH gemeinsam mit der TU Berlin durch Erweiterungen des Anforderungsmanagementwerkzeugs DOORS sowie durch einen Eclipse-basierten Featureeditor realisiert. Durch den vorgestellten methodischen Ansatz

und die prototypische Werkzeugumsetzung ist jetzt die Darstellung komplexer Variabilitätsstrukturen möglich und mehrstufige Produktlinien im Bereich Anforderungsspezifikationen können explizit dokumentiert und verwaltet werden.

Außerdem ermöglicht die Werkzeugunterstützung aus einer gemeinsamen Basisspezifikation stufenweise Spezifikationen zu generieren die genau auf den gewünschten Ausprägungsgrad hin angepasst sind (z.B. als Basis einer neu entwickelten Elektronik–Plattform, einer neu entwickelten Steuergeräte-Familie, oder einer spezifischen Baureihenentwicklung). Nach der Konfiguration können mit Hilfe des Multi-Level Konzepts die Informationsflüsse zur Ausgangspezifikation beidseitig verwaltet werden.

5 Literaturverzeichnis

[AS] http://www.autosar.org

[Doors] htttp://www.telelogic.com

[ML] http://www.mathworks.com/

[Ka90] Kang, K. C. et al.: “Feature Oriented Domain Analysis (FODA) – Feasibility Study”. Technical Report, CMU/SEI-90-TR-21, 1990.

[So06] Sokenou, M.: Produktlinienorientiertes Anforderungsmanagement im Automobilbereich – Evaluierung und Erweiterung bestehender Konzepte, Diplomarbeit TU-Berlin, 2006.

[Ne08] Neitsch, L.: Multi-Level Modellsynchronisation für die Entwicklung komplexer Produktfamilien softwarebasierter Systeme, Masterarbeit HPI Potsdam, 2008, to appear.

[RW05] Reiser, M-O..; Weber, M.: Using Product Sets to Define Complex Product Decisions, in: J. Henk Obbink, Klaus Pohl (Eds.): Proceedings of the 9th International Conference on Software Product Lines (SPLC 2005). Lecture Notes in Computer Science 3714, Springer 2005, pp. 21-32.

[RW07] Reiser, M-O..; Weber, M.: Multi-Level Features Trees -- A pragmatic approach to managing highly complex product families, RE Journal, vol. 12, no. 2, pp. 57-75, 2007.

Multiple software product lines in automotive software development

Uwe Beher[1], Günter Bönisch[1], Mike Heidrich[2]

[1] VDO Automotive AG / A company of Continental Corporation, Regensburg
uwe.beher@esg.de, Guenter.Boenisch@continental-corporation.com

[2] Fraunhofer Einrichtung Systeme der Kommunikationstechnik (ESK), München
mike.heidrich@esk.fraunhofer.de

Abstract: Since today's well known software product lines (SPL) approaches [At00] [Ba99] [Ka90] [Ka98] [Lö04] [MND02] [Ob00] [Pu08] [WLT99] focus on one single SPL, a methodology for connection and adjustment of multiple product lines is needed. The paper will shortly survey existing processes and methods for product lines and show adaptations to automotive industry. The focus of the paper will be an adapted process for automotive functional development, which is based on multiple software product lines (MSPL) and particularly regards customer-supplier relationship. The paper proposes a SPL interface to manage the software development using MSPL. The interface consists of a SPL Interface Methodology which defines steps for the adjustment of two or more different SPL as well as a data interface (SPL Interface Data) which defines a data format for the exchange between different SPL tools. The SPL adjustment is demonstrated using a case study of an imaginary advanced driver assistance system called mobilSoft Adaptive Cruise Control (MCC), which consists of three product lines, one for the whole MCC and two for the subsystems linear tracking and traverse control.

1 Introduction

In automotive functional development aspects like software reliability and productivity draw more and more attention. As a consequence automotive manufacturers and suppliers address improved software engineering processes. Introducing software product lines (SPL) increases reuse of software elements and supports accomplishing higher quality at less effort. Implementing improved SPL adaptations for automotive industry was one of the goals in the research project mobilSoft [So07]. With the help of latest scientific methods and the experiences of the partners in industry, which were automotive manufacturers and suppliers, solutions for manifold requirements to existing development processes especially for automotive applications were found. Among these requirements were short development timelines, less development effort and high quality for each single software product. However, a more and more important challenge for development processes is the increasing variability and complexity of the final products. Optimized SPL can help automotive manufactures and suppliers to meet this challenge while fulfilling existing requirements to their development processes.

Four steps were stated as necessary for Automotive OEMs or suppliers to obtain optimized SPL and to get to a functional demonstrator, which verifies the feasibility of the planned adaptations for the SPL. The first step was an analysis at automotive manufactures and suppliers which lead to global requirements and characteristics for automotive SPL and supporting processes. The second step was the evaluation of existing methods and tools, which define the technical state of the art, with the requirements found. The third step was evolving a specification of an automotive specific SPL, because existing approaches for SPL were not particularly made for automotive application. The last step was the setup of a demonstrator, where the appropriate processes and methods were verified. Aim of this last step is not only proving the concept of an adapted process, but also testing acceptance of comprehensive tool landscapes which are often combined with the introduction of integrated approaches. Among these four steps the analysis of existing methods in the academic environment and available tools on the market turned out to be an appropriate basis for introducing SPL improvements. In order to adapt existing tools to fit into existing processes, systematic and purposive investigations and determination of efforts are necessary. Especially introducing new methods in existing complex development structures is a critical task.

Additionally, existing methods and processes were investigated for coverage of the interface between automotive OEMs and suppliers. Because of missing approaches for this important task in automotive development, additional methods were evolved which fit into to existing SPL as an extension of the process framework. The paper will have a main focus on this aspect and will propose an SPL Interface Methodology for adjustment of two or more different SPL and SPL Interface Data as data interface during parallel development in different organizations. The result of applying these interfaces is a connection of single SPL to a Multiple Software Product Line (MSPL).

2 State-of-the-art software product lines

2.1 Existing SPL approaches

The arrangement of software development as a software product line is an effective method for increasing reuse in software development and well known in branches with high software share.

Short	Name	Short description
FODA	Feature Oriented Domain Analysis	Domain analysis method based on feature trees [Ka90]
FORM	Feature Oriented Reuse Method	Extension of FODA for domain design [Ka98]

COPA	Component Oriented Platform Architecting Method	Approach for setup of SPL based on architecture and component assets [Ob00]
FAST	Family-Oriented Abstraction Specification Translation	Total SPL framework [WLT99]
QADA	Quality Driven Architecture Design	Quality oriented software design [MND02]
PuLSE	Product Line Software Engineering	Generic and comprehensive framework for SPL [Ba99]
KobrA	Komponentenbasierte Anwendungsentwicklung	Component based software development, application of PuLSE [At00]
EAST-ADL	EAST - Architecture Description Language	Usage of variants in automotive software design [Lö04], [Vo04]
pure::variants	pure::variants	SPL modeling tool [Pu08]

Table 1: Overview of SPL approaches

It was initially applied e.g. in telecommunication, medical equipment or consumer products, but recently automotive industry shows increasing interest as well. In table 1, a short overview of existing SPL approaches is given. The listed SPL approaches mainly differ in the focus on software engineering tasks. The application of one of these approaches as whole or partial as individual interpretation of the SPL methodology is, in some instances, realized at automotive manufacturers or suppliers. However, the adaptation of the given approaches to the needs of automotive product development remains being a very complex task and only few parts of the listed approaches fit in the special automotive environment. Thus the following chapter proposes characteristics needed for a SPL to be suitable as an (automotive) MSPL.

2.2 Characteristics of an automotive SPL

Regarding [Bö04] the SPL is divided into the process areas domain engineering and application engineering. Domain engineering leads to the setup of the software product line and the design of the assets, application engineering deploys the assets of the software product line in order to generate a final product. According to the generative approach [CE00] the automotive SPL is divided into requirements space and solution space. This results in the following alignments of the SPL:

- Domain analysis is the requirements space of the domain engineering

- Domain design is the solution space for the domain engineering
- Application analysis is the requirements space of the application engineering
- Application design is the solution space for the application engineering

A common method for organizing the requirements space is the usage of feature models for variant requirements, which include features and the relations between features. Generally a feature model e.g. FODA represents all requirements which lead to functional properties in the product line.

- A Feature describes the requested product properties from the point of view of a stakeholder, which in the context of a SPL is a customer, developer, manager, investor or supplier. A feature can be derived from single requirements or a set of aggregated requirements.
- The feature relation describes the relationship between one or more features, which is mandatory, optional or excluding.

The selections of features in the feature model during application engineering builds up single products of solution space.

The solution space contains a solution model for the domain and the resulting solutions by specific selections in the solution model. The solution selection itself is the result of decisions made along the solution path. The solution model consists of selected assets and their relationship. An asset is the smallest unit of a solution model and can't be divided into smaller units. Assets can be categorized into three types.

- A basic asset is a solution, which can be used in many products without any change.
- A customized asset is a basic asset, which can be used in many products and which includes complete calibration possibilities.
- A specific asset is a solution for only one product, which is initially introduced into the product line and which aims at utilization for further products.

For example an innovation is a specific asset that is implemented in a product for the first time and which, in the case of a successful introduction in the market, is planned to be extended to other products as customized or basic asset. Aligning the design of an asset with an existing product line eases the implementation of the project specific solution into the SPL.

The feature-asset relation expresses the relationship between an asset and a feature. Because of a feature being a set of one or more requirements, an assignment between requirements and solution is achieved. The FA Linker (feature asset linker) includes the relationship between chosen features and related assets and thus regards only the relevant subset from the total set of all feature asset relations. For the interface between SPL, which is described in the following chapter, the FA linker plays a crucial role.

With the characteristics and main tasks proposed for SPL software engineering, an appropriate fundament is formed for the specification of an automotive SPL. This leads to connecting different SPL to a Multiple Software Product Line (MSPL), which is also an automotive requirement.

3 Connection of software product lines

3.1 Fundamentals of connecting product lines

The automotive final product “vehicle” of the OEM is not the result of only one integrated product line. It rather consists of different subsystems with their own product lines and additional single solutions e.g. innovations, which are initially implemented only in few car models. Additionally, each subsystem may be delivered from its own organization, which may be internal or an external supplier. As a result the structure of each product line and the stakeholders mapped to it may differ.

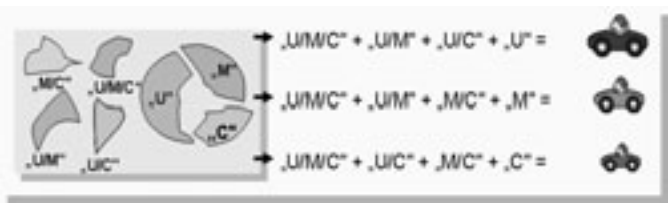


Fig. 1: Example: Structure of an OEM SPL

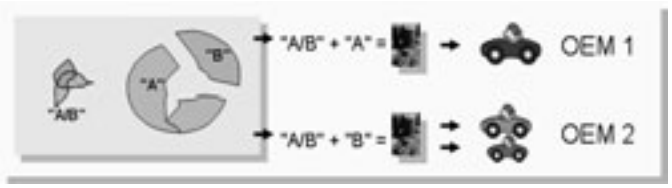


Fig. 2: Example: Structure of a supplier SPL

Examples for possible product structures with assets at OEM and supplier are depicted in figure 1 and figure 2. Figure 1 shows an OEM with its three platforms “Upper”, “Middle” and “Compact”. The “Upper” platform itself consists of four assets. “U/M/C” is used in “Upper”, “Middle” and “Compact” platform, “U/M” in “Upper” and “Middle” platform, “U/C” in “Upper” and “Compact” platform and finally “U” only in “Upper”, which is a single solution. The structure of the supplier in figure 2 is similar in methodology but different for the solution. Its platform “OEM1” consists of “A/B”, a common asset for its platform “OEM1” and “OEM2”, and a single solution “A”. It is used in the “Upper” platform of the OEM in figure 1, but not restricted to it. The supplier has actually no influence on the OEM decision, to put it into “U/M/C”, “U/M”, “U/C” or “U”. And finally, supplier may sell a similar product to another OEM, which will not be regarded in the SPL of the first OEM. As a consequence, even if products from the supplier are used in the platform of the OEM, the configuration of assets which build up the solution space is independent.

Furthermore a simple one-to-one match in the solution space where a product “A/B”+”A” may fit into “upper class” vehicles is hardly applicable for automotive industry. In general, separate SPL exist at different levels of functional hierarchy and depend on their own field of activity. In this case, all individual SPL need to be synchronized in order to deliver the correct product at the right time.

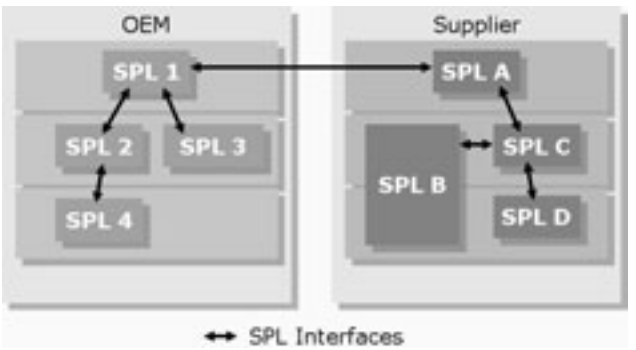


Fig. 3: Connected software product lines

After synchronization, the setup being composed of connected SPL is stated as a Multiple Software Product Line (MSPL). Figure 3 shows an example for the structure of connected SPL. The interfaces in this example occur between SPL internally in one company or externally in many companies. For a functional connection of SPL it is crucial, to align each single SPL in a correct hierarchical order.

A layered architecture e.g. EAST-ADL [Lö04] is able to structure SPL hierarchically. In the given example, “SPL A” at supplier may be at the highest level “user” as interface to the customer. But also internal product lines “SPL C” at “cluster” level or “SPL D” at “platform” level need correct alignment to facilitate “SPL A” providing the requested product.

3.2 Interfaces of multiple software product lines

As stated in the last chapter an important prerequisite for a MSPL is the synchronization of each SPL to each other in order to align development of the SPL assets. This paper proposes two main tasks for synchronization:

- Exchange of requirements and design data via a standardized data model “SPL Interface Data”
- Adjustment of SPL specific tasks via a general methodology “SPL Interface Methodology”

The SPL Interface Data generally contains design data like:

- Product requirements
- Features of the product as set of requirements
- Architecture patterns as reference solutions

The data exchange for synchronization mainly takes part at an early stage of the development process in a SPL. An appropriate phase is after domain analysis, where each single SPL defines its own reuse concept and asset structure for the product development. Before reaching domain design phase, inputs from all other SPL complete own basic product requirements by aligning own product application to the requirements of the total product. The SPL interface based on the data models and synchronization tasks needs to cope with a contradiction. On the one hand it shall be as flexible as needed to react on changed constraints after domain design, which at this point typically come from functional implementation. On the other hand the SPL interface shall serve as a consistent backbone for development data. Figure 4 illustrates the role of SPL Interface Data and SPL Interface Methodology in the context of single SPL implemented into the total product development.

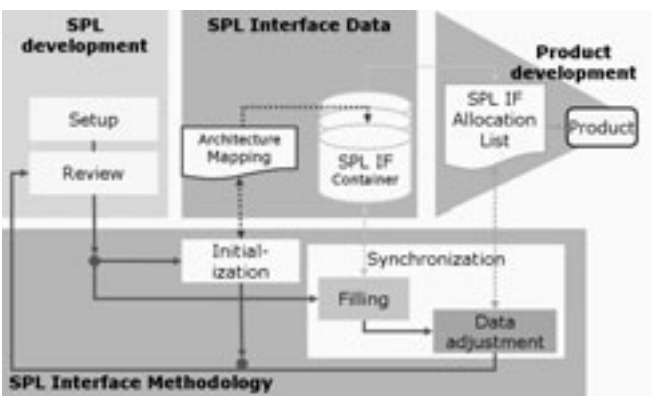


Fig. 4: Concept of the SPL interface

The part SPL development represents the individual SPL process of an organization like a company or a profit centre in a company. The product development covers the development of the final product as the result of the combination of the individual SPL processes. The SPL interface in between consist of two main elements, the central data container SPL Interface Data and the SPL Interface Methodology in order to synchronize the individual SPL processes. The SPL Interface Data serve as exchange mechanism for design data. It consists of:

- an architecture mapping
- an SPL IF Allocation List
- the data container itself

The reason for an architecture mapping is founded in the different SPL approaches, where the connection into the total product architecture is not provided. The architecture mapping verifies the architecture patterns of the single SPL and assigns each component to its correct place in the product, thus a common understanding of the product is established throughout all SPL.

The allocation list synchronizes the product features and assets and supports the compliance of the requirements to a consistent product design. It deals as configuration support and is a main document for the total product development.

The data container is the physical container for design data for combined SPL development which are mainly requirements and assets. It is recommended to structure the container in terms of connected SPL, this eases each SPL updating their own content during change management.

The SPL Interface Methodology describes the process for connecting, adapting and aligning two or more SPL. It provides initialization and synchronization, which realize a strong connection of the individual development processes to the final product. The SPL Interface Data and the SPL Interface Methodology will be explained in more detail in the following chapters.

3.3 Architecture mapping and initialization

A key element of a SPL is a reference architecture, where single products are derived from. In order to connect software product lines, the elements of the reference architectures have to match together to generate a functional total architecture. Reference architectures may be EAST-ADL [Lö04] or Car-DL [Wi06]. The common base of most reference architectures is a layered system model, where abstract system elements at a higher level are divided into smaller elements at a lower lever which provide more details of the internal structure.

The architecture mapping coordinates the different reference architectures at initialization of SPL alignment. It contains methods for the combination of requirements and solution spaces in the correct level of abstraction. Provided, that each individual SPL has a reference architecture, which fits into the general architecture frame for the architecture mapping, the architecture mapping merges all SPL into one central architecture of the MSPL. As a consequence of all reference architectures being abstract and generic, no SPL has to open its internal detailed architecture to the central architecture of the MSPL for the final product. That is e.g. strategic product plans and customer structures, which are often modeled in internal architectures, stay closed to other competing SPL.

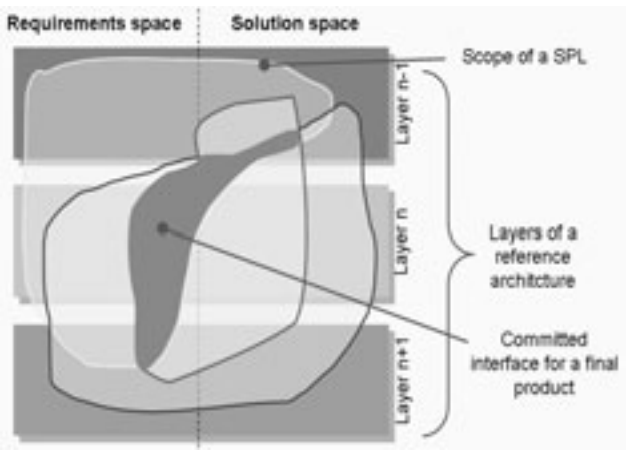


Fig. 5: Alignment of different reference architectures at initialization

Figure 5 shows the result of architecture mapping of individual SPL. In this figure, three possible SPL are drawn as bordered shapes. The main objective of the architecture mapping in the SPL Interface Data is to assign each element of the single SPL to the correct layer of the reference architecture and to requirements or solution space. By overlaying all SPL valid intersections of all SPL can be determined. The dark grey area in the middle of all shapes is the common part of all SPL, which needs a common problem and solution descriptions. Otherwise areas without any interaction have no significance for the SPL Interface Methodology.

3.4 Synchronization and allocation

In a MSPL individual SPL exchange data mainly during synchronization phase. The data base for the data exchanged is the SPL IF container. Content of the container are requirements, features as sets of requirements, assets, relations of container elements and additional documents, which support version control and asset history. Synchronization is separated into two steps, filling and data adjustment. During filling phase, all interface data of the other product lines are collected and structured. Basis of the structure is the architecture mapping described previously, which is elaborated during initialization.

As a result, a comprehensive configuration for all features, assets and relations is found, which is valid for realizing the total product. This configuration is a global SPL IF allocation list which contains the relationship between all features and assets involved in the product development.

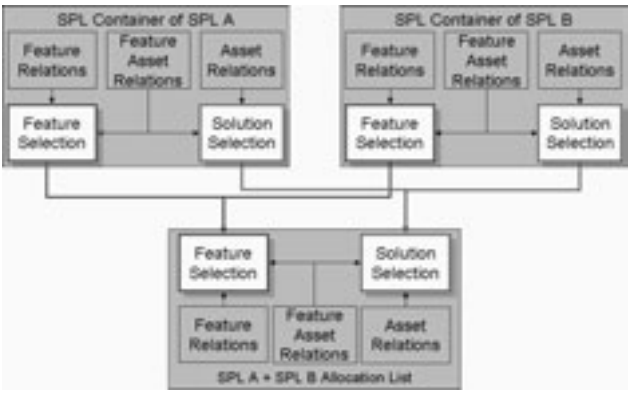


Fig. 6: Allocation list

Figure 6 shows the structure of the allocation list, if two SPL "SPL A" and "SPL B" are connected. Only if the allocation list as a combination of both SPL contains only viable product solutions, data adjustment phase is finished. Product application design and domain design follow after the adjustment phase. In case of changes in single SPL are required due to the result of a review, the overall SPL adjustment is triggered again.

4 Demonstrator for Multiple Software Product Lines

4.1 Main task of the demonstrator

The demonstrator supports the verification of an adapted SPL regarding implementation into a MSPL. A second effect, which development organizations should not neglect, is the acceptance test for introducing integrated software tool landscapes which comes along with the SPL adaptation. An option for a MSPL demonstration is building up the individual SPL as hardware independent function, which eases realization of the demonstration. The software platform to be used may be a standardized platform like AUTOSAR, but AUTOSAR is not a prerequisite for the demonstrator. The concept of the stakeholder need for performing the interface for the SPL does not reflect existing personnel but describes abstract roles and tasks that need to be performed. The mapping of the requested roles to an organizational chart has to be found for each organization individually.

4.2 Results of connection of software product lines

Figure 7 shows the setup of the function mobilSoft Cruise Control (MCC), which is composed of three product lines: the overall MCC function, linear tracking for MCC function and the traverse control for MCC function. In this case study tracking and traverse control are SPL of two automotive suppliers and the total MCC function a SPL at one OEM. The modeling of all feature models for the problem and solution space was realized with pure::variants [Pu08]. Architecture mapping was accomplished within the research project mobilSoft [So07] by the use of Car-DL [Wi06] as a reference architecture. However the shown approach can be used with other architectures, e.g. EAST-ADL [Lö04] as well. The appropriate files for the SPL IF container and the SPL IF allocation list were realized as *.xml-documents according to XMS scheme definitions (XMD) as output of UML based data models. Finally an iterative refinement of the MSPL approach was accomplished.

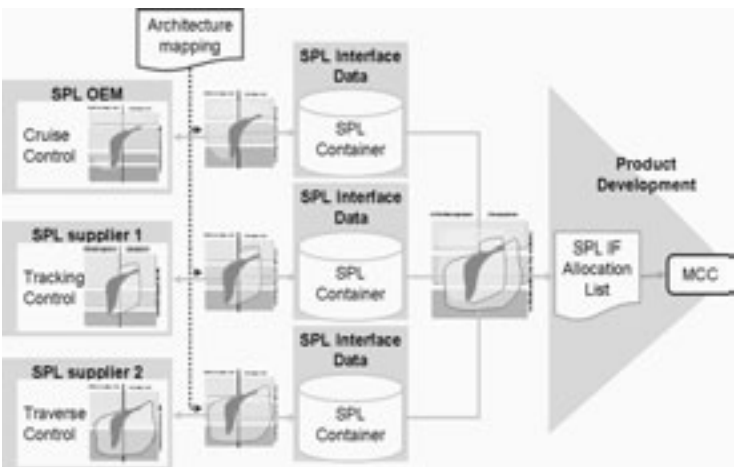


Fig. 7: mobilSoft Cruise Control as a MSPL

5 Conclusion

This paper has provided the structure of SPL Interface Methodology and SPL Interface Data in order to adjust many individual SPL into one Multiple Software Product Line. The result is a global adjustment list for requirements and solution space of all involved SPL.The adjustment list is the base of product application in each SPL and is an additional specification at the early stage of a product development process. In automotive applications, after implementing additional requirements emerging from the adjustment list, existing SPL process for product application shall be able to perform SPL product development. The extension of the MSPL has no necessity for special SPL tools and shall be ready to be integrated in most tool supported SPL.

References

[At00] Atkinson, C.; Bayer, J.; Laitenberger, O.; Zettel, J.: Component-Based Software Engineering: The KobrA Approach. In: 3rd ICSE Workshop on Component-Based Software Engineering. Reflection on Practice. Workshop Proceedings, Limerick, 2000.

[Ba99] Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J.: PuLSE™: A Methodology to Develop Software Product Lines. Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), (Los Angeles, CA, USA), May 1999, pp. 122-131.

[Bö04] Böckle, G.; Knauber, P.; Pohl, K.; Schmid, K.: Software-Produktlinien. Methoden, Einführung und Praxis. Dpunkt Verlag, 2004.

[CE00] Czarnecki, K.; Eisenecker, U.: Generative Programming. Addison-Wesley, Reading, MA, 2000.

[Ka90] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Spencer, P.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Carnegie Mellon University,1990.

[Ka98] Kang, K.; Kim, S.; Kim, K.; Kim, G., Shin, E.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. In: Annals of Software Engineering, Springer, Netherlands, 1998, pp. 143-168

[Lö04] Lönn, H. (ed.): Definition of language for automotive embedded electronic architecture. EAST-EEA Deliverable D3.6, public report, Version 1.02, 2004.

[MND02] Matinlassi, M.; Niemelä, E.; Dobrica, L.: Quality driven architecture design and quality analysis method. VTT Publications 456, Espoo, 2002

[Ob00] Obbink, H.; Müller, J.; America, P.; Ommering, R.: COPA - A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electronic Products. Tutorial, 1st Software Product Line Conference, Denver, 2000

[Pu08] Pure-Systems GmbH. WWW, http://www.pure-systems.com, 2008.

[So07] Softwaretechnik für das Automobil der Zukunft. WWW, http://www.itm.tum.de/mobilsoft/Startseite.htm, 2007

[Vo04] Voget, S. (ed.): Embedded Electronic Architecture – Glossary. EAST-EEA public report, Version 7.3, 2004.

[Wi06] Wild, D.; Fleischmann, A.; Hartmann, J.; Pfaller, C.; Rappl, M.; Rittmann, S.: An architecture-centric approach towards the construction of dependable automotive software. In: S. of Automotive Engineers, editor, Proceedings of the SAE 2006 World Congress, Detroit, 2006.

[WLT99] Weiss, D.; Lai, C.; Tau, R.: Software product-line engineering: a family-based software development process. Addison-Wesley, Reading, MA, 1999.

The Sales Scenario: A Model-Driven Software Product Line*

Henrik Lochmann and Birgit Grammel
SAP Research CEC Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany
{firstname.lastname}@sap.com

Abstract: The management of variability is essential to reduce efforts and increase efficiency in development, especially for large size software companies who offer complex product portfolios. Within the research projects FeasiPLe and AMPLE, we develop concepts and tools to facilitate an end-to-end process for Software Product Line Engineering (SPLE). Both projects emphasize different viewpoints of combining Model-Driven Development (MDD) and Aspect-Oriented Software Development (AOSD) with concepts of traditional SPLE to integrate the benefits of each of these technologies. In this paper, we present the case study we are developing in both projects - the model-driven software product line *Sales Scenario*. We introduce its application domain and explain our approach in development. Although our work is to date not finished, we are able to give first insights into the challenges of product line engineering with a combination of MDD, AOSD and SPLE from scratch.

1 Introduction

In this paper we present a model-driven software product line, which is used as case study in the two research projects FeasiPLe [Con07b] and AMPLE [Con07a], SAP Research is partner of. Our goal is to document the stepwise procedure of creating a software product line with a combination of model-driven, aspect-oriented and product line technologies. With this case study, we want to demonstrate the feasibility of such a combined approach, which, to the best of our knowledge, does not yet exist within the area of business applications. We followed a bottom-up approach, starting from a reference implementation and leading to feature-based decomposed system models. We want to disclose the challenges of such a process and to rate our development approach. The here presented case study will be used to evaluate the tools and concepts we are developing within the two research projects. Our vision is to provide an end-to-end tool chain for SPLE, from feature description to implementing code, with regards to traceabilty.

The paper is structured as follows: we begin with an introduction to the application domain of our product line in Sec. 2. An overview about the solution space is given in Sec. 3. Our approach to development and the challenges herein are documented in the sections 4 and 5. We conclude and give a short outlook in Sec. 6.

*This work is partially supported by the EC-funded project AMPLE and the German BMBF-funded project feasiPLe.

2 Problem Space

This section starts with an introduction to the domain of our case study. Followed by a description of its feature model, the core features are pointed out, stating their dependencies and interactions.

2.1 Introduction to the Domain

Our case study demonstrates business application engineering in the domain of enterprise software, a rather large domain. Exemplary solutions centered around Enterprise Resource Planning (ERP) include Product Life Cycle Management (PLM), Supply Chain Management (SCM), or Supplier Relationship Management (SRM). Such solutions must be adapted and customized to a particular company (no two companies are the same), business applications often have thousands of configuration settings. To reduce the complexity for the sake of conciseness, we focus on one specific sub-domain in this paper – Customer Relationship Management (CRM) – combined with some parts of the aforementioned solutions. Furthermore, the case study is not intended to be complete, it rather concentrates on those parts of CRM that seem to be relevant and meaningful enough to be integrated into a variability-driven context. Therefore, the presented example is called *Sales Scenario* to clearly distinguish from CRM as explained by Buck-Emden and Zencke in [BEZ04].

2.2 Feature Model

Main purpose of the Sales Scenario is the holistic management of business data, including central storage and access controlled retrieval. It focuses on product sales processes, where the core features comprise: *Customer Order Management*, *Payment*, *Account Management*, *Product Management* and *Communications*. These features are depicted in the case study's feature model in Fig.1, created with an Eclipse-based feature modeling tool developed by Christian Wende et al. from Dresden University of Technology, Germany. The feature notation relates to the original FODA notation [KCH+90] but is purely cardinality-based and abstains from the original, often confusing bubble syntax. In order to clarify the interplay between certain features and to give an understanding of what is covered in the Sales Scenario, the following description of the overall sales process and its individual components is presented. To better break down certain features, this process was clearly divided into separate steps.

Step 1 To start business with a potential customer, the corresponding master data, such as customer name and address, budget estimation and a description of the sales opportunity and its time frame is saved to a customer profile (*Prospect*) by using the functionalities of *Account Management*.

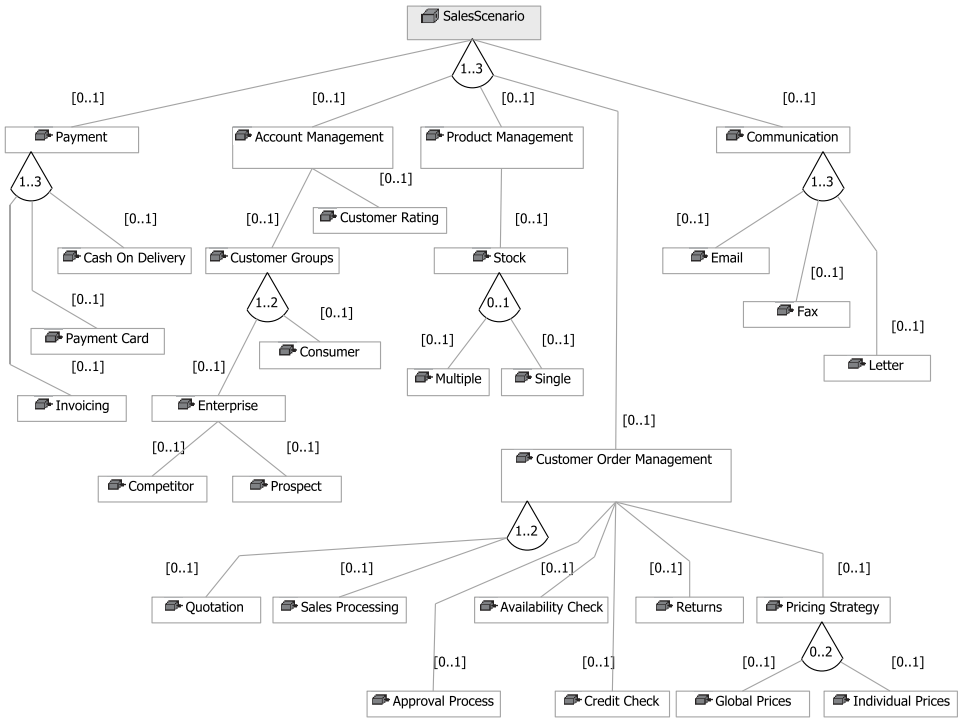


Figure 1: Feature Model of the Sales Scenario Product Line

Step 2 After subsequent evaluation and go/no go decision handling by sales management, another employee of the sales office creates a quotation (i.e., offer) using the *Quotation* functionalities and configures a quotation template based on the sales opportunity data. A potential discount is included into such a quotation by using *Pricing Strategy*. The calculation strategy is based on the categorization of *Prospects* in a *Customer Group*, estimated sales volume, and sales probability, resulting in an overall *Customer Rating*.

Step 3 After the sales office has contacted the customer and received an order, the system automatically converts the quotation into an order upon mouse click using the corresponding *Sales Processing* functionality.

Step 4 To check the creditworthiness of the customer, an (optional) *Credit Check* is performed before creating a quotation as well as before accepting a sales order, by interacting with the *Payment* module.

Step 5 An *Availability Check* is performed to check for sufficient *Stock* and necessary capacities in the warehouse. In case of *Multiple Stocks*, meaning, delivery processes and

storage of different goods depend on several warehouses, only those warehouses sufficiently close to the shipping address are included.

Step 6 If *Payment* is to be integrated into the sales process, it would be activated automatically upon creation of a binding sales order. Depending on the method of payment offered by the system and selected by the customer, at least one of the following feature options are realized: An automatic debit transfer from the customer's account can be triggered (*Payment Card*) or an invoicing document can be attached to the delivery to account for *Cash On Delivery*, or if payment is settled at a later stage, *Invoicing*.

Step 7 The order status is set to "sent" by an employee as soon as the order is delivered to the customer. Once an open invoice is paid by the customer, the order is marked as "paid" and "closed". In case an open or already "sent" sales order is returned by the customer, carried out by *Returns*, an *Approval Process* is triggered, which involves sales management processes for commitment.

The description of all *Communication* channels has been omitted, for the sake of clarity. Such instances are naturally related to mediation of quotations, invoices etc., whether sent by *e-Mail*, *Fax*, or *Letter*.

3 Solution Space

Following this paper's focus on the development process of model-driven SPLE, we decided to divide the explanation of our case study's solution space into two different parts. Thus, this section starts with an overview about the application's general architecture and applied technologies. In Sec. 4 a more detailed explanation to concrete solution artifacts is presented orthogonally to the development approach we followed to create them.

3.1 Architecture Overview

A descending node-by-node analysis of the feature model led to an initial architecture for the product line's solution space. The functional nature of the Sales Scenario features allowed the derivation of modular architecture blocks for main features. The interconnections between these blocks arose from the domain description and therein contained information about process-centric interdependencies. Additional parts, such as the user interface (UI), came from technical considerations and corresponding stakeholders.

Our product line's high-level architecture is depicted in Fig. 2 following the FMC notation [KGT06]. It outlines the key entities and their interactions according to the functionality described above. Prominent are well modularizable, coherent features on the one hand and wide spread, crosscutting features on the other. Exemplary for coherent features are *Stock Management* and *Payment Processing* components. Such components are to a certain

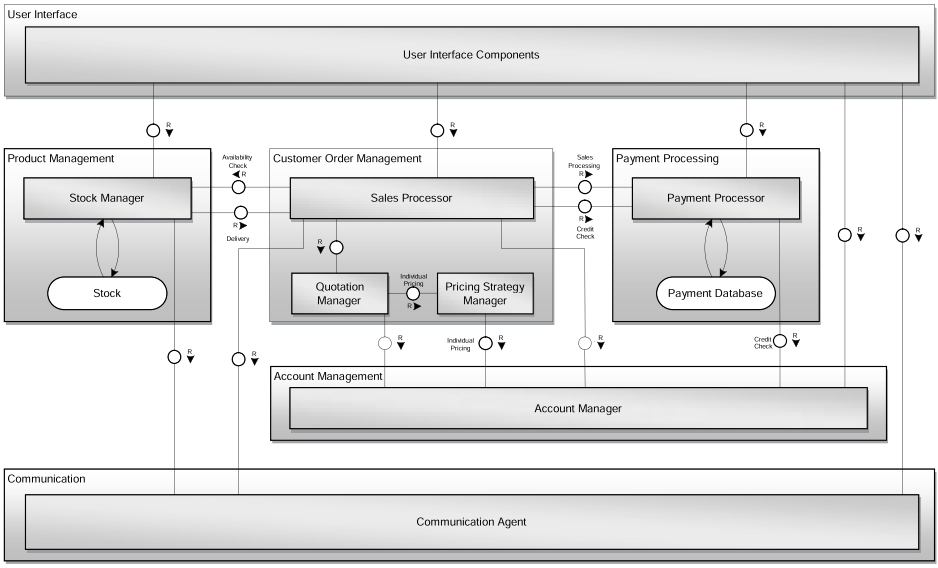


Figure 2: Sales Scenario Architecture Overview

degree self-contained and mainly expose their functionality for use by other components. The second feature type labels all those components that are scattered over a platform and interact actively with other components. Thus, such components encapsulate non-obvious cross-cutting concerns by widely distributing implementational or invokational parts of their functionality. A proper visual distinction between the two feature types and their corresponding architecture modules respectively is not given in Fig. 2. Nonetheless, one indication may be the number of communication connections of one block to others, where many connections prove cross-cutting functionality.

3.2 Platform Selection

Our software product line is a business application for product selling companies. Commonly, software applications in this domain are widely distributed, allowing multiple users and user groups to retrieve and modify different kinds of information. The distributed nature of business applications is a major technical constraint when searching for appropriate implementation means. In general, we had to make a selection between a Rich-Client Platform (RCP) with additional communication support and central knowledge database, and a web application platform. We decided to implement our product line as web application to minimize the engineering effort when following an RCP-based strategy, with regard to the various web application development frameworks, such as Spring [Spr07] or JBoss [JBo07]. A comprehensive state-of-the-art analysis revealed the JBoss Application Server

(AS) with its rich feature set and Eclipse IDE support as a promising solution. The JBoss AS provides an implementation of the J2EE component framework Enterprise JavaBeans (EJB). JBoss Seam [JBS07], the enhancing application framework for the JBoss AS, even increases the efficiency of this platform.

JBoss Seam bridges the gap between user interface and behavior implementations, i.e., Java Server Faces (JSF) and EJB session beans. This is done through a multi-layered session context stack managed with so-called *conversations* and helping templates on both sides. Additionally, the jBPM Process Definition Language (JPDL) allows seamless integration of navigation structures over dialog pages. For detailed information consider [JBS07].

4 Development Process

In this section we elaborate the approach we followed when designing and implementing the Sales Scenario product line, which was a bottom-up process. Thus, we decided to tackle the challenge by implementing one single product variant (cf. 4.1). This way we built a reference architecture for the modeling workbench we focus in our model-driven context (cf. Sec. 4.2). The last step in development of the Sales Scenario is the feature-based decomposition of the resulting system models. The current progress of this task is described in Sec. 4.3.

One might consider our development approach as *reverse engineering* process, which actually would not be correct. A reverse engineering process derives alternative, usually more abstract respresentations from an existing software system [CI90]. Here, this is only one part of the whole process. Before modelling our system (the actual reverse engineering), we have to identify, design and elaborate the necessary Domain-Specific Languages (DSL). Additionally, the system models we have to develop cover not only the initial product variant but also all other features of our product line. This way, the DSLs, their generators and according models constitute main assets of traditional domain engineering. Through a proper binding between features and model elements (cf. Sec. 4.3), we try to reduce the application or product engineering step to the choice of an appropriate variant from our feature model and adjacent adaptation steps of our domain models.

4.1 Implementing the Reference Architecture for a Single Variant

The initial exercise in our bottom-up development approach was to select a single product variant from our product line. The variant had to have preferably significant features, from a technical point of view, to provide an as complete as possible impression of the application's architecture. The chosen variant is depicted in Fig. 3. Because it does not anymore contain variable parts, only mandatory features are illustrated.

Based on the given feature selection, the domain description and architecture considerations, a reference implementation had to deliver a first set of realizations for the major parts

Figure 3: Variant for Reference Implementation

of our JBoss Seam web application. These parts are: data structures (EJB EntityBeans), UI screens (JSF pages), behavioural parts (EJB SessionBeans) and navigation models (JPDL page flows). Common analysis, design and implementation phases led to initial structures. We followed the concept that each coherent and for a business user meaningful feature, became a management component reflected in a *management page*, from the user interface point of view. Each management page lists corresponding data entities and provides access to appropriate functions. All management pages were aggregated in the so-called *control center*, another UI screen, which is also the entry point of the application. Figure 4 illustrates this concept on the basis of the *Quotation Management* feature.

The figure's upper left shows the management page for quotations. Here, all system-wide available quotations are listed, including their current lifecycle states and interaction possibilities. The figure's bottom-right part presents a step-by-step wizard to create quotation entities.

4.2 Deriving Modeling Means

The second task in our product line development process was the derivation of the necessary means to facilitate a purely model-driven approach. As modeling workbench we made use of the Eclipse Modeling Framework (EMF) [EMF07], mainly due to its rich feature set and lively supporting community. For code generation, we applied the generator framework openArchitectureWare (oAW) [OAW07], which integrates in EMF.

The creation of modeling means took place in four phases. *First*, we had to identify which DSLs are necessary to completely cover the ingredients of our web application and to facilitate the code generation into a JBoss Seam based platform. *Second*, the metamodels for the chosen languages had to be created. *Third*, appropriate generators based on the developed metamodels had to be prepared, mapping our system models to an executable

Figure 4: Sales Scenario Quotation Management UI Excerpt

EJB web application. *Fourth* and last, within the system modeling phase, we created the actual implementation of the Sales Scenario on model level.

In general, our focus lay clearly on the creation of platform independent models (PIM) [MDA03], to not hamper the ability of porting our system models to other platforms. The languages we identified are covering: data structures, data interaction, dialog navigation and presentation, stateful data contexts and behavior. Table 1 summarizes identified languages, indicating their names, domain and architecture parts for generated code.

4.3 Feature-Based Decomposition

This last step in our development of the Sales Scenario deals with the feature-based decomposition of the developed system models. Here, a mapping between problem and solution space has to be established. To date, we are not finished with this last task. However, we are able to explain what challenges we are facing here and which tools were already developed or could at least be applied to tackle these challenges.

To achieve a feature-based model decomposition, each feature has to be bound to all the model elements it corresponds to. Different problems are to be solved here: (1) the actual mapping of one feature to its corresponding model elements, which has to be done with

Language	Domain	JBoss (Seam) Platform Part
businessojects	data structures	EJB EntityBeans
actions	data interaction	persistence manager calls in EJB SessionBeans
concreteactions	refines actions for fine grained data structure interaction	EJB SessionBean method bodies
state	state machines for entity life cycles	EJB SessionBean method bodies and EntityBean attributes
dialog	dialog navigation	JPDL page flow definitions
presentation	dialog layout and events	JSF pages
context	stateful data contexts	JBoss Seam conversation annotations and navigation rules

Table 1: Sales Scenario Modeling Languages

(2) respect to the constraints between features (e.g., if feature A supplies model elements, feature B contributes to), and (3) the dependencies between separate technical models of different architecture parts have to be resolved. As an answer to (1) and (2), Heidenreich et al. developed an Eclipse-based tool in [HW07] within the research project FeasiPLe, which allows the graphical specification of such feature mapping templates. The resulting mapping specification, will then be used as template for appropriate technologies, such as Graph Rewrite Systems (GRS) or Aspect-oriented Modeling (AOM), that are used to integrate corresponding model fragments. In [HW07], the authors consider both collaborative and cross-cutting features with the help of the graph rewrite engine Tiger EMF [Tea07]. However, for the integration of model-crossing dependencies stated with (3), alternative approaches, such as initially proposed in [Loc07] and elaborated in [BL07], might be more appropriate.

5 Challenges in Development

In this section, we explain the main challenges we experienced within the development of our model-driven software product line. As our current key problems we pointed out the balancing of stakeholder interests, the development of modeling languages and the employment of *multiple* DSLs.

5.1 Stakeholder interests

Large software projects are always influenced by many factors, such as hard product requirements, subjective interests and technical constraints. Commonly, these factors are

represented by certain persons, who are involved in the software life cycle - the *stakeholders*. The challenge is, to balance all stakeholders' interests wisely, and not to end up in a tilted project situation, where requirements cannot be satisfied by applied techniques or obligatory milestones cannot be held.

Even though the here presented case study remains a research prototype and its development is more or less easy to manage - mainly because of one local development team - above mentioned stakeholder interests had their effect as well. Mainly the tension between the functional specification in the problem space and the capabilities of the solution space's technical platform led to a trade-off in different requirements. This was especially challenging, because of sequential dependencies between feature specification and platform decision.

What we learned from this challenge is to always consider all stakeholder's demands early in development and arrange a regular, short-term information exchange, as suggested by light weight project management methods or development processes, such as Scrum [Mur04] and Extreme Programming [Bec99] respectively.

5.2 Language Development

Today, to the best of our knowledge, literature lacks of concrete design patterns in MDD, which lead from loose domain descriptions to sophisticated language constructs. This proved, the development of modeling languages to be a difficult task. Besides the challenge to express semantically correct concepts and relationships of a domain, one major problem is to ensure that the created language covers all important use cases of the corresponding domain.

In this regard, our development approach of a model-driven product line validated itself as sensible and useful. Our single variant platform implementation led to an initial understanding of the different parts involved in the Sales Scenario's architecture, and how they correspond to each other. The chosen variant provided major use cases, which could be used to derive main language concepts. Additionally, the platform's own domain specific languages, which some of our metamodels conform to large extents, helped to refine and elaborate our results. One of the main experiences from this challenge is that two important issues in development of modeling languages are *use cases* and an *iterative refinement* procedure. Besides this, common problems in MDD, e.g., round-tripping between metamodel and generator templates, hampered our development progress.

5.3 Employment of Multiple Languages and Models

In general, there were two main approaches possible when creating the modeling means for our product line. The first one was to develop one single language and model accordingly for the whole architecture. This approach could have been well done with respect to a platform centric staging. However, one single language is much more difficult to han-

dle in both development, e.g., due to complex dependencies in generator templates, and application, with regard to maintainability.

We decided to follow the second approach, to split system modeling into multiple languages. This was also motivated (1) through the applied platform and (2) from a separation of concerns point of view. However, the architectural split into multiple modeling languages produces new challenges. When decoupling one solid architecture into different parts, dependencies arise between these parts. The challenge is then to resolve, describe and respect these dependencies during system modeling and generator specification. The literature mainly proposes so called weaving models as solution to such dependency problems. An advanced approach is proposed by Hessellund et al. [HCW07], who applies Prolog to check dependencies between models. An even more advanced approach was proposed in [Loc07] and [BL07]. Here, the use of a system modeling ontology in combination with description logic axioms was proposed to resolve dependencies and provide active guidance when modeling systems with multiple modeling languages.

Fortunately, the employed JBoss Seam platform alleviates this challenge to a certain degree. Here, most interconnections between architecture parts are managed on with String-based references. In JPDL page flows, for instance, concrete identifiers of available JSF view pages are applied for specification.

6 Conclusion and Future Work

Within this paper we introduced a software product line in the area of business applications. For the implementation, we apply different technologies, namely Model-Driven Development with multiple DSLs, Aspect-Oriented Software Development and Software Product Line Engineering. We explained our bottom-up approach to development of this SPL and highlighted the challenges we are facing in development. As stated above, our work is to date not finished. Accordingly, in future work we will report on the feature-based decomposition of the Sales Scenario system models. Additionally, an evaluation of the developed system models concerning their actual platform independence would be interesting. Therefore, the implementation of alternative platform generators by using the same system models had to be performed.

7 Acknowledgment

We would like to thank Patrick Wustmann, Christoph Pohl, Steffen Goebel and other reviewers for their valuable comments on previous versions of this paper.

References

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.

[BEZ04] Rüdiger Buck-Emden and Peter Zencke. *mySAP CRM: The Official Guidebook to SAP CRM 4.0*. Galileo Press, 2004.

[BL07] Matthias Bräuer and Henrik Lochmann. Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In *Proceedings of ATEM'07 co-located with MoDELS 2007, Nashville, Tennessee*, October 2007.

[CI90] E.J. Chikofsky and J.H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software Engineering Journal*, pages 13–17, January 1990.

[Con07a] AMPLE Consortium. AMPLE Research Project. `www.ample-project.net`, Dec 2007.

[Con07b] FeasiPLe Consortium. FeasiPLe Research Project. `http://feasiple.de`, Dec 2007.

[EMF07] Eclipse Modeling Framework (EMF) Project. `http://www.eclipse.org/modeling/emf`, 2007.

[HCW07] Anders Hessellund, Krzysztof Czarnecki, and Andrzej Wasowski. Guided Development with Multiple Domain-Specific Languages. In *Proceedings of MoDELS 2007, Nashville, TN, USA*, volume 4735 of *LNCS*. Springer, September/October 2007.

[HW07] Florian Heidenreich and Christian Wende. Bridging the Gap Between Features and Models. In *Proceedings of AOPLE workshop co-located to GPCE*, 2007.

[JBo07] JBoss Enterprise Application Platform. `http://www.jboss.com/products/platforms/application`, 2007.

[JBS07] JBoss Seam. `http://www.jboss.com/products/seam`, 2007.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University Pittsburgh, Pennsylvania 15213, 1990.

[KGT06] Andreas Knopfel, Bernhard Grone, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. John Wiley & Sons, May 2006.

[Loc07] Henrik Lochmann. Towards Connecting Application Parts for Reduced Effort in Feature Implementations. In *Proceedings of 2nd IFIP CEE-SET 2007 (WIP Track), Posen, Poland*, October 2007.

[MDA03] Object Management Group (OMG). *MDA Guide Version 1.0.1*, June 2003. OMG document omg/2003-06-01, `http://www.omg.org/docs/omg/03-06-01.pdf`.

[Mur04] Craig Murphy. Adaptive Project Management Using Scrum. *Methods & Tools*, 2004.

[OAW07] openArchitectureWare. `http://www.eclipse.org/gmt/oaw/`, 2007.

[Spr07] Spring Framework. `http://www.springframework.org/`, 2007.

[Tea07] Tiger EMF Transformation Project Team. Tiger EMF Transformation. http://tfs.cs.tu-berlin.de/emftrans/, Dec 2007.

Modellierung von Variabilität in der Geschäftsanalyse – eine industrielle Fallstudie

Thomas von der Maßen[1], Andreas Wübbeke[2]

[1]DC Application Development, arvato services,
An der Autobahn, 33310 Gütersloh
Thomas.vonderMassen@bertelsmann.de

[2]s-lab – Software Quality Lab, Universität Paderborn,
Warburger Str. 100, 33098 Paderborn
awuebbeke@s-lab.upb.de

Abstract: Der Umgang mit Variabilität ist eine zentrale Herausforderung bei der Einführung einer Software Produktlinie. Variabilität sollte dabei schon zu Beginn der Anforderungsanalyse besondere Beachtung finden. Dieser Beitrag beschreibt erste Schritte und die daraus resultierenden Herausforderungen beim Einsatz zweier bekannter Modellierungstechniken für Artefakte der Geschäftsanalyse im Rahmen einer industriellen Fallstudie.

1 Einleitung

Software Produktlinien (SPL), gemäß der Definition von [CN01], bieten für die industrielle Entwicklung von gleichartigen Softwareprodukten für eine bestimmte Domäne große Vorteile. Sie unterstützen die Produktentwicklung innerhalb einer Domäne, indem sie Maßnahmen zur Kostensenkung bzw. Beschleunigung der Entwicklung zum Beispiel durch verbesserte Wiederverwendbarkeit von Entwicklungsartefakten anbieten. Einführende Informationen zu SPL finden sich in [PBL05] und [CN01]. Im Kontext von SPL ist der Umgang mit Variabilität eine große Herausforderung.

Die in dieser Fallstudie vorgestellten Erfahrungen sind im Kontext der Einführung einer Softwareproduktlinien-Entwicklung bei arvato services entstanden. arvato services hat langjährige Erfahrung im Bereich der Dienstleistungen rund um die Domänen CRM, Fulfillment und Loyalty. DC als Geschäftseinheit von arvato services entwickelt dabei Software in enger Zusammenarbeit mit den jeweiligen Auftraggebern. In jeder Domäne entstehen individuelle Softwarelösungen, die aber gleichzeitig viele domänenübergreifende Gemeinsamkeiten besitzen und daher auch ein gemeinsames Vorgehensmodell für deren Entwicklung benötigen, um u. a. den flexiblen Einsatz von Mitarbeitern in unterschiedlichen Projekten garantieren zu können..Auf Basis der Projekte vergangener Jahre ist ein *Softwarekomponenten-Baukasten* entstanden, der eine Grundlage für neue Projekte bildet und auf der Ebene des Systementwurfs bzw. der Systemarchitektur sowie der Implementierung eingesetzt werden kann.

Dieser Baukasten enthält u.a. wiederverwendbare *Geschäftskomponenten*. Beim Einsatz einer Komponente in einem konkreten Entwicklungsprojekt wird diese dann durch Modifikation bzw. Erweiterung an die projektspezifischen Anforderungen angepasst. Aus Sicht des Systementwurfs und der Implementierung dient der Komponenten-Baukasten als Plattform im Sinne einer SPL.

Im Gegensatz dazu wird im Bereich der Geschäftsanalyse bisher dem Vorhandensein von projektübergreifenden, gemeinsamen Anforderungen und der davon abhängigen Betrachtung der individuellen Projektanteile unzureichend Aufmerksamkeit geschenkt. Somit ist die Wiederverwendung von Entwicklungsartefakten in der Geschäftsanalyse zu optimieren. Die in dieser Phase der Entwicklung entstehenden Entwicklungsartefakte, wie z. B. das Geschäftsmodell, bieten zum aktuellen Zeitpunkt keine Möglichkeit der expliziten Modellierung von Variabilität. An dieser Stelle würde ein generisches Modell (Plattform), welches für jedes Projekt abgeleitet werden kann, Vorteile hinsichtlich der Wiederverwendbarkeit und sinkende Spezifikationszeit bedeuten. Ableitung meint in diesem Kontext die Auflösung der im generischen Modell explizit modellierten Variabilität, d.h. die Bindung von variablen Geschäftsanforderungen an bestimmte Punkte des Modells. Nach der Ableitung des projektspezifischen Modells können dann weitere Geschäftsanforderungen angefügt werden, die nur im Kontext dieses Projekts relevant sind.

In diesem Beitrag wird anhand eines Praxisbeispiels diese Problematik der Behandlung von Variabilität in der Geschäftsanalyse erörtert und daraus Lösungsmöglichkeiten unter Einsatz zweier verschiedener Konzepte zur expliziten Modellierung von Variabilität skizziert. Die resultierenden Ergebnisse stellen dabei einen ersten Schritt zu einem ganzheitlichen und durchgängigen Ansatz zur Behandlung von Variabilität dar. Im folgenden Kapitel 2 wird unser Vorgehen bei der Geschäftsanalyse grundlegend dargestellt und anhand eines Beispiels die vorhandenen Unzulänglichkeiten bei der Behandlung von Variabilität und die daraus resultierenden Optimierungspotentiale dargestellt. In den Kapiteln 3 und 4 werden dann zwei etablierte Modellierungsansätze erläutert und auf ihr Potential zur Lösung der in Kapitel 2 identifizierten Unzulänglichkeiten hin untersucht. Abschließend werden die erarbeiteten Ergebnisse in Kapitel 5 zusammengefasst und ein Ausblick über diesen Beitrag hinaus gegeben.

2 Geschäftsanalyse bei DC

2.1 Konzepte und Vorgehensmodell

Der Entwicklungsprozess bei DC orientiert sich am Rational Unified Process (RUP) [Kr99]. Jedes Projekt beginnt mit einer sogenannten Akquisitionsphase. Diese dient der Aufwandsschätzung für das jeweilige Projekt, die darauf folgende eigentliche Geschäftsmodellierung (im RUP Geschäftsprozessmodellierung) dann der Ausformulierung des Geschäftsmodells. Die Akquisitionsphase unterteilt sich in die Teilbereiche Geschäftsanalyse, Systemanalyse (grob) und Bewertung. Die zentralen Artefakte der hier im Fokus stehenden Geschäftsanalyse sind die *Geschäftsskizze* und das *Geschäftsmodell,* bestehend aus allen *Geschäftsanwendungsfällen* (englisch Business Use Case, abgekürzt BUC) und deren *Geschäftsregeln*.

Die Geschäftsanwendungsfälle bilden in diesem Kontext die zugrundeliegenden Geschäftsprozesse ab. Die Geschäftsskizze dient als Grundlage für die Ausformulierung eines Geschäftsmodells und wird immer gemeinsam mit dem Auftraggeber erstellt. Dabei werden die BUCs nur namentlich beschrieben, eine Detaillierung findet dann im Geschäftsmodell statt. Dieses vollständige, konsistente und hinreichend spezifizierte Geschäftsmodell wird dann später in der eigentlichen Projektphase im Abschnitt Geschäftsmodellierung erstellt.

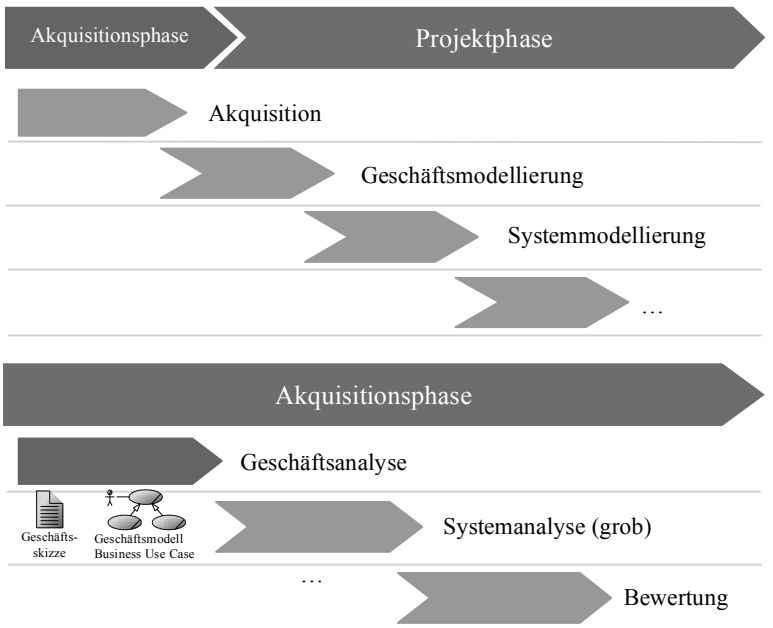


Abbildung 1: Vorgehensmodell

Der Prozess der Geschäftsanalyse sieht also zunächst das Erstellen einer Geschäftsskizze vor. Diese Skizze bietet thematisch gruppiert eine grobe Beschreibung der Geschäftsanforderungen (meist in Form einer Überschrift) an die zu realisierende Lösung. Die zu erstellende Geschäftsskizze wird aus einer Vorlage abgeleitet, welches bereits thematische Gruppen und ihnen zugeordnete Geschäftsanforderungen enthält. Diese Vorlage stellt die gesammelten Erfahrungen aus Geschäftsskizzen vergangener Projekte der gleichen Subdomäne dar. Bei der Ableitung einer Geschäftsskizze aus der Vorlage der Domäne wird folgendermaßen vorgegangen:

- Alle relevanten thematischen Gruppen und die jeweils in jeder Gruppe relevanten Geschäftsanforderungen müssen für das vorliegende Entwicklungsprojekt identifiziert und ausgewählt werden.

- Nach der Auswahl werden die noch fehlenden, projektspezifischen Gruppen bzw. Geschäftsanforderungen eingefügt.

Der nächste Schritt ist dann die Verfeinerung der grobgranularen Geschäftsskizze zu einem Geschäftsmodell. Dieses besteht aus Anwendungsfällen [OM04] (in diesem Fall den bereits erwähnten Geschäftsanwendungsfällen), die das zentrale und bei DC festgelegte Modellierungskonzept in allen Entwicklungsstufen darstellen. Es handelt sich dabei um eine natürlich sprachliche, textuelle Spezifikation eines Anwendungsfalls (vgl. [Ja92] und [Co01]). Diese Anwendungsfälle beschreiben Anwendungsszenarien durch Darstellung des Standardablaufs unter Berücksichtigung von Alternativen und Ausnahmen. Die Erstellung des Geschäftsmodells geschieht wiederum gemeinsam mit dem Auftraggeber des Entwicklungsprojekts, um die relevanten Geschäftsprozesse richtig auszugestalten.

Die in der Geschäftsmodellierung der Projektphase fertig gestellten BUCs des Geschäftsmodells werden im Rahmen der Systemmodellierung analysiert. Auf dieser Basis werden dann Systemanwendungsfälle (englisch System Use Cases, abgekürzt SUC) abgeleitet, die die Grundlage für den Systementwurf und den Systemtest bilden.

2.2 Beispiel einer Geschäftsskizze

Abbildung 2 zeigt anhand eines Beispiels, wie die einzelnen Entwicklungsartefakte zusammenhängen. *Antragsmanagement* und *Interessentenbetreuung* sind hierbei thematische Gruppen und die darunter angeordneten Punkte die Grobbeschreibung der jeweiligen Geschäftsanforderungen (*Automatisierte Antragsdatenverarbeitung*, *Manuelle Antragsdatenverarbeitung*, …). Unterhalb einer jeden Geschäftsanforderung werden weiterhin geforderte, übergreifende Geschäftsanforderungen aufgeführt. Übergreifend meint hier, dass die Anforderungen in mehreren anderen Geschäftsanforderungen benötigt werden.

Zur Geschäftsanforderung *Automatisierte Antragsdatenverarbeitung* gehören z. B. die Anforderungen *#Adressvalidierung* und *#Abgleich: Blacklist*. Die übergreifende Geschäftsanforderung *#Adressvalidierung* wird auch in der Anforderung *Manuelle Antragsdatenverarbeitung* verwendet.

An dieser Stelle lässt sich bereits erkennen, dass bei dieser Art der textuellen Modellierung die in den Geschäftsanforderungen geforderten übergreifenden Geschäftsanforderungen explizit in jeder Anforderung aufgelistet werden und somit redundant modelliert werden. Aus der Geschäftsskizze werden nun beispielhaft zwei BUCs für das Geschäftsmodell abgeleitet. Spezifizieren bedeutet in diesem Kontext das konstruieren von BUCs aus den existierenden Geschäftsanforderungen. Dies stellt einen kreativen Akt dar. Die BUCs in unserem Beispiel sind eine Spezifikation aus den beiden Geschäftsanforderungen *Automatisierte Antragsdatenverarbeitung* und *Manuelle Antragsdatenverarbeitung*. Diese Spezifikation muss nicht immer eine 1:1 Beziehung sein. Vielmehr sind einerseits eine Aufteilung einer Geschäftsanforderung in mehrere BUCs sowie die Konsolidierung mehrerer Geschäftsanforderung in einem BUC möglich und praktisch relevant (n:m Beziehung).

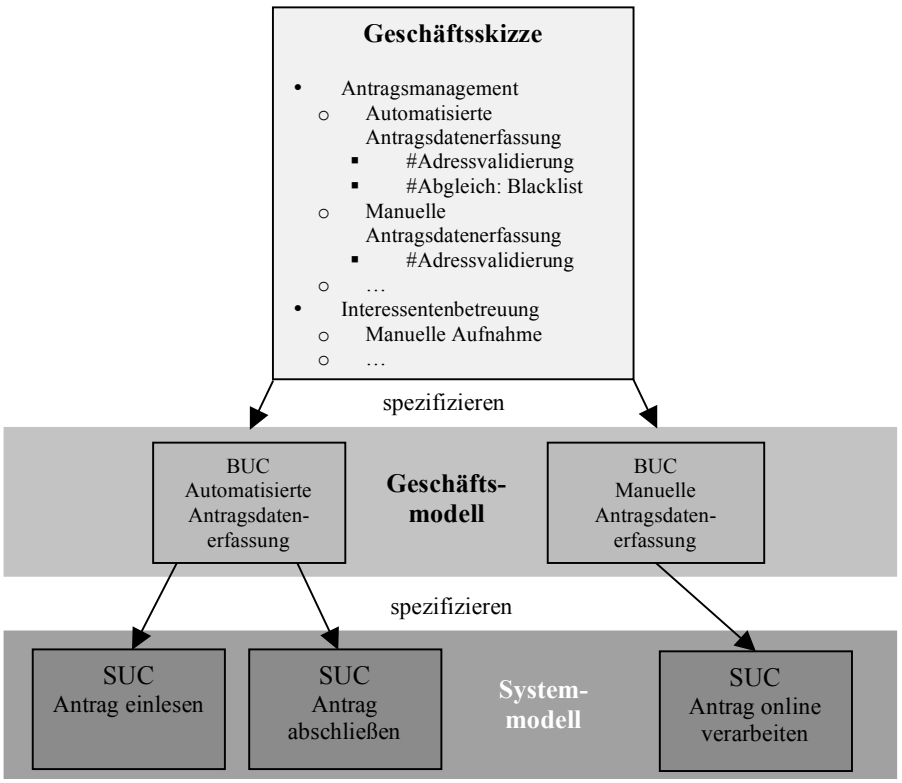


Abbildung 2: Entwicklungsartefakte in der Geschäftsanalyse und dem Anwendungsentwurf

Im letzten in Abbildung 2 gezeigten Schritt wird exemplarisch das Ableiten der Systemanwendungsfälle (englisch System Use Case, abgekürzt SUC) gezeigt. Auch zwischen BUCs und SUCs existiert eine n:m Beziehung.

2.3 Variabilität in der Geschäftsanalyse

Die explizite Modellierung von Variabilität ist eine der Grundlagen der erfolgreichen Geschäftsmodellierung einer Software Produktlinie. Die Forschung bietet verschiedene Ansätze, um dieser Herausforderung zu begegnen. Vor der Auswahl eines Vorgehens wird zunächst die in der vorgestellten Geschäftsanalyse implizit enthaltene Variabilität analysiert, um die Auswahl einer passenden Methodik zu unterstützen.

Die Variabilität in der hier vorgestellten Geschäftsanalyse wird in zwei verschiedenen Entwicklungsartefakten sichtbar. Zum einen werden in der Geschäftsskizze verschiedene übergreifende Anforderungen den einzelnen Geschäftsanforderungen zugeordnet. Diese Abhängigkeiten können in unterschiedlichen Projekten verschieden sein und sind daher projektübergreifend als variabel erkennbar.

Auch die verwendeten thematischen Gruppen und deren Geschäftsanforderungen sind abhängig von den Anforderungen des jeweiligen Projektes. Das zweite zentrale Artefakt, welches von der Variabilität betroffen ist, ist der BUC. In ihm werden bei der Modellierung je nach Projekt einerseits verschiedene Geschäftsanforderungen eingefügt und andererseits der Ablauf projektspezifisch modelliert. Weiterhin können BUCs untereinander referenziert werden, um die den verschiedenen Geschäftsanforderungen entsprechenden Prozesse darstellen zu können.

2.4 Resultierende Unzulänglichkeiten und Optimierungspotentiale

Bei der bisherigen Art der Modellierung einer Geschäftsskizze wird die in der Geschäftsskizzen-Vorlage implizit enthaltene Variabilität hinsichtlich der Geschäftsanforderungen und schließlich der möglichen, ableitbaren Geschäftsskizzen nicht explizit modelliert. Daraus resultieren folgende Probleme:

- Keine explizite Modellierung der Variabilität
- Die Abhängigkeiten einzelner Funktionalitäten untereinander werden nicht sichtbar, wodurch die Fehleranfälligkeit der Modellierung steigt
- Der geringe Grad an Formalisierung kann zu unterschiedlicher Granularität und schließlich auch Qualität bei der Modellierung einer Geschäftsskizze führen
- Die Ableitung einer Geschäftsskizze aus einer Vorlage bedeutet in jedem Projekt das Ableiten und Erstellen einer im Endeffekt vollständig eigenen Geschäftsskizze. Dadurch entsteht ein erhöhter Aufwand beim Ableiten jeder Skizze

Bei der nach der Geschäftsskizze folgenden Modellierung der BUCs werden die zuvor beschriebenen, übergreifenden Geschäftsanforderungen in die Prozessmodellierung des BUCs eingefügt. Hierbei entstehen projektspezifische BUCs. Zum aktuellen Zeitpunkt existiert hier keine explizite Modellierung der Variabilität in Bezug auf die übergreifenden Geschäftsanforderungen und den Ablauf des BUCs. Dadurch werden die Wiederverwendbarkeit der BUCs sowie die Austauschbarkeit der Geschäftsanforderungen stark eingeschränkt. Dies führt zu erhöhten Aufwänden.

Die bisher skizzierten Probleme in der Geschäftsanalyse sollen nun unter Einsatz von in der Literatur bekannten Techniken zur Modellierung von Variabilität und der Anwendungsfall basierten Entwicklung im Kontext von Software Produktlinien gelöst werden. Die Herausforderung bei der Modellierung der Variabilität innerhalb der Geschäftsskizze und der davon abhängigen BUCs besteht dabei in der ganzheitlichen Betrachtung der Variabilität in diesen beiden sehr unterschiedlichen Entwicklungsartefakten. Dabei beschreibt eine Geschäftsskizze eine statische Sicht auf die zu entwickelnde Lösung, wohingegen der BUC eine dynamische Sicht, nämlich die des Geschäftsprozesses, zeigt.

Auf Ebene der Geschäftsskizze wird eine Methodik benötigt, die sehr verschiedene Artefakte (thematische Gruppen und (übergreifende) Geschäftsanforderungen) behandeln und deren Beziehungen untereinander explizit machen kann. Es handelt sich dabei um eine statische Sicht auf das zu entwickelnde System. Dabei ist weiterhin zu beachten, dass nicht mehr für jedes einzelne Projekt eine vollkommen neue Geschäftsskizze erstellt werden soll, sondern eine Ableitung der Vorlagen-Skizze durch Bindung der Variabilität auf Ebene des Projekts erfolgt. Die Definition für Ableitung findet sich in [Ma07]. Auf Ebene der BUCs wird auch eine Methodik benötigt, die die Modellierung von Variabilität explizit ermöglicht. Dabei legen wir den Fokus auf die Variabilität in Bezug auf die Geschäftsanforderungen und vernachlässigen zunächst die Variabilität im Ablauf.

Aus den dargestellten sehr verschiedenen Anforderungen an die Modellierung von Variabilität in den beiden Entwicklungsartefakten, Geschäftsskizze und BUC, wird deutlich, dass jedem Artefakt mit einem eigenen Vorgehen für die Modellierung von Variabilität Rechnung getragen werden muss. Im Fall der Geschäftsskizze bietet die Literatur ein Featuremodell als semiformales Modellierungsinstrument an. Für die Modellierung der BUCs werden wir anschließend auf Produktlinien Anwendungsfälle (PLUC) eingehen. Beide Methoden sollen im Folgenden in unser Vorgehen integriert und anschließend kritisch betrachtet werden.

3 Featuremodellierung als Methodik für die Geschäftsskizze

Featuremodelle waren erstmals in der Domain-Engineering-Methode *Feature Oriented Domain Analysis (FODA)* [Ka90] als zentrales Element der Modellierungsaktivitäten für Software-Systeme zu finden. Sie bieten die Möglichkeit abgrenzbare, für den Auftraggeber relevante Eigenschaften eines Produkts und deren Abhängigkeiten (Variationspunkte) untereinander darzustellen. Diese als Feature bezeichneten Produkteigenschaften werden in Kernfeature und sogenannte optionale Feature unterteilt. Somit wird es möglich, variable (optionale) Produkteigenschaften explizit zu modellieren. Die Ableitung eines Featuremodells zu einem produktspezifischen Featuremodell geschieht durch Bindung von Features an Variationspunkte, wodurch die Variabilität aufgelöst wird [Ma07].

Das Featuremodell eignet sich für unsere Geschäftsskizzen-Modellierung aus folgenden Gründen:

- Es ist ein weit verbreitetes Konzept für die Modellierung von Variabilität
- Es besitzt verschiedene Variabilitätskonzepte (z.B. Auswahl, Alternativen, Implikationen, …)
- Es stellt die Abhängigkeiten zwischen den einzelnen Features dar
- Es bietet dem Auftraggeber die Möglichkeit zunächst statische Eigenschaften bzw. Funktionalitäten des späteren Systems auszuwählen

Wir verwenden deshalb ein Featuremodell zur stufenweisen Modellierung anhand unseres Beispiels Subdomäne Loyalitätsprogramme anstatt eines Modells auf Basis von Anwendungsfällen, da es zunächst nicht um die Modellierung dynamischer Abläufe geht. Im Featuremodell wird zunächst die Beziehung der einzelnen Geschäftsanforderungen zu der thematischen Gruppe modelliert. Bei dieser Modellierung werden, im Gegensatz zum bisherigen Vorgehen, die in den Geschäftsanforderungen geforderten übergreifenden Anforderungen unabhängig von den jeweiligen Geschäftsanforderungen konsolidiert modelliert. Dieses Vorgehen bietet folgende Vorteile:

- Es entstehen keine Redundanzen bei der Modellierung der übergreifenden Geschäftsanforderungen
- Die Abhängigkeiten der (übergreifenden) Geschäftsanforderungen untereinander können explizit modelliert werden

In Abbildung 3 wird ausschnittweise die thematische Gruppe *Antragsdatenmanagement* und zwei mögliche Geschäftsanforderungen dieser Gruppe dargestellt. Man kann hier erkennen, dass erst bei der Ableitung dieses Featuremodells in einem konkreten Projekt der Variationspunkt beim Feature *Antragsdatenmanagement* gebunden wird, in dem aus der Menge der zur Auswahl stehenden Geschäftsanforderungen gewählt wird. Die zuvor angesprochene Konsolidierung der übergreifenden Anforderungen geschieht im Feature *Antragsdatenprüfung*.

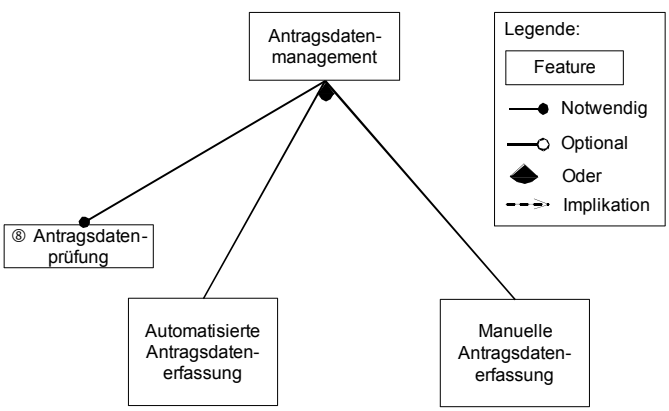


Abbildung 3: Featuremodell auf Ebene der thematischen Gruppierungen

Dieses Feature wird nun aus Platzgründen in Abbildung 4 dargestellt. Die Abbildung zeigt die übergreifenden Geschäftsanforderungen und ihre Abhängigkeiten untereinander. Die nun noch fehlenden Abhängigkeiten zwischen den Geschäftsanforderungen können einfach mit Hilfe von Implikationen modelliert werden (Beispiel in Abbildung 4: *Prüfung: Emaildubletten* impliziert *Prüfung: Email-Adresse)*.

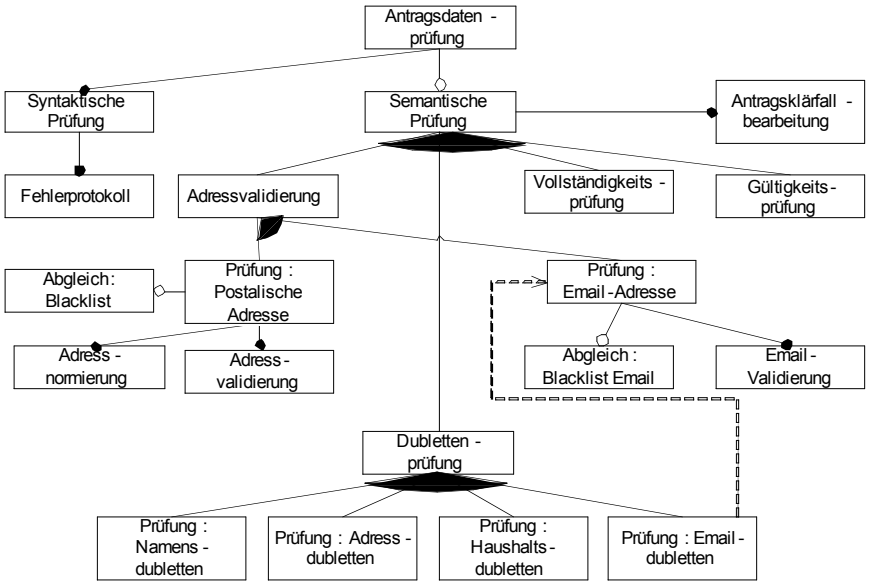


Abbildung 4: Featuremodell der übergreifenden Geschäftsanforderungen

Bei der hier vorgestellten Modellierung entsprechen die dargestellten Features Geschäftsanforderungen oder auch übergreifenden Geschäftsanforderungen. In Abschnitt 2.2 wurde bereits die existierende n:m Beziehung zwischen Geschäftsanforderungen und BUCs verdeutlicht. Auch innerhalb eines BUCs muss Variabilität explizit modelliert werden. Dabei handelt es sich nicht wie bisher in Bezug auf das Featuremodell um eine statische Sichtweise auf die Geschäftsanforderungen, sondern um eine dynamische Sicht auf die Geschäftsprozesse. Die Variabilität stellt sich hierbei in zwei Formen dar:

- Variabilität in der Ablaufbeschreibung
- Variabilität der Geschäftsanforderungen

Für die Modellierung von Variabilität innerhalb eines BUCs wird also ein eigenes Modell benötigt. Hier bietet die Literatur das Konzept der Product Line Use Cases (PLUC) [Be04] an, welches wir nun für die Modellierung der Geschäftsanforderungen in einem BUC verwenden werden.

4 Modellierung von BUCs mit der PLUC Methodik

Die Beschreibung von natürlich sprachlichen Anforderungen unter Berücksichtigung von Variabilität ist eine Aufgabe, die sich angemessen durch ein methodisches Vorgehen unterstützen lässt. Die Auswahl dieser Methode geschieht auf Basis der Tatsache, dass der Anwendungsfall als zentrales Modellierungselement in der Entwicklung bei DC vorgegeben ist.

In diesem Zusammenhang bietet sich die von Bertolino et. al. beschriebene Methodik des Produktlinien Anwendungsfalls (englisch Product Line Use Case, abgekürzt PLUC) [Be04] für eine Adaption auf den von uns beschriebenen Problembereich an. Im Folgenden wird nun ein kurzer Überblick über PLUCs gegeben.

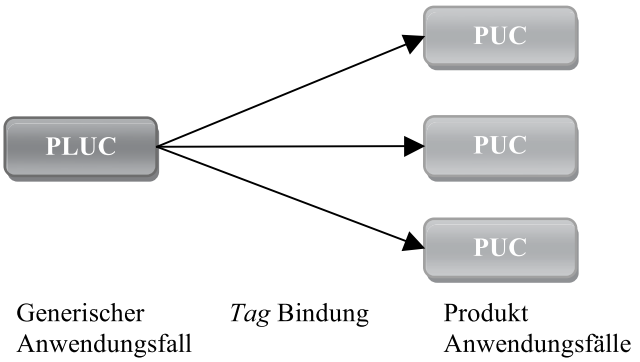


Abbildung 5: Zusammenhang PLUC und PUC

Die von Bertolino et. al. getätigte Erweiterung von Anwendungsfällen um Beschreibungsmöglichkeiten von Variabilität bietet mit *Tags* die Möglichkeit der Modellierung von Variabilität in der Form von Attributen. Diese Attribute stellen Wahlmöglichkeiten dar, die bei der Ableitung des PLUCs an ein Tag gebunden werden (Abbildung 5).

Name:		Automatisierte Antragsdatenerfassung
Vorbedingung:		Anträge liegen zur Verarbeitung bereit
Nachbedingung:		Anträge wurden in das System übernommen
Rollen:		IT-System
Erfolgreiches Szenario:		
Nr.	**Beteiligte Rollen**	**Beschreibung**
1.	IT-System	Antragsdateien einlesen
2.	IT-System	Antragsdateien auf syntaktische Korrektheit prüfen
3.	IT-System	Prüfung der **[ADR]**
4.	IT-System	Kundenvertrag erstellen
5.	IT-System	…
Variation:		
ADR: Parametrisiert [1..2]: • Prüfung: Postalische Adresse • Prüfung: Email-Adresse		

Abbildung 6: Abbildung der Variabilität mit der PLUC Methodik

Dabei entstehen produktspezifische Anwendungsfälle (englisch Product Use Cases, abgekürzt PUC). Unser BUC Beispiel *Automatisierte Antragsdatenerfassung* aus Abschnitt 4 wird nun in Abbildung 6 mit Hilfe der PLUC Methodik ausschnittsweise beschrieben. Dabei wird die Variabilität der beiden übergreifenden Geschäftsanforderungen *Prüfung: Postalische Adresse* und *Prüfung: Email-Adresse* berücksichtigt. Der Tag [ADR] zeigt hierbei die beiden Geschäftsanforderungen an, die bei der Ableitung dieses BUCs gebunden werden können. Dabei zeigt [1..2] an, dass mindestens ein und höchstens zwei, in diesem Fall also alle, Geschäftsanforderungen gebunden werden können. Mit diesem Vorgehen kann die Variabilität von Geschäftsanforderungen in Bezug auf PLUCs dargestellt werden.

5 Ergebnis und weitere Arbeit

In diesem Beitrag wurde exemplarisch gezeigt, dass Variabilität in der Geschäftsanalyse bei DC eine große Bedeutung hat. Erste Erfahrungen mit dem expliziten Umgang mit Variabilität wurden in einem Kundenprojekt gemacht. Dabei wurde zunächst die in der Geschäftsanalyse vorhandene Variabilität analysiert und die zum aktuellen Zeitpunkt existierenden Unzulänglichkeiten in diesem Kontext herausgestellt. Wichtig war dabei die Erkenntnis, dass der Variabilität in beiden Entwicklungsartefakten Geschäftsskizze (statische Sicht) und BUC (dynamische Sicht) durch zwei verschiedene Modelle Rechnung getragen werden muss. Dieser Beitrag unterscheidet sich dabei von [SP07] durch die Wahl des Anwendungsfalls zur Modellierung von Geschäftsprozessen im Vergleich zum dort gewählten BPMN Ansatz. Weiterhin wurden folgende Vorteile durch den Einsatz der neuen Modelle identifiziert:

- Das Featuremodell unterstützt die Qualität der Modellierung von statischer Variabilität und hilft bei der Vermeidung von Modellierungsfehlern, da die in der Geschäftsmodellierung vorhandene Variabilität explizit dargestellt wird
- Der Einsatz eines Featuremodells bei der Modellierung der Geschäftsskizze entfernt die Redundanzen in Bezug auf die übergreifenden Geschäftsanforderungen und stellt die Abhängigkeiten der einzelnen Geschäftsanforderungen explizit dar.
- Ein Featuremodell bietet darüber hinaus durch die Ableitung einen Mechanismus an, um für neue Produkte schneller Geschäftsskizzen abzuleiten. Dies stellt eine Zeitersparnis im Vergleich zum bisherigen Vorgehen dar.
- Die bei der Modellierung von BUCs existierende Variabilität wurde durch den Einsatz von PLUCs explizit dargestellt.
- PLUCs bieten durch die Ableitung in PUCs eine Zeitersparnis im Vergleich zum bisherigen Vorgehen.

Durch den Einsatz eines Featuremodells und den PLUCs entstehen offensichtlich Abhängigkeiten zwischen diesen beiden. Diese Abhängigkeiten führen zu neuen Fragestellungen:

- Die Wahl einer Geschäftsanforderung in einem BUC bedeutet auch eine Auswahl und Bindung eben dieser Anforderung auf Ebene des Featuremodells. Diese Abhängigkeit ist im Moment nicht explizit sichtbar bzw. modelliert und kann bei der Ableitung eines Produktes zu Problemen führen, wenn die Anforderung bei der Ableitung der Geschäftsskizze nicht gewählt wurde.
- Ein bisher unberücksichtigter Punkt ist die Modellierung von Variabilität in der Ablaufbeschreibung eines Anwendungsfalls (Abschnitt 3), welche nicht mit TAGs abzubilden ist.

Aus diesen Gründen wäre ein nächster Schritt die genaue Analyse der existierenden Abhängigkeiten zwischen Features und PLUCs und der Abbildung von Features auf PLUCs. Dabei ist die Betrachtung von statischer und dynamischer Variabilität von großer Bedeutung. In diesem Zusammenhang sollte auch grundlegend geklärt werden, welche Artefakte der Geschäftsanalyse wie auf das Artefakt Feature abgebildet werden sollen. Die in dieser Fallstudie getätigte Abbildung von Geschäftsanforderungen auf Features ist nur eine Möglichkeit.

Literaturverzeichnis

[Be04] Bertolino, A.; Fantechi, A.; Gnesi, S.; Lami, G.: Product Line Use Cases: Scenario-Based Specification and Testing of Requirements. In (Käkölä, T.; Ducnas, J. C.; Hrsg.): Software Product Lines: Research Issues in Engineering and Requirements, Springer Verlag, 2006.

[CN01] Clements, P.; Northrop, L.: Software Product Lines: Practices and Patterns, Addison-Wesley Professional, 2001.

[Co01] Cockburn, A.: Writing Effective Use Cases, Addition-Wesley, 2001.

[Ja92] Jacobson, I.; Christerson, M.; Jonsson, P.; Øvergaard, G.: Object-Oriented Software Engineering – A Use Case Driven Approach, Addision-Wesley, 2. Auflage, 1992.

[Ka90] Kang, K. C.; Cohen S.; Hess, J. A.; Novak, W. E.; Peterson, A. S.: Feature Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR21, 1990.

[Kr99] Kruchten, P.: Der Rational Unified Process - Eine Einführung, Addison-Wesley, 2. aufl. edition, 1999.

[OM04] OMG: UML 2.0 Superstrukture Specification, Verfügbar unter: http://www.omg.org, Januar 2008.

[PBL05] Pohl, K.; Böckle, G.; van der Linden, F. J.: Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

[SP07] Schnieders, A.; Puhlmann, F.: Variability Modeling and Product Derivation in E-Business Process Families. In: Technologies for Information Systems, Springer, 2007.

[Ma07] von der Maßen, T.: Feature-basierte Modellierung und Analyse von Variabilität in Produktlinienanforderungen, Dissertation RWTH Aachen, 2007.

Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme

Inhaltsverzeichnis

Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme: Bedeutung modellbasierter und formaler Ansätze für effiziente Entwicklung und Zertifizierung 299
Hardi Hungar, Erwin Reyzl

Die Zertifizierung des Modellierungswerkzeugs ASCET nach der IEC 6 15 08 303
Peter Dencker, Tilman Glötzner

Unified Modeling Style and Unified Process Architecture: Model-Driven Design and Model-Based Verification key to safety and cost-reduction 307
Ralf Fachet

Safety Critical Java erleichtert die Zertifizierung sicherheitskritischer Echtzeit-Anwendungen 311
Thomas Henties

Erfolgsfaktoren bei der Entwicklung der OBCE-WW 316
Thomas Buchberger

Modellsicht auf die Logik eines generischen Stellwerkssystems 319
Serhat Cakaloglu, Hans-Jürgen Nollau, Murat Sahingöz

Secure Coding als Bindeglied zwischen formalen Ansätzen um Safety in Infrastruktur-kritischen Systemen zu gewährleisten 323
Wolfgang Köppl, Gordon Rohrmair

Nachweis von Sicherheitseigenschaften in modellbasierten Entwurfsprozessen 329
Thomas Peikenkamp

Zertifizierung einer Sicherungskomponente mittels durchgängig formaler Modellierung 335
Udo Frese, Daniel Hausmann, Christoph Lüth, Holger Taubig, Dennis Walter

Positionsstatement: Zertifizierungsargumentationen mit mathematischer Präzision 339
Hardi Hungar

Positionsstatement: Über die Rolle von formalem Beweis und Entwicklung im Zertifizierungsprozeß 341
Christoph Lüth

Schwachpunkte und Grenzen gegenwärtiger Methoden 343
Theodor Tempelmeier

Ergebnisse des Workshops „Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme“ 345
Hardi Hungar, Erwin Reyzl

Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme: Bedeutung modellbasierter und formaler Ansätze für effiziente Entwicklung und Zertifizierung

Hardi Hungar[1], Erwin Reyzl[2]

[1] OFFIS, Bereich Sicherheitskritische Systeme,
Escherweg 2, 26121 Oldenburg
hungar@offis.de

[2] Siemens AG, Corporate Technology, Bereich Software&Engineering,
Otto-Hahn-Ring 6, 81730 München
erwin.reyzl@siemens.com

Abstract: Angesichts der zunehmenden Verwendung von Software bei der Realisierung sicherheitskritischer und hochverfügbarer Systeme wächst die Bedeutung der Nachweisbarkeit nicht-funktionaler Eigenschaften wie der Funktionssicherheit und der Zuverlässigkeit – sowohl bei der Entwicklung wie auch bei der Zertifizierung derartiger Systeme. Dem daraus resultierenden Anspruch an Nachvollziehbarkeit, an Rigorosität von Tests und anderen Prüfungen einerseits stehen innovative Trends der Software-Entwicklung wie modellbasierte Entwicklung, formale Ansätze für Design und Verifikation, Komponenten- und Plattformorientierung, Code-Generierung und Test-Automatisierung gegenüber, die ohne spezifische Adaptionen diesem Anspruch häufig nicht genügen. Hier viel versprechende Techniken auszuwählen, Einsatzperspektiven zu eruieren und dabei den Abgleich zwischen Forschungsrichtungen und industriellem Bedarf zu suchen soll Thema des Workshops sein.

1 Ausgangssituation

Software, insbesondere eingebettete Software, spielt eine zunehmend tragende Rolle im Umfeld sicherheitskritischer und hochverfügbarer Systeme. Traditionell beheimatet in der Luft- und Raumfahrt sowie dem militärischen Bereich, steigt die Bedeutung derartiger Systeme zunehmend auch im zivilen Bereich, sei es in der Verkehrstechnik, der Industrieautomatisierung, der Medizintechnik, der Kraftwerks- und Energietechnik [Lev95]. Die Erfüllung und der Nachweis nicht-funktionaler Eigenschaften wie Funktionssicherheit, der Zuverlässigkeit und Verfügbarkeit werden bestimmend in der Entwicklung, aber auch bei der Zertifizierung derartiger Systeme. Relevante Standards sind der allgemeine Standard IEC 61508, aber auch domänenspezifische Normen wie EN50128, DO178, ISO WD 26262 und FDA-Vorschriften (siehe etwa [BB+06]). Besonderheiten dabei, die sich in diesem Zusammenhang mit dem zunehmenden Softwareanteil ergeben, gilt das besondere Interesse dieses Workshops.

Kostentreiber liegen erwiesenermaßen in Aktivitäten zum Nachweis durch Test, Verifikation und Validierung, aber auch im wachsenden Dokumentationsbedarf zur Nachvollziehbarkeit von Änderungen und ihrer Auswirkungen. Effizienten Ansätze zur Verbesserung von Nachweisen und zur Vereinfachung der Nachvollziehbarkeit wird gesteigerte Bedeutung zukommen. Von speziellem Interesse sind einerseits bewährte Lösungen und Erfahrungen aus dem industriellen Umfeld, andererseits Ansätze aus der akademischen Forschung mit realer Anwendungsperspektive.

2 Trends im Software-Engineering

In der Software-Entwicklung haben sich in den letzten Jahren verschiedene Ansätze und Techniken weit verbreitet. Dazu gehören modellbasierte Entwicklung, formale Ansätze für Design und Verifikation, Komponenten- und Plattformorientierung, Code-Generierung und Test-Automatisierung [Lyu96]. Im folgenden sprechen wir einige dieser Ansätze kurz an und weisen jeweils auf hier relevante Fragestellungen hin.

2.1 Modellbasierte Entwicklung

Die Erstellung von Modellen in frühen Phasen der Entwicklung hilft, Anforderungen klar, konsistent und umfassend zu formulieren und Konzepte der Realisierung greif- und damit überprüfbar zu machen. Besonders, aber nicht nur, bei objektorientierter Umsetzung haben modellbasierte Methoden eine mittlerweile recht große Verbreitung erlangt – hier sei nur die viel verwendete UML [UML] genannt. Im Zusammenhang mit standardisierten Anforderungen an den Entwurfsprozess stellen sich verschiedene Fragen an die Belastbarkeit des Ansatzes, etwa:

1. Können die Spezifikationsmodelle auf ein ausreichend präzises Niveau gehoben werden, so dass sie textuelle Beschreibungen mit Vorteilen für spätere Validationsschritte ersetzen?

2. Lassen sich (modellbasiert) Lösungskonzepte in einem verbindlichen Sinn überprüfen, bzw., unter welchen Voraussetzungen wäre dies möglich?

2.2 Formale Methoden

Das Versprechen, durch Einsatz formaler Methoden zu sicheren und umfassenden Aussagen über das Verhalten von Entwicklungsartefakten zu gelangen, ist oft gemacht worden. Mittlerweile sind auch in der Praxis verschiedenste Ansätze aus diesem Bereich erfolgreich umgesetzt worden. Gleichwohl ist der Anwendungsbereich bisher noch schmal und oft lückenhaft, und durchgängige Anwendungen sind selten zu finden [HB+07]. Wir listen einige Beobachtungen auf:

1. Um den Erfordernissen einer Norm zu genügen, müssen in der Regel die Verifikationswerkzeuge sowie alle Übersetzer oder Codegeneratoren qualifiziert sein.

2. Eine vollständige formale Analyse (etwa Modelchecken) ist nach wie vor zumeist nur bei sehr kleinen Systemen möglich. An welcher Stelle lassen sich solche Werkzeuge heute erfolgreich einsetzen, oder welche Änderungen, beispielsweise an Systementwurfssprachen, würden einen weitergehenden Einsatz erlauben?

2.3 Testgenerierung

Heutzutage basieren die meisten Verifikationen und Validationen auf ausführlichen Tests, die nach mehreren Gesichtspunkten durchzuführen sind, etwa Tests auf Basis von Fehlererwartung und Fehlereinstreuung, Berücksichtigung von Grenzwertanalysen, strukturabhängige Tests etc. Um dem hohen Aufwand der Erstellung von Sätzen von Testmustern, welche den jeweiligen Erfordernissen genügen, abzuhelfen, können etwa Verfahren der automatischen Testgenerierung in Erwägung gezogen werden. Zu beachten sind dabei u.a.:

1. Grundlage der Testgenerierung muss eine formale Beschreibung des Testobjektes sein. Neben Programmen in späten Entwicklungsphasen kommen in früheren Phasen Modelle in Frage.

2. Das Erreichen der Anforderungen muss separat nachprüfbar sein, oder das Werkzeug zur Testgenerierung ist in dieser Hinsicht zu qualifizieren.

2.4 Codegenerierung

Der Einsatz automatischer Codegenerierung, bei der etwa aus Modellen direkt Zielcode gewonnen wird, entspricht der Einführung einer weiteren, höheren Ebene der Programmierung, für die sich Fragen der Codierungsregeln und Übersetzerkorrektheit neu stellen. Erstrebenswert wäre es, für Verifikation und andere Entwurfspflichten nicht den erzeugten Code, sondern die Quelle der Generierung heranziehen zu können. Das aber erfordert aber in der Regel die Ergänzung der etablierten Normen auf bisher nicht erfasste Bereiche.

2.5 Weitere Ansätze

Die oben angeführten Ansätze und Techniken stellen nur einen Ausschnitt aus der großen Menge an Methoden dar, welche für eine Anwendung in Betracht kommen. Viele relevanten Fragen, etwa der Plattformorientierung, der Wiederverwendung, des Konfigurations- und Änderungsmanagements, der Anforderungsnachverfolgung und so weiter haben wir nicht genauer angesprochen. Alle diese Aspekte müssen in der Praxis berücksichtigt werden, und auch auf diesen Gebieten stellen sich alte Fragestellungen neu und verlangen nach innovativen Lösungen.

3 Zusammenfassung

Im Spannungsfeld zwischen kosteneffizienter Entwicklung, steigender Systemkomplexität, bei gleichzeitig hohen Qualitätsanforderungen und fundierter Zertifizierung besteht Bedarf nach übergreifenden Methodiken, welche auf Techniken wie die oben genannten zurückgreifen und sie mit geeigneter Werkzeugunterstützung in durchgängigen Prozessketten integrieren. Verschiedene Anforderungsprofile, teils auf Grund anwendungsspezifischer Vorschriften (Bahn, Luftfahrt, Medizin etc.), teils wegen Unterschieden in Kritikalität oder Systemumgebung, werden in Zukunft eine vielgestaltige Landschaft an Lösungen bedingen. Bei aller Vielgestaltigkeit jedoch lässt sich voraussagen, dass allein schon die Notwendigkeit, zunehmend komplexere Lösungen noch wirtschaftlich tragbar beherrschen zu müssen, den integrierten Einsatz entwurfsunterstützender Technologie in großem Umfang bedingen wird. Zum Explorieren dieser Landschaft beizutragen ist zentrales Anliegen des Workshops.

Literaturverzeichnis

[UML] Object Management Group: Unified Modeling Language, Version 2.1.1:, http://www.uml.org/#UML2.0, 2007.

[BB+06] Braband, J.; Brehmke, B.-E.; Griebel, S.; Peters, H.; Suwe, K.-H.: Die CENELEC-Normen zur funktionalen Sicherheit. Eurailpress, Hamburg, 2006.

[HB+07] Hungar, H.; Bruhns, G.; Plan, O.; Lemke, O.: „OpRail – Normenkonforme Entwicklung sicherheitsrelevanter Software unter Einsatz der UML“, Signal + Draht 09/2007, S. 6ff.

[Lev95] Leveson, Nancy G.: Safeware, System Safety and Computers, Addison-Wesley Publishing Company, Inc, 1995.

[Lyu96] Lyu, Michael R.: Handbook of Software Reliability Engineering, IEEE Computer Society Press, Mc Graw-Hill Comp, Inc, 1996.

Die Zertifizierung des Modellierungswerkzeugs ASCET nach der IEC 6 15 08

Peter Dencker und Tilman Glötzner

ETAS GmbH
Borsigstrasse 14
70469 Stuttgart
Tilman.Gloetzner@etas.com

Abstract: Im Folgenden wird die Zertifizierung des Modellierungswerkzeugs ASCET nach der IEC61508 beschrieben. Nach einer Kurzbeschreibung von ASCET und der Norm wird die Motivation für eine Zertifizierung beleuchtet und ihr Nutzen für den Werkzeuganwender. Danach folgt eine Erklärung des Zertifizierungsprozess und der dabei entstehenden und zu prüfenden Dokumente.

1 Einleitung

Dieser Beitrag beschreibt die Zertifizierung des Modellierungstools ASCET nach der IEC 6 15 08 als „Fit for purpose“.

Im ersten Teil wird kurz die Funktion und der Aufbau von ASCET erläutert. Danach wird der IEC 6 15 08 und seine Zielsetzung beleuchtet. Der nächsten Teil geht auf die Motivation für die Zertifizierung eines Codegenerators ein. Dann wird die Zertifizierung für den Codegenerator an Hand des allgemeinen Vorgehensmodells erklärt. Der letzte Teil beinhaltet einen Ausblick auf die wachsende Bedeutung von Sicherheitsstandards.

2 ASCET

ASCET ist ein Modellierungswerkzeug, das sich an Funktionsentwickler und Softwareingenieure in der Automobilindustrie wendet. Es unterstützt einen iterativen Entwicklungszyklus nach dem V-Modell im Wesentlichen durch 5 Funktionen: Modellierung, Rapid Prototyping, Codegenerierung, Integration, Dokumentengenerierung. Codegenerierende Modellierungswerkzeuge verschieben die Funktionsentwicklungsaktivitäten von der Code-Ebene auf die abstrakte Modellebene. Sie erzwingen dabei einen Ablauf in einer Richtung, nämlich vom Modell zum Seriencode. Das Modell wird mittels Simulation und Rapid Prototyping immer tiefer überprüft und schrittweise verfeinert.

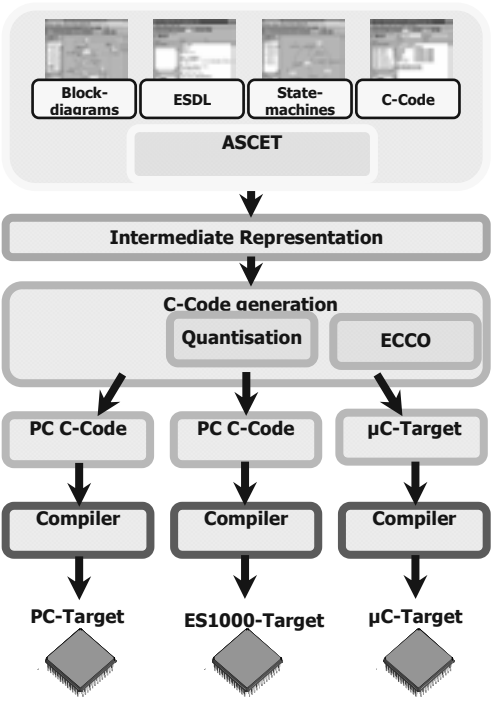


Abbildung 1: Funktionsdiagramm von ASCET

Die Effizienz der Softwareentwicklung steigt dadurch bei abnehmenden Entwicklungsrisiken und Entwicklungsaufwand.

Abbildung 1 zeigt den funktionalen Aufbau von ASCET. Die Modellkomponenten in den verschiedenen Notationen werden zuerst in eine Zwischendarstellung überführt, aus der nach einer Optimierung C-Code generiert wird. Soll das Modell in Fließkomma-Arithmetik auf dem PC simuliert werden, wird die im linken Ast dargestellte Compilerkette verwendet. Festkomma-Arithmetik wird mit der Quantisierungskomponente durch den mittleren Ast abgedeckt. Im rechten Ast erkennt man zusätzlich die Codegenerierung für die Zielplattform. Die mit ECCO bezeichnete Komponente führt während der C-Code-Generierung zusätzliche Optimierungen und Anpassungen für Zielplattform durch.

3 Die IEC 6 15 08

Die IEC 6 15 08 bietet eine systematische Herangehensweise an die funktionale Sicherheit von elektrischen, elektronischen und programmierbaren elektronischen Systemen in einem sicherheitsrelevantem Kontext an. Ausgehend von den Risiken, die durch Fehlfunktionen von einem System ausgehen können, fordert die Norm die Implementierung von Sicherheitsfunktionen. Um ein korrektes Funktionieren im Bedarfsfall zu gewährleisten, formuliert sie Methoden und Ziele im Lebens- und Entwicklungszyklus des Systems. Sie repräsentiert den derzeitigen Stand der Technik und des wirtschaftlich Sinnvollen.

Der „Safety Integrity Level (SIL)“ ist ein zentraler Begriff in der Norm. Er formuliert eine Anforderung an die funktionale Sicherheit eines Systems. Der SIL wird in 4 diskreten Stufen nach akzeptierten Auswirkungen von „Leichte Verletzungen einiger Personen, Sachschäden“ (SIL 1) bis „Tod vieler Personen (Katastrophe)“ (SIL 4) angegeben.

Erfüllt das Werkzeug und sein Entwicklungsprozess die Anforderungen des SIL n, so kann es nach „SIL n, fit for purpose“ zertifiziert werden. Damit ist das Werkzeug geeignet, Entwicklungen von Steuergeräten bis zu dem SIL n zu unterstützen.

Im Automobilbereich gibt es derzeit keinen branchenspezifischen Standard, so dass man direkt auf die IEC 6 15 08 zurückgreift. Typischer Weise kommen SIL2 oder SIL3 zu Anwendung. Für einen elektrischen Fensterheber wird SIL 2 ausreichend sein. Für ein ESP-Steuergerät, das aktiv in die Querdynamik des Fahrzeugs eingreifen kann, wird man SIL 3 erreichen wollen.

4 Motivation für eine Zertifizierung

Eine Zertifizierung nach IEC 6 15 08 wird häufig als Maßnahme zur Risikominimierung von Haftungsfällen angestrebt. Klagen aufgrund von **grober Fahrlässigkeit** gegen einzelne Personen werden sich eigentlich immer ausschließen lassen. In **Produkthaftungsfällen** wird das Zertifikat in der Regel die Nachweispflicht umkehren können, d.h. der Kläger (und nicht der Beklagte) wird nun nachweisen müssen, dass die Unternehmung ihrer Sorgfaltspflicht bei der Produktentwicklung nicht nachgekommen ist. Die Wahrscheinlichkeit eines Haftungsfalls und die dafür gebildeten Rückstellungen können verringert werden.

Zusätzlich kann man bei Steuergerätezertifizierung durch Verwendung eines zertifizierten Codegenerators auf zeitintensive, fehleranfällige und teure manuelle Reviews des aus dem Modell generierten Quellcodes verzichten. Die Reviews werden auf die abstraktere und übersichtlichere Modellebene verschoben. Auch Modultest, statische Codeanalyse und formale Reviews des autogenerierten Codes werden überflüssig. Bei großen Projekten mit viel autogeneriertem Quellcode verringert sich der Aufwand drastisch.

5 Die Zertifizierung

Die letzte Zertifizierung von ASCET war die dritte ihrer Art. Man verständigte sich daher auf eine „Delta-Zertifizierung“, bei der man sich auf Änderungen seit der letzten Prüfung durch den TÜV Nord konzentrierte. Von der ersten Anfrage bis zum Zertifikat vergingen fast 15 Monate. Der Projektverlauf lässt sich in 6 Abschnitte teilen:

1. Definitionsphase

   In diese Phase wird der Umfang der Zertifizierung mit TÜV Nord als zertifizierende Organisation abgestimmt. Im Fall von ASCET bestimmten ETAS und der TÜV Nord dabei die Transformation des Modells zum C-Code, d.h. den Codegenerator, als sicherheitsrelevante Funktion. Ausserdem verständigte man sich auf eine bestimme Konfiguration bzw. Version von ASCET. Auch der grobe Zeitrahmen des Zertifizierungsprojektes wurde festgesetzt.

2. Vorbereiten der Zertifizierung

   In dieser Phase wurden ein Zertifizierungsplan (Projektplan), ein Prüfplan und ein Bewertungsplan erarbeitet. Der **Prüfplan** listet die Anforderungen der

Norm, gegen die ASCET und sein Entwicklungsprozess geprüft wurden. Der **Bewertungsplan** hingegen beschreibt einen Maßstab, mit dem der Erfüllungsgrad der Anforderungen bestimmt wurde. Die Erfüllungsgrade der individuellen Anforderungen fließen gewichtet in ein Endergebnis ein, das über den Erfolg der Zertifizierung entscheidet. Auch das ist im Bewertungsplan dokumentiert.

3. Erstellen von Dokumentation

   In diesem Arbeitspaket wurde die für ASCET vorhandene Produkt- und Prozessbeschreibungen sowie Prozessartefakte, wie z.B. Testresultate, mit den Anforderungen aus IEC61508 in Beziehung gesetzt, um ihre Erfüllung zu begründen. Dieses war die aufwändigste Phase des Zerifizierungsprojekts.

4. Prüfung bzw. Audit

   Der TÜV Nord sichtete die übergeben Dokumentation, und führte mehrere Audits vor Ort durch. Bei diesen wurden Detailfragen, die sich aus den Dokumenten ergaben, geklärt, und Interviews mit Entwicklern und Produktmanagern durchgeführt.

5. Auswertung der Prüfergebnisse

   Der TÜV Nord wertete die Prüfergebnisse mittels des zuvor aufgestellten Bewertungsplans aus. Basierenden darauf entschied sich der TÜV für eine Zertifzierung.

6. Projektabschluss

   Mit Ausstellung und Übergabe des Zertikates wurde das Projekt abgeschlossen.

6 Ausblick

Die Anwendung der IEC 6 15 08 breitet sich derzeit schnell aus. Mehr und mehr Sicherheitsfunktion finden ihren Weg in das Auto und werden durch Steuergeräte implementiert. Das verstärkt die Notwendigkeit systematischer und risikobasierter Ansätze, wie sie die IEC 6 15 08 anbietet. Die Norm und der daraus abgeleitete branchenspezifische Standard ISO WD 26262, der gegenwärtig entwickelt wird, dürften in den nächsten Jahren in der Automobilindustrie stark an Bedeutung gewinnen.

Die IEC 6 15 08 hat viel mit den qualitäts- und prozessorientierten Standards wie CMMI oder ISO 9001 gemein. Konformität zur IEC 6 15 08 sollte deswegen nicht als zusätzlicher Prozessaufwand gesehen werden. Sie ist vielmehr eine Maßnahme, die Qualität und die Effizienz der Entwicklung und des Produktes selbst nachhaltig zu verbessern. Zertifizierte Entwicklungswerkzeuge unterstützen diese Ziele durch deutliche Verringerung des Zertifizierungsaufwands beim Automobilhersteller.

Unified Modeling Style and Unified Process Architecture: Model-Driven Design and Model-Based Verification key to safety and cost-reduction

Ralf Fachet

Esterel Technologies GmbH
Otto-Hahn-Straße 13b
85521 Ottobrunn-Riemerling
ralf.fachet@esterel-technologies.com

Abstract: Safety-processes and standards such as IEC61508 and ISO26262 are increasingly dominating the world of Software Development, and becoming a serious challenge for traditional approaches. The broad deployment of diverse methods and toolsets calls for a unified approach to design, process and verification. A proprietary approach with a "one size fits all" single vendor philosophy will not last into the future.

1 Introduction

Stringent requirements from safety standards such as DO-178B for aerospace, EN50128 for railways, ISO (WD) 26262 for automotive, and the meta-standard IEC61508 require specific attention and measures.

We are going to present an approach, which provides seamless integration across tools and methodologies using a meta- model/ open interface approach, a unified modelling style to transparently integrate data- flow and control- flow algorithms on model level, and a unified process approach to fully cover the requirements and objectives from safety and documentation standards.

2 Metamodel approach

In theory and science, Metamodeling has been around for a long time. The basic idea is to have a model of models which can be transformed into each other using model-transformation techniques. A Metamodel can provide several views and can also be used as a repository to extract information. In the past UML models where typical examples of metamodeling. Recent research goes into the direction to open this approach to other methodologies as well.

The tool suite SCADE (Safety Critical Application Development Environment) from Esterel Technologies has been developed with safety standard objectives as main design factor. From the beginning it was built around a UML metamodel approach. Model transformation interfaces based on open and documented interface provide the capability for model transformation from other metamodel standards (such as UML) but also even a means to translate proprietary models like Matlab/Simulink® into a component in a model of models by transforming them for example into a SCADE model.

The advantage of a model of models is seamless integration of the different phases of a project as well as the various subsystems of a complex programmable electronic system (such as hardware, systems software and application software). For example as SysML Framework may be used to depict the overall systems design. A SCADE model inside the framework may describe the application layer of the system and be used as base for automatic code generation and production of all certification collaterals.

3 Unified modelling style

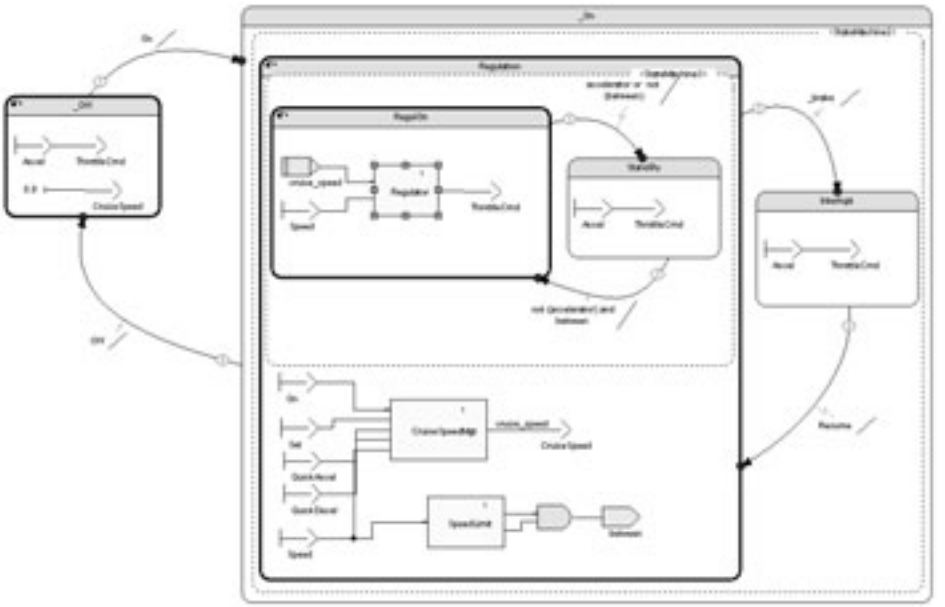
Model- based software engineering has evolved from traditional CASE tools to model-driven design methodologies. Since executable specifications have proven to be crucial for successful model- based design, pure documentation CASE tools have disappeared from the market.

Two tracks of modelling tools are widely deployed:

- Structure and analysis oriented tools, like the typical UML and SysML world
- Data-Flow oriented tools. Typically you see boxes that contain functionality and lines where the data flow between the boxes. Semantics might however vary widely.
- Control- Flow oriented tools: Typically you see state machines and event-triggered transitions from one state to the next.

While some of the existing tools require that the user moves to other methods in order to produce the software based on the analysis and design done in his tool (the other method often being handcoding), others provide means of autocoding. Some of the tools are either purely dataflow or control flow oriented, while others allow state-machine functionality to be plugged in by means of a separate tool which is coupled with the dataflow modelling environment.

SCADE now provides the possibility to seamlessly mix and stack dataflow and statemachines, and to integrate such a model into a system model of models so that for the first time the most complex applications can be fully expressed in a model- driven design flow.



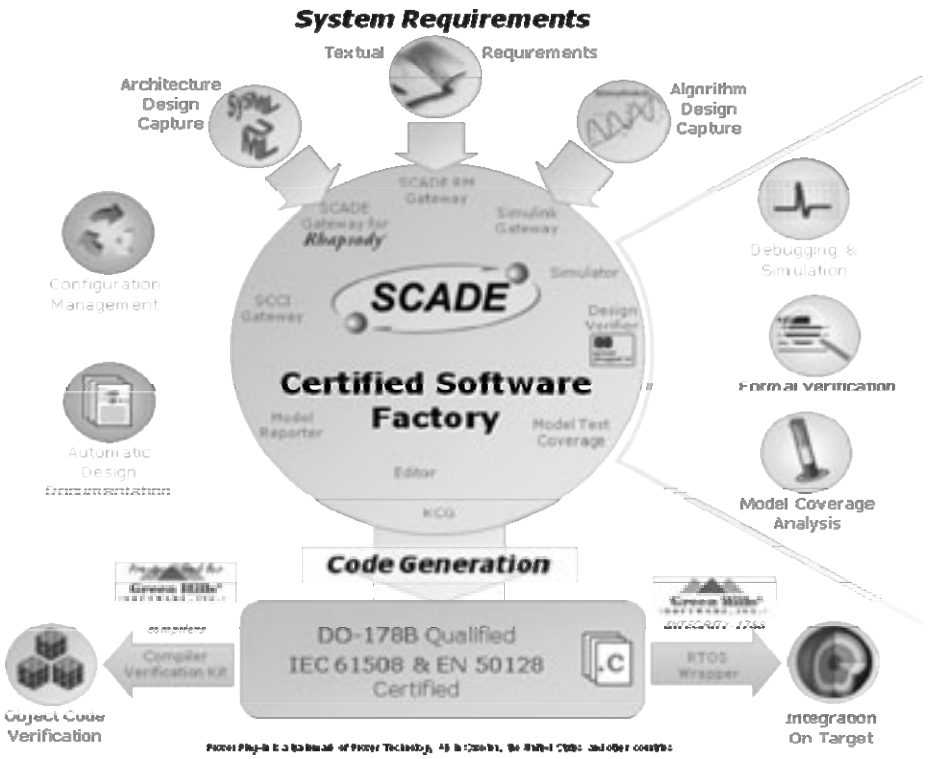
The dataflow and state- machines are not coupled on tool level but expressed in a strictly formalized modelling language. This formal model is then the source for automatic code generation and prerequisite for qualification of this automatic transformation step. The automatic code generator KCG 6 is therefore qualifiable to DO178B Level A, IEC 61508 and EN50128 as well as to all derived standards. The certification means that most code-level verification activities such as testing and reviews may be omitted in a safety critical project.

4 Unified process architecture

SCADE provides an open framework which serves as the backbone of as well the software development process as well as the validation and certification process. This open framework can be seen as the certification and software factory.

It takes in information from meta- models or functional prototypes and produces not only the certifiable software item but also manages the full certification workflow from system requirements to functional prototypes, algorithms and architecture done to the target- integrated application

Documentation, verification and validation are seamlessly connected as well as software development IDE, may the be vendor specific or open and eclipse- based.



As we see, this factory automates all steps that are necessary to produce safe software and also provide the safety proof. It follows the structure of the design, verification, validation, and certification processes but also provides structure and a efficient workflow to them.

The user can now ensure seamless interfacing to his system model, fully describe the needed functionality and model level and work in a fully integrated design, v&v and certification workbench.

SCADE provides the backbone and the open interfaces, enabling to use the right tools at the right place and allowing him to integrate his existing production islands into an efficient production cluster.

Safety Critical Java erleichtert die Zertifizierung sicherheitskritischer Echtzeit-Anwendungen

Thomas Henties

CT SE 2
Siemens AG
Otto-Hahn-Ring 6
81730 München
thomas.henties@siemens.com

Abstract: Der Vortrag informiert über den aktuellen Stand des *Java Community Prozesses* für die beiden *Java Specification Requests*:

- JSR-282 (*Real-Time Specification for Java 1.1*) und
- JSR-302 (*Safty Critical Specification for Java*).

Java war von Anfang an auch für *Embedded-Devices* gedacht, was auch die frühe Spezifikation von Echtzeit-Java (JSR-001) zeigt. Inzwischen werden von einigen Firmen *Java Virtuelle Maschinen* (JVMs) angeboten, die die *Real Time Specification for Java 1.0* (RTSJ) implementieren. Im Laufe des Jahres 2008 wird eine aktualisierte und verbesserte Version der RTSJ verabschiedet werden. Außerdem wird die erste Spezifikation für sicherheitsrelevante Java-Anwendungen (SCSJ) der Öffentlichkeit vorgestellt werden, die auf der neuen *Real-Time Spezifikation* basiert. Dabei wird auf alles verzichtet, was die statischen Analyse-Möglichkeiten zu sehr einschränkt. Oberstes Ziel ist es, die Zertifizierung der Java Applikationen (z.B. nach *DO 178B level A*) zu ermöglichen. Im Folgenden wird anhand der Anforderungen an sicherheitsrelevante Programme gezeigt, wie RTSJ modifiziert wird, um den gewünschten Anforderung gerecht zu werden.

1 Einleitung

Über 90 Prozent aller Mikroprozessoren werden für Echtzeit-Anwendungen eingesetzt, die meist mit ihrer Umgebung interagieren [Cor04]. Solche Systeme müssen nicht nur mit beschränkten Ressourcen realisiert werden, sondern außerdem zuverlässig und vorhersehbar funktionieren.

Zudem werden die funktionalen Anforderungen immer komplexer bei gleichzeitiger Verkürzung der Entwicklungszeiten. Deshalb ist es wichtig, dass die Vorteile von High-Level-Programmiersprachen, deren Methoden und deren Entwicklungswerkzeuge auch für Embedded- und Echtzeit-Anwendungen genutzt werden, um bessere Qualität und höhere Produktivität zu erreichen.

Java hat sich in Desktop- und Serversystemen durchgesetzt, weil es viele unnötige Schwierigkeiten, wie z.B. Typ- und Speicherfehler vermeidet und ein hohes Maß an Sicherheit und Produktivität bietet. Diese Attraktivität hat dazu geführt, dass "abgespeckte" Versionen (*Java Micro Edition*) weite Verbreitung in *Embedded Systemen* wie PDAs, Handys oder Set-Top-Boxen gefunden haben.

Trotz dieser guten Eigenschaften ist Standard-Java für Echtzeit-Anwendungen ungeeignet. Dies liegt hauptsächlich daran, dass Java zwar *Threads* und Synchronisierung unterstützt, das *Scheduling* aber weder genormt noch durch die Applikation beeinflussbar ist. Außerdem verhindert die *Garbage-Collection* und ein zu grob getakteter Timer den Einsatz von Java [Sto07]. Diese Probleme werden durch die *Real-Time Specification for Java* [RTSJ] gelöst. RTSJ hat sich inzwischen unter anderem in Anwendungen der Automatisierungstechnik, der Luft- und Raumfahrt [Bak06], aber auch bei Finanzapplikationen bewährt. Deshalb möchte man diese Technologie jetzt auch für sicherheitskritische Systeme einsetzen. Die Definition der SCSJ [JSR302] und neue Analysewerkzeuge machen die Verwendung von Java in sicherheitskritischen Systemen attraktiv.

2 Der Java Community Process

Der *Java Community Process* (JCP) ermöglicht die Beteiligung von Personen und Organisationen an der Definition von Funktionen und Eigenschaften der Java Plattform. Außer der Spezifikation muss auch eine *Reference Implementation* (RI) und ein *Technology Compatibility Kit* (TCK) verpflichtend erstellt werden. Die *Reference Implementation* (RI) zeigt, dass die Spezifikation auch wirklich implementiert werden kann. Das TCK ist eine *Testsuite*, die alle Anforderungen der Spezifikation vollständig überprüft. Jede Implementierung eines *JSRs* muss alle *TCK-Tests* bestehen, um ihre Spezifikationstreue nachzuweisen. Dadurch wird die Kompatibilität verschiedener Implementierungen gewährleistet.

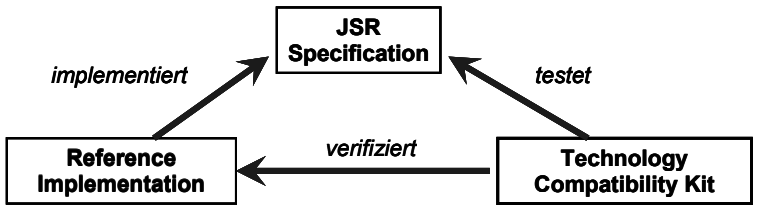


Abbildung 1: Komponenten eines *Java Specification Requests*

Für den JSR-282 (RTSJ 1.1) stehen inzwischen Vorabversionen der Spezifikation und der RI zur Verfügung. Die *Final Specification* wird im Laufe des Jahres 2008 verabschiedet werden. Für den JSR-302 (SCSJ) wird zur Zeit die erste Version der Spezifikation in der *JSR-Expert-Group* diskutiert. Mit der *Final Specification* ist erst nach der Abstimmung über den JSR-282 zu rechnen. Für den JSR-001 (RTSJ 1.0) stehen inzwischen verschiedene *Java VMs* zur Verfügung, unter anderem von Sun, IBM, Aonix und von der deutschen Firma Aicas. Alle genannten Anbieter wollen zukünftig auch JSR-282 und JSR-302 kompatible *VMs* anbieten.

3 Begriffsklärung

In einem Echtzeitsystem ist es erforderlich, dass – neben der Berechnung korrekter Ergebnisse – diese Ergebnisse rechtzeitig, also weder zu früh noch zu spät zur Verfügung gestellt werden. Anhand der Folgen, die eine Nichteinhaltung der Zeitschranken auslösen kann, werden Echtzeitsysteme in drei Kategorien unterteilt: Man spricht von:
soft real time wenn eine unpünktliche Berechnung lediglich zum Qualitätsverlust der Anwendung führt. Beispiele hierfür sind: Videoverschlüsselung und Telefonvermittlung.
hard real time wenn eine unpünktliche Berechnung zu einer Störung, einem Schaden bzw. zu finanziellem Verlust führt. Beispiele hierfür sind: Steuerung von Fahrzeugfunktionen wie z.B. Bremsen, Roboter-Steuerung oder auch Aktienhandels-Programme.
safety critical wenn eine unpünktliche Berechnung dazu führen kann, dass Menschenleben gefährdet sind. Beispiele hierfür findet man in den Bereichen: Luft- und Raumfahrt, Militär, Medizin und Kraftwerksteuerung.

Die *Radio Technical Commission for Aeronautics* hat bereits im Jahr 1992 Richtlinien für die Entwicklung vom Software für die Luftfahrt in den USA definiert. Diese tragen die Bezeichnung **DO-178B** [Do178] und sind in sogenannte *Levels*: A bis E unterteilt; wobei *Level A* die höchsten Anforderungen beinhaltet. Der Erfolg von DO-178B hat sich auch in anderen Branchen und Ländern fortgesetzt und ist die Grundlage zahlreicher Zertifizierungsprozesse.

Es gibt eine ganze Reihe von Werkzeugen, die bestimmte Eigenschaften von Java-Programmen, bzw. *Java class-files*, statisch analysieren. Die SCSJ geht davon aus, dass solche Tools existieren - sie sind jedoch nicht Teil der SCSJ, da hier Wettbewerb und Weiterentwicklung erwünscht ist. Datenflussanalyse, *Model-Checking* und Deduktive Verifikation sind Kandidaten zum Nachweis von *Post-Conditions*, Erreichbarkeit oder *Worst-Case-Execution-Time*. So kann beispielsweise durch eine Analyse leicht überprüft werden, ob die Initialisierungsreihenfolge der verwendeten Klassen zyklusfrei ist.

4 Änderungen in der RTSJ im Vergleich zur Java Standard Edition

Neben dem bekannten *Heap-Memory* unterstützt RTSJ *Immortal-Memory*, das nie *garbage-collected* wird und das während der gesamten Programmlaufzeit existiert. Außerdem gibt es das sogenannte *Scoped-Memory*, das wie ein *Cactus-Stack* organisiert ist. *Scoped-Memory* muss ebenfalls nicht *garbage-collected* werden, sondern ein *Scope* wird stets komplett freigegeben.

Die von Standard-Java unterstützen Prioritäten müssen nicht verpflichtend beachtet werden, deshalb sind sie für Echtzeit-Anwendungen ungeeignet. RTSJ verwendet 28 zusätzliche, vorrangige Prioritäts-Level. Es werden erweiterbare Scheduling-Protokolle angeboten, um Prioritäts-Inversion zu vermeiden und speziellen Anwendungsbedürfnissen gerecht zu werden. Neben den bekannten *Java-Threads* gibt es *Real-Time-Threads*, die mit einer Real-Time-Priorität behaftet sind, die aber von einer *Garbage-Collection* unterbrochen werden können. Außerdem gibt es *No-Heap-Real-Time-Threads*, die den *Heap* nicht benutzen und deshalb auch keine *Garbage-Collection* auslösen können.

Asynchrone Ereignisse werden nicht an einen dedizierten *Thread* weitergeleitet. Vielmehr wird der von dem Ereignis betroffene *Thread* seine aktuelle Ausführung unterbrechen und einen alternativen, für diesen Fall vorgesehenen Algorithmus ausführen. Diese Vorgehensweise gleicht dem *Exception-Handling*; asynchrone Ereignisse müssen aber nicht an allen Stellen sofort bearbeitet werden.

Da Millisekunden für Echtzeit-Anwendungen nicht genau genug sind, werden in RTSJ-Programmen die Zeiten in Nanosekunden angegeben. Falls der Hardware-Timer keine Nanosekunden unterstützt, dann wird die feinstmögliche Auflösung verwendet. Auf die Unterschiede zwischen RTSJ 1.0 und RTSJ 1.1 [JSR282] wird nicht näher eingegangen.

5 Änderungen in der SCSJ im Vergleich zur RTSJ

SCSJ basiert auf den Ergebnissen des EU-Projekts HIJA [HIJA], wurde aber in vielen Punkten weiterentwickelt und verfeinert, um zu einer dem JCP genügenden Spezifikation und Referenz-Implementierung zu kommen. Das Hauptaugenmerk bei der Definition von JSR-302 ist es, eine Zertifizierung (z.B. nach DO 178B level A) so weit wie möglich zu unterstützen. Deshalb waren einige drastische Einschnitte in den Sprachumfang nötig, allen voran der Verzicht auf Heap-Memory und somit einigen Methoden des *java.lang-Paketes*. Von den Threads werden nur No-Heap-Realtime-Threads verwendet; die Synchronisation kann dadurch rein prioritätsgesteuert erfolgen. SCSJ unterstützt sowohl das *Priority-Inheritance* als auch das *Priority-Ceiling-Emulation-Protocol*, um Prioritäts-Inversion zu vermeiden.

Der von Java propagierte objektorientierte Programmierstil erfordert dynamische Allokation von temporären Objekten; dies wird durch Stack-basiertes *Scoped-Memory* erreicht. Dafür ist keine *Garbage-Collection* erforderlich und Werkzeuge können überprüfen, dass es keine *Dangling-Pointers* in freigegebenen *Scopes* gibt.

5.1 Ausführungsmodell: Mission

Safety-Critical-Applikationen werden als sogenannte *Safelets* (analog zu *Applets* und *Midlets*) ausgeführt. Ein *Safelet* besteht im wesentlichen aus drei Phasen:
Initialisierung: Alle Klassen müssen à priori verfügbar sein und eine zyklusfreie Initialisierungsreihenfolge muss explizit angegeben werden. *Threads* werden vor der eigentlichen Mission-Phase reserviert und initialisiert. Während der gesamten Initialisierungsphase wird kein *Heap-Memory* verwendet. Die verwendeten Datenstrukturen liegen im sogenannten *Mission-Memory*, dies ist eine Unterklasse von *Scoped-Memory*.
Mission-Phase: Ausführung der Run-Routine, die typischerweise den zyklische Aufruf einer oder mehrerer Methoden beinhaltet. Es können prioritätsbehaftete asynchrone *Event-Handler* verwendet werden. Für dynamische Speicheranforderungen steht nur *Scoped-Memory* zur Verfügung. Nach der Beendigung der *Mission-Phase* folgt die ***Cleanup-Phase***, die durch verschiedene Mechanismen erreicht werden kann. Die Mission kann neu gestartet, oder eine weitere Mission zur Ausführen vorgesehen werden. Das *Mission-Memory* wird dafür neu initialisiert aber nicht physisch freigegeben.

5.2 Konformitätsebenen

Um den widerstrebenden Interessen nach einem möglichst komfortablen Sprachumfang einerseits und dem Wunsch nach einfacher Analysierbarkeit andererseits gerecht zu werden, definiert SCSJ drei Konformitätsebenen. Der Hauptunterschied besteht darin, inwieweit geschachtelte Missionen möglich sind, aber auch der Sprachumfang wird auf jeder Ebene erweitert. Code von niedrigen Konformitätsebenen kann auch in komplexeren Missionen auf höherer Konformitätsebene verwendet werden. Die genauen Details der Abgrenzung sind zur Zeit noch in der Diskussion.

6 Zusammenfassung

Der Sprachentwurf von Java zielt darauf ab, möglichst viele Fehler bereits zur Übersetzungszeit zu finden; auch die Ausnahmebehandlung ist integraler Bestandteil der Sprache, ebenso wie *Threads* und deren Synchronisation. Java unterstützt Objektorientierung, Kapselung und *Information-Hiding*, zudem bietet es eine große Menge vordefinierter *Packages* in einer weltweit standardisierten Klassen-Hierarchie an. Durch sichere Objektreferenzen kann es keinen *General-Protection-Fault* oder *Segmentation-Fault* geben; auch Feldgrenzen werden überwacht; wenn möglich, schon zur Übersetzungszeit.

Alle Probleme, die den Einsatz von Java für Echtzeit- und sicherheitskritische Systeme verhindert haben, werden in der SCSJ ausgeräumt: Die Anwendung hat volle Kontrolle über das *Scheduling* und kann auf spezielle Speicherbereiche zugreifen; es werden hochauflösende Timer unterstützt und *Worst-Case-Execution-Times* sind berechenbar. Das Potential zum Nachweis von Korrektheits-Eigenschaften durch Werkzeuge ist durch eine präzise Definition der Semantik wesentlich höher als bei C oder C++. Sobald die *Virtuellen Maschinen* die JSR-282 (RTSJ) und JSR-302 (SCSJ) unterstützen von den Zertifizierungsbehörden anerkannt sind, bringt der Einsatz von Java eine hohe Produktivität, da - neben den bekannten Vorteilen - die geforderte Qualität leichter zu erreichen und nachzuweisen ist.

Literaturverzeichnis

[Bak06] Baker, J. et. al.: A Real-time Java Virtual Machine for Avionics - An Experience Report. In 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), IEEE Computer Society, 2006, S. 384-396

[Cor04] Corsaro, A.: Techniques and patterns for safe and efficient real-time middleware. PhD Washington University. http://portal.acm.org/citation.cfm?id=1087703&dl=ACM&coll=

[Do178] RTCA: DO 178B, Software Considerations in Airborne Systems and Equipment Certification. http://en.wikipedia.org/wiki/DO-178B, 1992

[HIJA] aicas GmbH: HIJA Safety Critical Java Proposal. http://www.hija.info, 2005

[JSR282] JSR 282: RTSJ version 1.1. http://www.jcp.org/en/jsr/detail?id=282, 2008

[JSR302] JSR 302: Safety Critical Java Technology. http://www.jcp.org/en/jsr/detail?id=302, 2008

[RTSJ] Bollella G. et. al.: The Real-Time Specification for Java. Addison-Wesley, 2000

[Sto07] Mark Stoodley et. al.: Real-Time Java, Part 1: Using the Java language for real-time systems. http://www-128.ibm.com/developerworks/java/library/j-rtj1/, 2007

Erfolgsfaktoren bei der Entwicklung der OBCE-WW

Dr. Thomas Buchberger

SIS PSE CVD CCC 3
Siemens AG Österreich
Erdbergerlände 26
A-1030 Wien
thomas.buchberger@siemens.com

Abstract: "GSM on Board" ist eine Lösung, welche die Benutzung von Handys in Flugzeugen ermöglicht. Die Erfahrungen, die wir bei der Entwicklung einer Teilkomponente - der OBCE-WW - gemacht haben, sind jedoch nicht spezifisch für den Bereich Aerospace (DO-178B), sondern typisch für alle Entwicklungen sicherheitskritischer Systeme.

1 Einführung

SIS PSE CVD CCC 3 ist eine Siemens Organisationseinheit, welche ihre Tätigkeit auf die Entwicklung von "Safety Devices" in den Bereichen Aerospace, Automotive und Transport konzentriert.

Unser wichtigstes Projekt in den vergangenen 2 Jahren war bzw. ist die Entwicklung einer Kontrolleinheit für die Generierung und Überwachung von HF-Signalen im System "GSM on Board", welches die Benutzung von Handys in Flugzeugen ermöglicht. Dieses aus Hardware und Software bestehende Gerät (OBCE-WW) wird im Auftrag von Airbus Deutschland für weltweiten Einsatz entwickelt. Die geltende internationale Richtlinie für Software-Entwicklung im Bereich Avionics ist der RTCA Standard DO-178B, der dem Produkt zugeordnete Software Level ist "C" bzw. "D".

Rückblickend haben folgende Konzepte beziehungsweise Tätigkeiten entscheidend zum Erfolg des Projektes beigetragen:

1. Tool-unterstütztes Requirements-Management
2. HW und SW Partitioning als Grundlage des Designs
3. Eigenentwicklung eines einfachen Tools für die Ermittlung der Statement Coverage
4. Testkonzept, Testautomatisierung

Auf diese 4 ausgewählten Schwerpunkte wird im Folgenden eingegangen.

2 Requirements-Management

System Requirements wurden vertikal in Teilbereiche aufgebrochen (HW, Level „C“ SW, Level „D“ SW, Prozess), in horizontalen Hierarchien verfeinert (High-Level bzw. Low-Level Requirements) und bis zu Test-Spezifikation und Test-Ergebnissen bidirektional verlinkt. Als Tool wurde Telelogic DOORS verwendet. Folgende Vorteile wurden von uns sehr geschätzt:

- Hersteller und Kunde „sehen“ dieselbe Version
- Einheitlicher Aufbau und einheitliches Format fördern Übersichtlichkeit und gemeinsames Verständnis
- Unterstützung des Requirement Linking und Tracing (Traceability Matrix, uncovered Requirements, suspect links)

3 Partitioning

Der Kundenwunsch nach einem hochsicheren Betriebs-Mode (Level „C“) und einem etwas weniger kritischem Maintenance-Mode (Level „D“) wurde mit HW und SW Partitioning implementiert

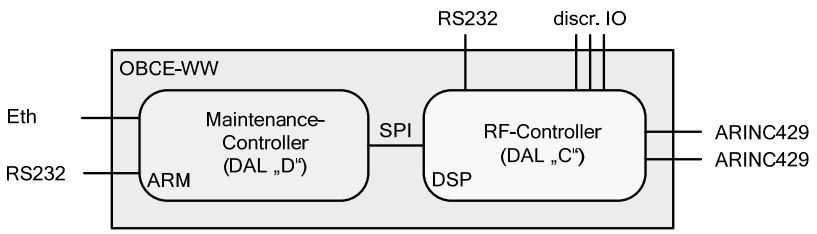


Abbildung 1: HW Partitioning (2 Prozessor Lösung)

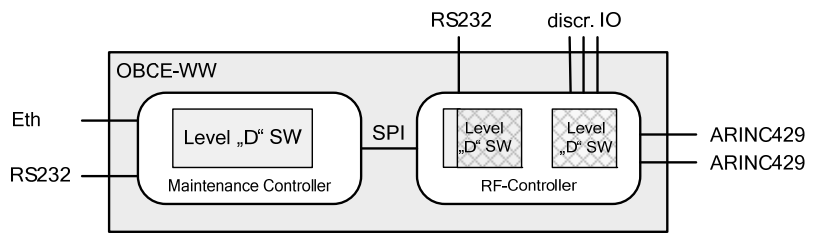


Abbildung 2: SW Partitioning

Im RF-Controller wird je nach Betriebsmode eine unterschiedliche Load verwendet. Die Maintenance-Load benutzt die Module der Betriebs-Load und stellt zusätzliche Funktionalität zu Verfügung.

4 Nachweis der Statement Coverage im Test

Der Einsatz kommerzieller Tools zur Ermittlung der Statement Coverage ist in kleinen Projekten nicht immer möglich, da die in sicherheitskritischen Umfeldern verwendeten Tools qualifiziert werden müssen bzw. auf speziellen Targets auch nicht sofort einsetzbar sind. Beides kann enorme Kosten und Aufwand verursachen.

Für das Projekt OBCE-WW wurde ein einfaches und „billiges“ Umfeld zur Ermittlung der Statement Coverage geschaffen. Instrumentierung des Source Codes und Visualisierung der Coverage erfolgen tool-unterstützt, der Einfluss auf das Laufzeitverhalten ist akzeptabel (<50%) und der Speicherbedarf minimal. Für die SW wurde eine Statement Coverage >99% erreicht.

5 Testkonzept, Testautomatisierung

Die Anforderung an die Verifikation, mittels eines weitgehend automatisch ablaufenden Test Frames ein abgeschlossenes embedded System als „Black-Box“ zu testen und dabei eine Statement Coverage von 100% zu erreichen scheint widersprüchlich, zumal aus den erwähnten Gründen kommerzielle Tools weitgehend ausscheiden, vollständige Transparenz gefordert wird und ein nur zum Testen verwendeter Code als „dead code“ gilt und somit nicht zulässig ist.

Als praktikable Methode die Testziele zu erreichen hat sich eine Auftrennung in einen Grey-Box und einen Black-Box Teil erwiesen. Im Grey-Box Test wird eine Test-Applikation zur Target SW dazugebunden, die einerseits das Interface erweitert und andererseits Zugang zu internen Daten und Funktionen ermöglicht.

Bei der OBCE-WW erfüllt der Maintenance-Mode exakt die Anforderungen an diese Test-Applikation, er kann also beinahe unverändert dazu herangezogen werden. Damit und mit der laufend erweiterten Automation konnte der Systemtest unseres durchaus komplexen Targets soweit verbessert werden, dass jetzt mehr als 80% der Testfälle automatisch ablaufen, wobei 90% aller SW Requirements abgedeckt und 85% aller Source Code Statements durchlaufen werden.

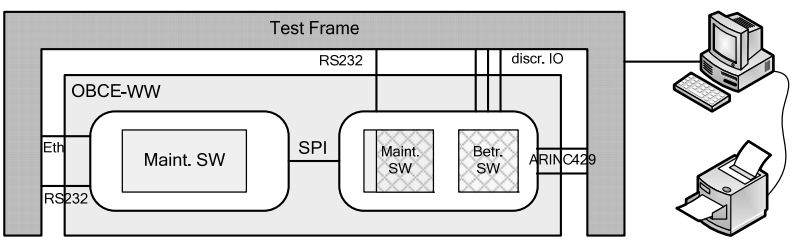


Abbildung 3: Test Setup

Modellsicht auf die Logik eines generischen Stellwerkssystems

Serhat Cakaloglu, Hans-Jürgen Nollau, Murat Sahingöz

Siemens AG, Industry Sector, Mobility Division
Ackerstraße 22, 38126 Braunschweig
{Serhat.Cakaloglu, Hans-Juergen.Nollau, Murat.Sahingöz}@Siemens.com

Zusammenfassung: In diesem Beitrag werden Methoden und Werkzeuge dargestellt, mit denen eine Modellsicht auf eine vorhandene Stellwerkslogik erstellt werden kann. Auf die besonderen Anforderungen eines generischen Systems wird eingegangen und die Einbindung in den Verifikations- und Validierungsprozess dargestellt. Ein Ausblick auf eine Anwendung für generalisierende Testaussagen wird gegeben.

1 Einführung

Bei dem elektronischen Stellwerk vom Typ Simis D (Simis - Sicheres Mikrocomputersystem von Siemens) handelt es sich um ein generisches System für die Anwendung bei der DB AG. Unser Team erstellt hierfür die Stellwerkslogik, die Softwarekomponente, die die sicherheitskritischen Abhängigkeiten beispielsweise zwischen Weichen und Signalen herstellt. Die Entwicklung der Stellwerkslogik beruht auf dem Konzept der generischen Komponenten (Fahrwegelemente). Jedes Fahrwegelement wird dabei isoliert betrachtet und seine Reaktionen auf eintreffende Ereignisse mittels GRACE-Modellen (GRACE - Graphical Requirements Analysis and design method in a CENELEC based Engineering process) spezifiziert. Die Elemente kommunizieren über definierte Kanäle – sogenannte Spuren.

Das Überführen ins Zielsystem erfolgt automatisiert mit der zugelassenen Werkzeugkette GRACE.

Von den Elementen werden im Rahmen der Projektierung Instanzen gebildet und mit Spuren nach definierten Regeln verknüpft. Die Instanzen der Elemente können auf verschiedenen Rechnern eines Rechnernetzwerkes liegen.

Das Funktionsverhalten eines konkreten Stellwerkes wird nicht nur durch die in den Elementen implementierte Logik bestimmt, sondern auch durch die Kommunikation der Elementinstanzen untereinander.

2 Warum Modellsichten auf eine vorhandene Logik anwenden?

Zu Entwicklungsbeginn im Jahr 2000 stand uns kein stabiles Werkzeug für eine modellbasierte Entwicklung eines komplexen generischen Systems zur Verfügung, das in unsere zugelassene Werkzeugkette integrierbar gewesen wäre.

Der Zwang zur Abstraktion in der Entwurfsphase war aufgrund der Komplexität der Steuerungsaufgabe gegeben und wurde beispielsweise durch Spezifikation von Funktionen mit Statemate, Animation und Tests der Modelle beantwortet. Es erfolgte eine manuelle Transformation in die zugelassene Entwicklungsumgebung GRACE.

Für eine gewachsene Stellwerkslogik ist das Einführen einer modellbasierten Entwicklung mit hohen Risiken verbunden. Für unser Team rückten folgende Fragestellungen in den Vordergrund:

1. Wie wird sich das System nach einer Änderung verhalten?

2. Wie interagieren die Elementinstanzen einer bestimmten topologischen Struktur für eine bestimmte Funktionalität?

3. Wie wird sich das System verhalten, wenn Elementinstanzen aufgrund topologischer Anforderungen anders angeordnet werden?

Um diese und verwandte Fragen effektiv beantworten zu können, erschienen insbesondere Sequenzdiagramme für einfach zu konfigurierende Elementanordnungen als Lösung geeignet. Diese Aufgabe wurde bei uns tätigen Studenten übertragen. Sie nutzten Zwischenresultate eines von der Siemens AG geförderten Promotionsprojektes GRACE-SIM (SIM - Simulation). GRACE-SIM basiert auf einem Codegenerator und einer domänenspezifischen Testsprache. Diese Testsprache dient der Konfiguration des aus den GRACE-Modellen erzeugten Codes und ermöglicht eine entwicklungsbegleitende Validierung der Systemfunktionen.

3 Entwicklungsbegleitende Validierung mittels GRACE-SIM

3.1 Datenbasis

Die Stellwerkslogik von Simis D wird für externe Werkzeuge in XML-Dateien abgebildet. Diese XML-Dateien sind als offene Schnittstellen zu GRACE-Quellen zu verstehen. Sie verfügen über alle Deklarationen und Telegramm-Empfangsmethoden und sind plattformunabhängig. Um dynamische Aspekte in einem Modell repräsentieren zu können (z. B. in einem Sequenzdiagramm) war es erforderlich, einen lauffähigen Code zu erzeugen und die Eigenschaften des Zielsystems nachzubilden.

3.2 Codegenerator

Der Codegenerator von GRACE-SIM nutzt die XML-Repräsentation der Stellwerkslogik. Er erzeugt einen Java-Code mit offen gelegter API (Application Programming Interface). Mit dieser API ist es möglich, das System beliebig zu konfigurieren und zu stimulieren. Die Konfiguration entspricht einer Vernetzung der Fahrwegelemente. Die Stimuli sind die Telegramme, die die Aufrufe von Empfangsmethoden in Fahrwegelementen auslösen.

Die Ablaufumgebung und die funktionalen Eigenschaften des Zielsystems werden mitgeneriert. GRACE-SIM behandelt Telegramme genau so, wie sie das Zielsystem umsetzen würde.

3.3 Definition des Testobjektes und Testsprache

Bestandteil von GRACE-SIM ist ein Editor zum Herstellen von Elementinstanzen und deren Verbindung mit anderen Elementinstanzen. Mit diesem Editor können beliebige komplexe topologische Strukturen abgebildet werden.

Die Artefakte der entwickelten Testsprache definieren die Konfiguration des Prüflings und können mehrere Testfälle enthalten. Die Konfiguration erlaubt beliebige Abbildungen komplexer topologischer Strukturen darzustellen, indem Instanzen von Fahrwegelementen und deren Kommunikationsverbindungen zu einander deklariert werden. Der interne Zustand von Instanzen kann initialisiert werden.

Die Testfälle definieren, welche Telegramme in welcher Reihenfolge an die Elementinstanzen gesendet werden. Das Senden von Telegrammen erfolgt erst nach dem Einnehmen eines stabilen Zustandes des zuvor gesendeten Testobjekttelegramms. Ein stabiler Zustand bedeutet, dass keine Kommunikation zwischen den Elementinstanzen stattfindet.

Es können auch Soll-Daten definiert werden, die nach dem Senden eines Telegramms auftreten müssen. Die Kommunikationen zwischen Elementinstanzen können ebenfalls als Soll-Daten genutzt werden. Um Timer und zyklische Ereignisse aus der Stellwerkslogik berücksichtigen zu können, wurde die Option Wartezeiten für Testfälle vorgesehen.

Die Testfälle werden durch einen für diese Testsprache entwickelten Parser auf dem generierten Code ausgeführt. Am Ende eines Testlaufs liegen die Ergebnisse als Sequenzdiagramme
(mit integrierten Berichten über durchlaufene/abgedeckte Empfangsmethoden oder Funktionen) vor.

Nicht relevante Datenflüsse können durch Anweisungen der Testfalldefinition ausgeblendet werden. Eine hierarchische Strukturierung von Testobjekten ist möglich. Komplexe topologische Strukturen können abstrahiert und wiederverwendet werden.

Die aufbereiteten Modellsichten werden verwendet, um elementübergreifende Systemverhalten zur Entwicklungszeit zu beobachten und auf deren Basis eine frühzeitige Validierung zu ermöglichen.

3.4 Anwendungsergebnisse

Erfolgreiche Anwendungsergebnisse liegen bei

- der Verifikation und Validierung komplexer Teilfunktionen
- der Fehlersuche
- der Schulung neuer Mitarbeiter

vor.

3.5 Weitere Zielsetzungen

Zurzeit werden die Testläufe an einer konkreten topologischen Struktur (Testbahnhof) durchgeführt. Aufgrund der beobachteten Soll-Daten kann für diese Struktur ein Gut-Schlecht-Entscheid ausgeführt werden. Auf Basis der repräsentativen Struktur dieser Testbahnhöfe wird auf die Soll-Funktionserfüllung in den anderen zur Anwendung freigegebenen Strukturen geschlossen.

Für das Definieren zulässiger topologischer Strukturen wurde an der TU Braunschweig das Beschreibungsmittel LpSpec (LpSpec - Lageplan-Spezifikation) entwickelt. LpSpec beherrscht das Darstellen von Häufigkeiten von Elementinstanzen und das Verschieben von Elementinstanzen in einer toplogischen Struktur.

Es liegt nahe, eine zulässige Ableitung der beschriebenen topologischen Struktur manuell zu untersuchen und weitere zulässige Ableitungen zu automatisieren. Neben der Schnittstelle zwischen LpSpec und GRACE-SIM bleiben eine optimale Suchstrategie und ein Ende-Kriterium für den Testprozess weiterhin offen.

Secure Coding als Bindeglied zwischen formalen Ansätzen um Safety in Infrastruktur-kritischen Systemen zu gewährleisten

Wolfgang Köppl, Dr. Gordon Rohrmair

Siemens AG, Corporate Technology
IC CERT
Otto-Hahn-Ring 6
81739 München
w.koeppl@siemens.com
gordon.rohrmair@siemens.com

Abstract: Safety und IT Security sind eng miteinander verknüpft. Safety schützt den Menschen vor Maschinen, während bei der IT Security ein System vor böswilligen Menschen geschützt wird. Der wichtigste Themenkomplex innerhalb der IT Security ist „Secure Coding" – die Kunst, sichere Applikationen zu erstellen. Die steigende Komplexität der Systeme führt dazu, dass vollständige formale Verifikation nicht mehr bezahlbar ist. 90% der erfolgreichen Angriffe auf IT Systeme basieren auf dem Ausnützen bekannter Programmierfehler. [Ko05] [Mi07] Durch Angriffe kann der Kontrollfluss des Gesamtsystems verändert werden. Um Safety wirklich herzustellen müssen die formalen Ansätze oder Modelle richtig auf der Implementierungsebene verankert werden. Secure Coding schließt die entstehenden Lücken und erhöht die Verlässlichkeit von Safety Funktionalität selbst bei absichtlicher Fehlbedienung des Systems wie beispielsweise einem Angriff. Die Autoren zeigen, warum Security und im speziellen Secure Coding in der Praxis eine Voraussetzung für Safety ist. Secure Coding stellt dabei auch die passenden Ergänzungen für aktuelle Programmiertrends zur Verfügung und integriert sich nahtlos in den Entwicklungsprozess.

1 Zusammenhang von Safety und Security

Safety und Security werden im Deutschen meist jeweils als Sicherheit übersetzt. Während Safety das Ziel hat, Schaden an Personen in jeder erdenklichen Situation abzuhalten, behandelt Security den Schutz vor Risiken vorwiegend durch Angreifer. IT Security soll sicherstellen, dass feindliche Aktionen niemals dazu führen, dass der vordefinierte Kontrollfluß verändert wird. Bei Änderung der Zustandsübergänge des Systems können auch Safety Maßnahmen ausgehebelt werden.

Wenn man Safety von einer theoretischen Seite her betrachtet, also das Sicherstellen, dass unter allen Umständen das System das Verhalten offeriert, das spezifiziert wurde – und nur das, dann kann Security als Unterpunkt von Safety gesehen werden.

In den vorigen Jahrzehnten wurde Safety vor allem dadurch erzielt, Systemsicherungen zu inkludieren und diese auf abstrakter Ebene zu verifizieren. Die Sicherungen sollten garantieren, dass das Gesamtsystem - unabhängig von dem Zustand in dem es sich befindet – menschliches Leib und Wohl nicht in Gefahr bringt.

Unabsichtliche Fehlbedienungen sind dabei die Hauptbedrohung. Heutzutage spielt aber auch IT Security eine zunehmend wichtigere Rolle für Safety, da die Angriffsoberfläche wächst. Personen, die Schaden verursachen wollen, haben dazu eine Vielzahl an Möglichkeiten. Die drei Hauptgründe für diese Beobachtung sind:

- Die zunehmende Vernetzung von kritischen Systemen
- Die steigende Komplexität
- Das immer weniger benötigte Know-How um gefährliche Angriffe auszuführen.

Medizinische Geräte wie Computer-Tomographen sind an Krankenhaus-Netzwerke angebunden. Jeder andere Rechner in diesem Netzwerk kann potentiell als Werkzeug für den Angriff verwendet werden. Dies sind in vielen Fällen einige hundert bis tausende Systeme.

Aufgrund des steigenden Kostendrucks wird auch bei kritischen Systemen auf Drittkomponenten zurückgegriffen. Komplexe Steuerungssysteme in der Energieversorgung integrieren meist einige Dutzend Fremd-Software-Teile und setzten fast immer auf Standard-Betriebssystemen auf. Bezogen auf das Produktportfolio der Siemens AG bedeutet das, dass Softwareprodukte bis zu 70 Komponenten von Drittherstellen aufweisen. Wenn eine Schwachstelle für einen dieser Komponenten bekannt wird, ist dadurch meist das gesamte Produkt angreifbar.

Ein Angreifer benötigt nur die Versionsnummer einer Komponente des Angriffsziels und schon kann er über öffentlich zugängige Quellen Informationen über Schwachstellen und sogar Angriffsprogramme herunterladen. Das benötigte Know-How wird dabei auf einen einfachen „Doppelklick“ reduziert. IT Security ist aus diesen Gründen in vielen Fällen zu einer Vorraussetzung geworden, um Safety erreichen zu können.

2 Safety ohne Security scheitert in der Praxis

Wenn IT Security nicht ernst genommen wird, kann dies zu beträchtlichen Bedrohungen für Menschen führen. Einige auftretende Vorfälle werden öffentlich bekannt, die meisten aber verschwiegen.

- Im Januar 2008 schaffte es ein polnischer Teenager, eine Fernseh-Fernbedienung so zu manipulieren, dass er S-Bahn steuern konnte. Vier S-Bahnen in Lodz entgleisten und mehrere Personen wurden dabei verletzt [He08].
- Angestellte eines deutschen Krankenhauses berichteten dem Siemens CERT im Rahmen eines Projektes von einem andernen Vorfall. Eine kritische Operation musste abgebrochen werden, da die nötigen Informationen des Patienten und die Ergebnisse der Voruntersuchungen nicht verfügbar waren. Ein Wurmbefall hatte die

medizinischen Geräte außer Betrieb gesetzt. Der Patient war bereits narkotisiert und auf dem Operationstisch.

Derartige Vorfälle können in den letzten Jahren von Siemens CERT zunehmend beobachtet werden. Es wird dabei immer offensichtlicher, dass Safety ohne Security in der Praxis bei komplexen Systemen meist scheitert.

3 Security ergänzt modellbasierte und formale Ansätze

In der Ausschreibung zu diesem Workshop wurden verschiedene Trends im Software-Engineering aufgezeigt und deren Relevanz und Anwendbarkeit hinterfragt. Jeder dieser Ansätze hat substantielle Schwächen, die durch IT Security Maßnahmen ausgebessert werden können.

3.1 Modellbasierte Entwicklung

Modellbasierte Entwicklung ist sowohl für funktionale Anforderungen als auch für Sicherheitsanforderungen wie Zugangskontrolle eine gute Hilfe. Allerdings kann ein Modell keine spezifischen Probleme einer Zielplatform behandeln. Dies reicht nur aus, wenn beweisbare Sprachen wie ein Subset von Ada [Gu90] und keine Fremdkomponenten verwendet werden. Für alle anderen Anwendungsfälle, die die Mehrheit darstellen, ist Security Wissen über die Zielplatform essentiell.

90 % der erfolgreichen Angriffe basieren auf dem Ausnutzen von bekannten Implementierungsfehlern [Ko05]. Da diese Fehler ihren Ursprung nicht im Design finden, können sie auch nicht über das Entwickeln von korrekten funktionalen Modellen vermieden werden. Für eine beträchtliche Anzahl von Schwachstellen kann allerdings ein IT Security Konzept modelliert und erstellt werden, das bei Auftreten von Softwarefehlern deren Einfluss auf das Gesamtsystem vermindert (z.B. Buffer Overflow Protections [Mi06] [HK01].

3.2 Formale Methoden

Viele kritische Infrastrukturen sind sehr komplex. Wodurch die formale Verifikation meist zu kostenintensiv wird. Daher werden zum einen nur Teile formal verifiziert und nicht das gesamte System. Zum anderen wird meist nicht auf Implementierungsebene verifiziert sondern eine Abstraktion des Source Codes als Beschreibung für das formale Modell herangezogen.

Die Erfahrungswerte der IT Security, welche Angriffe bei bestimmten Umgebungen wahrscheinlich sind, stellen somit eine essentielle Ergänzung dar. Sie geben einerseits Auskunft darüber, welche Komponenten mit großer Wahrscheinlichkeit angegriffen werden, andererseits geben sie eine Einschätzung über den erreichbaren Sicherheitslevel in einem bestimmten Abstraktionsniveau. Somit ermöglichen sie auch für nicht formal verifizierte Teile einen adäquaten Security und Safety Level zu erreichen.

3.3 Testgenerierung

Wenn Tests nur automatisiert aufgrund der Funktionalitätsbeschreibung eines Produktes erstellt werden, werden meist mögliche Hacker-Angriffe übersehen. Schwachstellen der Zieltechnologie werden dann nicht abgedeckt. Bei C/C++ sind Tests auf Buffer Overflows beispielsweise extrem wichtig. Bei Java wäre diese Bedrohung nicht relevant. Diese Bedrohung würde daher in einer abstrakten und Technologie-unabhängigen Funktionsbeschreibung nicht erwähnt werden. Hier haben nur Erfahrungen aus der IT Security die Chance, Angriffspunkte auf Safety zu finden.

3.4 Codegenerierung

Direkte automatisierte Code-Generierung, beispielsweise aus einem Modell, ist eine vielversprechende Technologie. Viele Schwachstellen, die Programmierer potentiell erzeugen, können so vermieden werden. Dies gilt vor allem in funktioneller Hinsicht.

IT Security betrachtet vor allem aber auch Schwachstellen, die sehr spezifisch für die Zielumgebung sind. Somit sollte Secure Coding Wissen in die Codegenerierungs-Funktionalität einfließen, damit sicher gestellt wird, dass sicherer Code generiert wird. Beispielsweise kann das Template für Datenbank-Abfragen so erstellt werden, dass SQL Injections nicht mehr möglich sind.

Generatoren erstellen meist nicht 100% des Source Codes. Der Programmierer muss noch eingreifen oder neuen Code ganz hinzufügen. Es ist essentiell, dass auch er keine Schwachstellen verursacht. Wichtig dabei ist, festzulegen, ab wann diese Ansätze im Produktlebenszyklus greifen müssen.

4 Integration von IT Security in den Entwicklungsprozess

Der beste Weg, um adäquate IT Security bei besonders kritischen Applikationen zu garantieren, ist die Integration von Maßnahmen in den Entwicklungsprozess. Abbildung 1 zeigt den Good Practice Ansatz, der aufgrund der Erfahrungen des Siemens CERT kreiert wurde und sich am Siemens Referenzprozesshaus orientiert.

Es ist wichtig, bereits in den Plan und Define Phasen umfassenden Fokus auf IT Security Maßnahmen zu legen. Eine Schwachstelle im Design auszumerzen kostet nur etwa ein Achzigstel von dem, was die Beseitigung im Betrieb kostet [Ib03].

4.1 Plan

Die ersten wichtigen Schritte sind Gesetze und Standards zu sichten und eine für IT Security verantwortliche Person zu benennen. Regulatorien zu beachten ist nicht nur für Safety essentiell, sondern spielt auch für IT Security eine zunehmend wichtige Rolle. In den USA definieren beispielsweise die NERC Standards [Ne08] umfassende IT Security Vorgaben für Energieerzeuger und -verteiler.

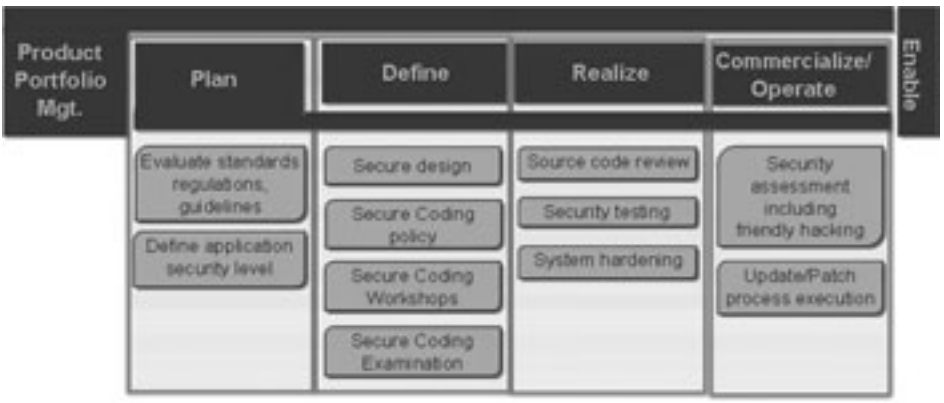


Abbildung 1: Integration von IT Security in den Entwicklungszyklus

4.2 Define

Die Define Phase kann als die wichtigste für IT Security gesehen werden. Nicht nur ein sicheres Design ist essentiell, sondern vor allem müssen in dieser Phase die Vorraussetzungen geschaffen werden, dass die Entwickler bei der Umsetzung die Schwachstellen von vorneherein vermeiden können. Einerseits sollen Programmiervorgaben (Secure Coding Policy) definiert werden, andererseits sollten diese aber auch durch Workshops für Entwickler ergänzt werden. Nur wenn die Programmierer die Fallen verstehen, können sie diese auch vermeiden. Bei kritischen Applikationen sollte man dieses Wissen auch durch Prüfungen verifizieren (Secure Coding Examination).

4.3 Realize

Bei der Implementierung soll die Einhaltung der Secure Coding Policy verifiziert werden. Dies geschieht einerseits im Rahmen des Source Code Reviews und andererseits im Rahmen des Tests der Applikation. Da meist eine Vielzahl von Fremdapplikationen eingesetzt werden, muss sichergestellt werden, dass diese richtig installiert, konfiguriert und aktualisiert werden (System hardening).

4.4 Commercialize/Operate

Bevor das Produkt an den Kunden geliefert oder beim Kunden in Betrieb genommen wird, sollte noch eine Sicherheitsüberprüfung aus Angreifer-Sicht gemacht werden. Oft tauchen Probleme erst dann auf, wenn Produkte in das Zielumfeld integriert werden. Im Gegensatz zu formal verifizierbaren Teilen einer Anwendung, kann man keine 100%-ige IT Security für das Gesamtsystem erreichen. Daher ist es auch wichtig, eine technische und prozessmäßige Möglichkeit zu schaffen, die Applikation zu aktualisieren, wenn Probleme bekannt werden. Dies ist besonders auch für Drittkomponenten relevant.

5 Zusammenfassung

Formale Ansätze sind lebensnotwendig für kritische Infrastrukturen. Allerdings sind diese Methoden in der Praxis nicht immer nahtlos ineinander integrierbar und anwendbar. Die entstehenden Lücken werden durch IT Security, im speziellen durch Secure Coding, geschlossen. Secure Coding Know-how liefert den formalen Ansätzen Erfahrungswerte darüber, welche Komponenten in welchem Abstraktionsgrad untersucht werden müssen. Technologie-spezifische Herausforderungen können durch formale Ansätze meist nicht ausreichend beantwortet werden. Secure Coding sorgt als Ansatz auf Implementierungsebene dafür, dass das Design sicher umgesetzt wird.

Literaturverzeichnis

[Gu90] Guaspari, D. et al, Formal verification of Ada programs, 1990

[He08] Heise Security Artikel, URL: http://www.heise.de/security/news/meldung/101674, [Stand 2008-01-25]

[HK01] Eto Hiroaki, Yoda Kunikazu, ProPolice: Improved stack-smashing attack detection, IEIC Technical Report, 2001

[Ib03] IBM Systems Sciences Institute, Security Defect Cost Multiplier, 2003

[Ko05] Eggün Kocatürk, Patch-Management vermeidet Flickwerk, Computerwoche, 2005

[Mi06] A detailed description of the Data Execution Prevention (DEP) feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003, URL: http://support.microsoft.com/kb/875352 [Stand 2008-01-25]

[Mi07] MITRE, Common Vulnerabilities and Exposures, http://cve.mitre.org/, 2007

[Ne08] NERC, URL: http://www.nerc.com/, 2008

Nachweis von Sicherheitseigenschaften in modellbasierten Entwurfsprozessen

Thomas Peikenkamp

Bereich Sicherheitskritische Systeme, OFFIS,
Escherweg 2, 26121 Oldenburg
peikenkamp@offis.de

Abstract: Mit der zunehmenden Modellbildung und modellbasierten Entwicklung im Bereich sicherheitskritischer Systeme gewinnt die Frage Bedeutung, wie effizient und zeitgerecht der Nachweis der Sicherheit für Systeme geführt werden kann, die über lange (und zunehmend länger werdende) Entwicklungszeiträume hinweg (nur) als Modell vorliegen.

Ein modellbasierter Ansatz zur Führung des Nachweises von Sicherheitseigenschaften unterliegt dabei folgenden Rahmenbedingungen:

1. Während früher vor allem Hardware Gegenstand der Modellierung war, gibt es heute praktisch keine Technologien mehr, für die nicht entsprechende Modellierungsmethoden etabliert sind. Im Hinblick auf die zunehmende Integration dieser Technologien müssen die auf ihnen basierenden Systeme mit den ihnen eigenen Modellierungsmethoden erfassen werden können.
2. Die notwendige Ersetzbarkeit von Komponenten eines Systems erfordert, dass die Gleichwertigkeit nicht nur im Hinblick auf funktionale, sondern auch auf nicht-funktionale Eigenschaften bewertet werden kann.
3. Wiederverwendbarkeit als erklärtes Ziel des komponentenbasierten Entwurfs führt zum Einsatz von Komponenten, für die Sicherheitsanforderungen nicht oder nur in anderem Kontext nachgewiesen sind.
4. Die zunehmende Integration von Komponenten unterschiedlicher Zulieferer erfordert eine Entwurfmethodik, die gestattet, den Nachweis der Sicherheit auch über eine Vielzahl von Entwurfsschritten, verteilt auf nicht-triviale Supply-Chains zu führen.
5. Die Methodik muss adaptierbar sein im Hinblick auf die unterschiedlichen Durchdringungsgrade modellbasierter Entwicklungsmethoden im industriellen Entwicklungsprozess, wobei anzumerken ist, dass dieser Faktor nicht allein unternehmensabhängig ist, sondern aufgrund der Heterogenität der involvierten Teilsystem die Modellbildung für diese unterschiedlich stark ausgeprägt ist.
6. Mit der gerade durch die modellbasierte Entwicklung vorangetriebenen Verwendung von Entwicklungswerkzeugen und automatisierten Entwicklungsschritten muss der Nachweis der Sicherheit auch die Rolle dieser „Komponenten“ bei der Validation erfassen.

Dieser Beitrag zeigt auf, wie der Sicherheitsnachweis im modellbasierten Entwurfsprozess unter Berücksichtigung der obigen Rahmenbedingungen er-

folgen kann: Ein um nicht-funktionale Eigenschaften erweitertes Komponentenmodell für heterogene Systeme in Verbindung einer auf den modellbasierten Ansatz zugeschnittenen Erweiterung traditioneller Sicherheitsanalysetechniken garantieren dabei zu jedem Zeitpunkt des Entwurfs eine kohärente Sicht auf die Sicherheitseigenschaften eines Systems und seiner Komponenten.

1 Etablierung einer Safety-Sicht

Als eine Maßnahme zur Komplexitätsreduktion im modellbasierten Entwurf hat sich die sichtenorientierte Modellierung bewährt. Typische Sichten, die dabei unterstützt werden, sind etwa Daten, Struktur/Architektur, Verhalten/Funktion, Ablauf/Szenarien. Die Bewertung der Sicherheit eines Systems stützt sich dabei typischerweise auf *alle* diese Sichten ab, so dass sich die Frage stellt, wie hier trotzdem eine „Sicherheitssicht“ etabliert werden kann. Der hier vorgeschlagene Ansatz besteht darin, eine Sicht auf das System eigenschaftsorientiert zu definieren, so dass etwa alle funktionalen Eigenschaften die funktionale Sicht, Performanceeigenschaften die Performance-Sicht oder die Gesamtheit aller Sicherheitseigenschaften die Safety-Sicht auf ein System liefern. Der Vorteil einer solchen Aufteilung ist, dass semantische Abhängigkeiten der Safety-Sicht von anderen Sichten (z.B. Echtzeiteigenschaften, Performanceeigenschaften) – zusammen mit den entsprechenden Validierungsanforderungen - früh explizit identifiziert werden können. So kann etwa überprüft werden, ob die Verletzung einer funktionalen Anforderung die Verletzung einer Sicherheitseigenschaft nach sich ziehen kann. Derzeit werden im Rahmen des SPEEDS Projektes die hierfür erforderlichen Analysetechniken entwickelt. Voraussetzung hierfür ist ein gemeinsames semantisches Modell für funktionale und nicht-funktionale Eigenschaften, welches in [S07] beschrieben ist.

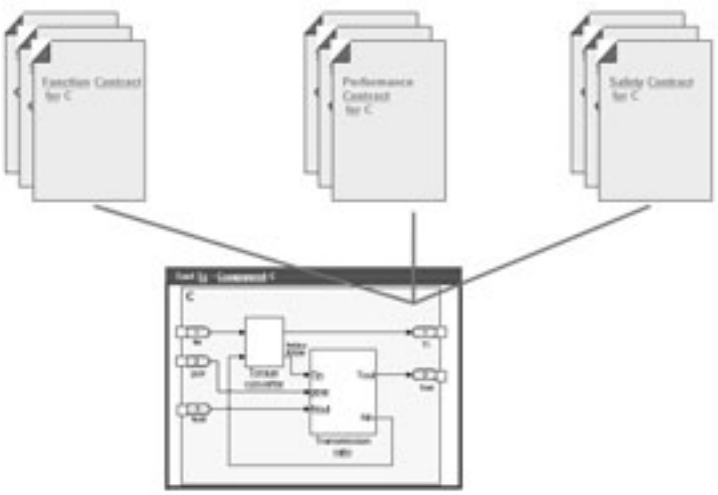


Abbildung 1: Contracts für 3 Sichten

2 Gleichwertigkeit/Ähnlichkeit von Komponenten aus Safety-Sicht

Zentrales Ziel beim komponentenbasierten Entwurf ist, erstens, die Wiederverwendbarkeit von Komponenten auf Basis definierter (und in der Regel bereits nachgewiesener) schnittstellen-orientierter Eigenschaften der Komponenten und, zweitens, die Ersetzbarkeit von Komponenten durch andere, bezüglich ihrer Schnittstellen „gleichwertiger" Komponenten. Die Gleichwertigkeitsanforderungen der Schnittstellen sind dabei abhängig von der jeweiligen Entwurfssicht. Für die Safety-Sicht bestimmen solche Gleichwertigkeitsanforderungen etwa, dass Teilsysteme nur durch Systeme *mit gleichem Safety Integrity Level* ersetzt werden dürfen. Da solche Anforderungen in der Regel nur unter bestimmten Umgebungsbedingungen und häufig auch nur unter Annahmen über den Bedienkontext eingehalten werden, sind sie als „Vertrag" (Contract) aufzufassen, der „Gegenleistung" für das Einhalten einer Umgebungsbedingung (Assumption) die Bereitstellung eines definierten Sollverhaltens verspricht (Promise). Beim Ersetzen einer Komponente durch eine neue (Implementierung der) Komponente kann dann überprüft werden, ob die ursprünglich vereinbarten Verträge noch eingehalten werden.

3 Kompositionelle Validierung

Um das obige Ersetzungsszenario auch (formal) validieren zu können müssen die Contracts einer Komponente aus den Contracts der Teilkomponenten ableitbar sein. Im Unterschied zur kompositionellen Validierung von funktionalen Eigenschaften [Dw98] muss hier auch das Fehlverhalten der Komponente aus dem Fehlverhalten der Teilkomponenten ableitbar sein.

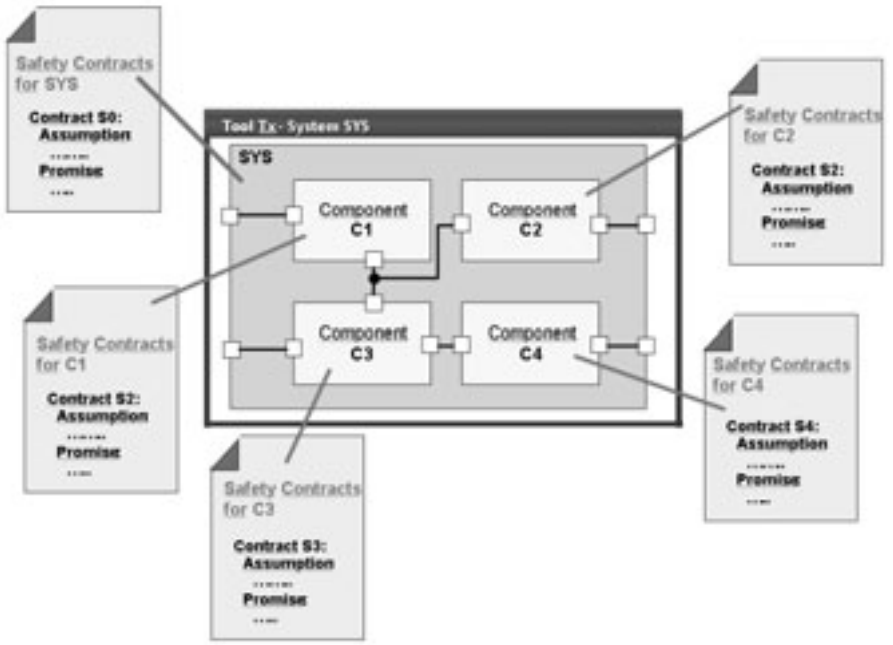


Abbildung 2: Ableitung von Contracts

Die hierfür erforderliche, ebenfalls im SPEEDS Projekt realisierte Analyse (Dominance-Check) stellt sicher, dass jedes Fehlverhalten welches durch die Safety-Contracts der Komponenten C1,...,C4 erlaubt ist auch durch den Safety-Contract von SYS erlaubt ist. Insoweit wird sichergestellt, dass auch komplexe Fehlersituationen, die durch das Zusammenspiel von u.U. defekten Komponenten C1,...,C4 entstehen durch den Safety-Contract von SYS erfasst sind. Typischerweise wird im frühen Entwurf der Safety-Contract von SYS gegeben sein und im Rahmen der Entwurfsverfeinerung sind dann entsprechende Sicherheitsanforderungen für die Teilkomponenten zu allokieren. Der Dominance-Check erlaubt dann diese Eigenschafts-Dekomposition abzusichern.

4 Untersuchung von detaillierten Systemmodellen: Fehlerinjektion

Ziel einer detaillierten Sicherheitsanalyse ist insbesondere die Identifikation von zusätzlichen Fehlern, die durch die Wahl einer Implementierung zustande kommen [Jp00]. Liegen detaillierte Modelle für einzelne Komponenten vor, so lassen sich diese durch eine Injektion von Fehlern um Fehlverhalten anreichern (Failure Mode Capturing) und erlauben somit eine Evaluierung der Fehlerfolgen in dem angereicherten Modell (Extended System Model), wobei die Modellanalyse dann das ursprüngliche, nominale Verhalten mit dem angereicherten Verhalten vergleichen muss um etwa Ursächlichkeit von Fehlern festzustellen.

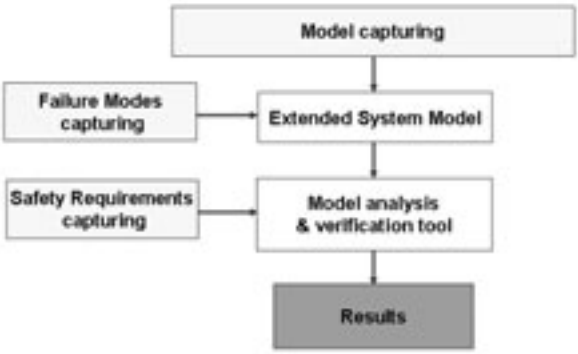


Abbildung 3: Modellbasierte Sicherheitsanlyse von Komponenten

Die Analysephase (Model analysis & verification tool) kann dann mit simulationsbasierten Techniken durchgeführt werden, wobei allerdings eine entsprechende Fehlerabdeckung sicherzustellen ist, wozu z.B. Techniken aus der automatischen Testgenerierung zum Einsatz kommen können. Eine neuere Methode mit erheblich höherer Zuverlässigkeit (bis zu 100% Abdeckung) wurde aus formalen Verifikationstechniken abgeleitet und gestattet etwa bei den obigen Ausgangsdaten die Erzeugung eines vollständigen (bezüglich der injizierten Fehler) Fehlerbaums aus einem Systemmodell heraus. In [Pt04] wurde damit ein Hochauftriebssystem eines Flugzeuges untersucht. In [Pt06] werden Erweiterungen dieser Methode im Hinblick auf Sicherheit von Diagnosesystemen, Berücksichtigung von *Common Causes* und Betriebsbedingungen vorgestellt.

5 Zusammenfassung

Das hier beschriebene Vorgehen erlaubt die Führung des Nachweises von Sicherheitseigenschaften inkrementell in bestehende modellbasierte Entwicklungsprozesse zu integrieren. Durch die Wahl eines Komponentenmodells, in dem Sicherheitseigenschaften unabhängig von dem (Entwurfs-)Modell der Komponente erfasst werden können, werden dabei auch Entwicklungsprozesse unterstützt bei denen etwa nur in frühen Entwurfsphasen oder nur teilweise (z.B. auf Komponentenebene oder Teilsystemebene) modellbasiert entwickelt wird. Für die technologische Unterstützung werden Integrationsplattformen benötigt, die insbesondere die um Contracts angereicherten Komponentemodelle unterstützen [Be05] und eine Einbindung von COTS Tools gestatten. Derzeit wird eine solche bus-basierte Plattform z.B. im SPEEDS Projekt entwickelt.

Literaturverzeichnis

[Be05] E. Brinksma, G. Coulson, I. Crnkovic, A. Evans, S. Gérard, S. Graf, H. Hermanns, J.-M. Jézeqél, B. Jonsson, N. Plouzeau, A. Ravn, Ph. Schnoebelen, F. Terrier, and A. Votintseva. Part ii: Component-based design and integration platforms. In B. Bouyssounouse and J. Sifakis, editors, *Embedded Systems Design: The ARTIST Roadmap for Research and Development*, number 3436 in LNCS, pages 103-214. Springer, 2005.

[Be06a] Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., et al. (2006). Compositional Performability Evaluation for STATEMATE. Proc. of QEST.

[Be06b] Böde, E., Damm, W., Hoyem, J., Josko, B., Niehaus, J., and Segelken, M.; Adding Value to Automotive Models, In Broy, M., Krüger. I.H., and Meisinger, M., Automotive Software - Connected Services in Mobile Networks. First Automotive Software Workshop (ASWSD 2004) San Diego. Revised Selected Papers, Lecture Notes in Computer Science 4147, Springer-Verlag, pp. 86-102, 2006.

[Dw98] W. Damm, B. Josko, H. Hungar, A. Pnueli: A Compositional Real-time Semantics of STATEMATE Designs, In de Roever, W.-P. , Langmaack, H., and Pnueli, A. (Edts.), Compositionality: The Significant Difference; International Symposium COMPOS '97, Lecture Notes in Computer Science 1536, Springer-Verlag, pp. 186-238, 1998.

[JMM08] B. Josko, Q. Ma, A. Metzner: Designing Embedded Systems using Heterogeneous Rich Components, 2008 (submitted).

[Jp00] P.H. Jesty, K.M. Hobley, R. Evans, I. Kendall, Safety Analysis of Vehicle-Based Systems. Proceedings of the 8th Safety-critical Systems Symposium, 2000.

[Pt04] T. Peikenkamp, E. Böde, I. Brückner, H. Spenke, M. Bretschneider, and H.-J. Holberg. Model-based Safety Analysis of a Flap Control System. In *Proceedings of the INCOSE 2004 -- 14th Annual International Symposium*, Toulouse, 2004.

[Pt06] Peikenkamp, T., Cavallo, A., Valacca, L., Böde, E., Pretzer, M., & Hahn, E. M. (2006). Towards a Unified Model-Based Safety Assessment.

[S07] SPEEDS Core Meta-model Syntax and Draft Semantics. SPEEDS internal Deliverable D.2.1.c, 2007.

Zertifizierung einer Sicherungskomponente mittels durchgängig formaler Modellierung*

Udo Frese[1], Daniel Hausmann[2], Christoph Lüth[1], Holger Täubig[1], Dennis Walter[1]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz, Bremen
[2] FB 3 — Mathematik und Informatik, Universität Bremen
christoph.lueth@dfki.de

1 Einführung

Dieses Papier berichtet über Arbeiten im Rahmen des Projektes SAMS, in welchem eine Fahrwegsicherungskomponente für Serviceroboter und fahrerlose Transportsysteme entwickelt und ihre Software nach IEC 61508 zertifiziert wird. Neu ist hierbei der durchgehende Einsatz formaler Modellierung und Beweis als Mittel zur Zertifizierung, das es uns erlaubt, die erforderlichen Sicherheitsbedingungen sehr abstrakt mathematisch zu formulieren, und gleichzeitig den Bezug zur Implementierung bewiesen korrekt herzustellen.

Indem wir die Grundlagen der Anwendungsdomäne (Geometrie und die Mechanik bewegter Objekte) in einem Theorembeweiser formalisieren, können wir die Sicherheitsanforderungen auf einer abstrakten, mathematischen Ebene formulieren. Diese sind dann wesentlich leichter zu validieren als implementationsnah formulierte Programmeigenschaften. Darüber hinaus modellieren wir die Implementierung innerhalb desselben formalen Rahmens. Dadurch erhalten wir einen nahtlosen Entwicklungsprozess mit einem klaren und beweisbar korrekten Übergang von den Sicherheitseigenschaften zu der Implementation.

Wir glauben, dass diese Vorgehensweise für alle Sicherheitszertifikationen nützlich sein kann: die formale Modellierung erzwingt, dass alle Annahmen, die in den Sicherheitsanforderungen, der Implementation oder den Beweisen implizit enthalten sind, explizit gemacht werden, so dass sich der Zertifikationsprozess auf die Sicherheitsanforderungen konzentrieren kann, da die Korrektheitsbeweise automatisch geprüft werden können.

2 Das Projekt SAMS

Ziel des Projektes SAMS ist die Entwicklung einer Fahrwegsicherungskomponente für Serviceroboter und fahrerlose Transportsysteme, welche die Zulassungsvoraussetzungen

*Forschung gefördert durch das BMBF im Rahmen der *Leitinnovation Servicerobotik*, Fkz. 01 IM F02

für Sicherheitsgeräte entsprechend IEC 61508 erfüllt. Die Komponente überwacht mittels eines Sicherheitslaserscanners ein Schutzfeld, das sich dem veränderlichen Zustand (Geschwindigkeit, Richtung, Lenkwinkel etc.) des Fahrzeugs dynamisch anpasst, und veranlasst rechtzeitig vor der Kollision mit einem Hindernis eine Bremsung. Industriell verfügbare Lösungen arbeiten nur mit wenigen vorkonfigurierten Schutzfeldern, was Bahnverlauf und Geschwindigkeit unnötig einschränkt

Für Forschungsprototypen ist die dynamische Schutzfeldberechnung Stand der Technik, aber keine existierende Implementation ist nach den entsprechend der Maschinenrichtlinie erforderlichen Normen zertifiziert worden. Kernvorhaben des Projektes ist deshalb die Zertifizierung einer dynamischen Schutzfeldberechnung nach der relevanten Norm IEC 61508 als eine Sicherheitskomponente, die für Anwendungen bis SIL-3 geeignet ist. Dieses ist sowohl für die Robotik als auch für die Sicherheitszertifikation neu, da typischerweise die Software zertifizierter Sicherheitseinrichtungen algorithmisch sehr einfach ist und Normen wie DIN EN ISO 10218 hardwarebasierten Lösungen wie Lichtvorhängen oder Käfige betonen.

3 Sicherheitsanforderungen und ihr Nachweis

Die globale Sicherheitsanforderung des zu entwickelnden Systems ist auf informelle Weise sehr einfach zu formulieren. Es muss garantiert werden, dass das gesicherte Gerät (der Roboter) nicht mit Hindernissen —insbesondere in der Nähe stehenden Personen— kollidiert. Die durch die Sicherungskomponente erhöhte Flexibilität im Fahrverhalten geht mit einer erhöhten Komplexität der eingesetzten Software einher. Hieraus ergeben sich weitere Sicherheitsanforderungen auf Modulebene, welche in ihrer Gesamtheit die Erfüllung der globalen Sicherheitsanforderung ergeben müssen.

In informellen Verifikationsansätzen kann der Schluss von der Erfülltheit der tiefer liegenden Anforderungen auf die Erfülltheit der globalen Anforderungen selbstverständlich auch nur informell gezogen werden, so dass z.B. Anforderung übersehen oder Annahmen nur implizit gemacht werden (z. B. dass mithilfe des Laserscanners auch wirklich alle Hindernisse erkannt werden können). Demgegenüber werden in der durchgängig formalen Modellierung und Verifikation —ausgehend von Eigenschaften und Gestalt des Roboters und der Umgebung, über die Sicherheitsanforderungen, bis hin zum Quellcode— derartige Fehler vermeiden, und damit trotz hoher Komplexität des entwickelten Systems Vertrauen in dessen Korrektheit geschafft. Als Modellierungs- und Beweiswerkzeug kommt in diesem Projekt Isabelle/HOL zum Einsatz, mit dem wir gute Erfahrungen gesammelt haben.

Domänenmodellierung. Die Welt, d. h. der Roboter und seine Umgebung, werden in der reellen Ebene modelliert. Dies ist adäquat, da die Sensoren des Roboters (Odometrie und Laserscanner) nur zweidimensionale Informationen liefern, und der Roboter ausschließlich auf planen Flächen zum Einsatz kommt.

Da sich die Objekte einschließlich des Roboters bewegen oder ihre Gestalt verändern, werden sie als über der Zeit veränderliche Funktionen des Typs $\mathbb{R}_0^+ \to \alpha$ dargestellt (wobei

```
/*@ requires \valid_ptr(v, 0, len)
  @ ensures (\forall int i; 0 <= i &&
  @                  i < len --> v[i] <= \result) &&
  @          (glob == \old(glob) + 1)  */
  int f(int *v, int len);
```

Abbildung 1: Einfaches Beispiel für die Spezifikation einer Funktion `f`; der Rückgabewert wird als größer als die Elemente des Feldes `v` spezifiziert.

$\mathbb{R}_0^+$ als Zeit interpretiert wird, und α ein beliebiger Typ ist.) Wir nennen solche Funktionen *Verhalten*. So liefert etwa die Funktion $robot : \mathbb{R}_0^+ \to \mathbb{R}^2$ die Position des Roboters zu jedem Zeitpunkt. Eigenschaften wie die Stetigkeit von Funktionen lassen sich in Isabelle/HOL einfach spezifizieren, und die globale Sicherheitsanforderung läßt sich leicht vereinfacht wie folgt formalisieren:

$$[\forall t,\, t'.\, \forall o \in Obs.\, o(t) = o(t')] \Longrightarrow \forall t.\, \forall o \in Obs.\, robot(t) \notin o(t) \qquad (1)$$

wobei Obs die Menge aller Funktionen $\mathbb{R}_0^+ \to \mathcal{P}(\mathbb{R}^2)$ ist, die Objekte repräsentieren.

Einbettung des Kontrollprogramms in das Weltmodell. Um nachzuweisen, dass *robot* die Eigenschaft (1) erfüllt, müssen wir die Tatsache ausdrücken, dass der Roboter durch sein Kontrollprogramm gesteuert wird, welches in unserem Fall ein reaktives System ist. Dies bedeutet, dass die eigentliche Sicherheitsroutine alle N Millisekunden mit den aktuell gelesenen Sensorwerten aufgerufen wird, und einen Rückgabewert berechnet, der angibt, ob ein Nothalt eingeleitet werden muss. Die Zykluslänge wird durch die Scanfrequenz des Laserscanners bestimmt. Hierfür werden *diskrete Verhaltensfunktionen* des Typs $\mathbb{N} \to \alpha$ eingeführt, wobei jede natürliche Zahl genau einen Zyklus repräsentiert. Diskrete Verhalten sind das Verbindungsglied zwischen Weltmodell und konkretem Sicherungsprogramm, anhand derer wir spezifizieren können, welchen Einfluss das Sicherungsprogramm auf die reale Welt hat. Insbesondere das diskrete Verhalten $stop$ nimmt Einfluss auf das Roboterverhalten $robot$: ab dem Zyklus, in dem ein Nothalt emittiert wird ($stop(n) = True$), bremst der Roboter bis zum Stillstand mit definierter Verzögerung ab.

Am unteren Ende der Abstraktionshierarchie im Entwicklungsprozess steht der konkrete Code, welcher den Sicherungsalgorithmus implementiert. Dieser ist im vorliegenden Projekt in C nach den MISRA Richtlinien [MIS04] geschrieben, und hat im Wesentlichen sicherzustellen, dass der Roboter darf seine Fahrt nur genau dann fortsetzt, wenn anhand der verfügbaren Sensordaten sichergestellt werden kann, dass innerhalb des nächsten Zyklus keine Notbremsung eingeleitet werden muss. Um Eigenschaften von C-Programmen in unserem Rahmenwerk zu formulieren und zu beweisen, betten wir wie JML [BCC+05] oder Caduceus [FM04] Spezifikationen in den Quellcode ein; Abb. 1 zeigt ein Beispiel.

Die formale Verbindung zwischen dem Ausgabeverhalten *stop* und dem C-Sicherungsprogramm wird hergestellt, indem wir die Semantik von C sowie der eingebetteten Spezifikationssprache in Isabelle/HOL definieren. Die Semantik eines C-Programms ist dann ein Zustandstransformer, d. h. eine Funktion, die einen Eingabezustand erwartet —dieser

beinhaltet die Programmvariablen sowie aktuelle Sensordaten— und einen Ausgabezustand sowie ein Resultat liefert. Ein Zustandstransformer lässt sich leicht in ein diskretes Verhalten übersetzen: in jedem Zyklus wird der Transformer mit dem Zustand des letzten Durchlaufs aufgerufen, wobei dieser Vorzustand noch um die aktuellen Sensordaten ergänzt wird. Damit ist die Einbindung des C-Sicherungsprogramms und seiner Eigenschaften in das Weltmodell gelungen: die Semantik des Programms ist Teil der Definition des Nothalt-Verhaltens $stop$.

4 Schlussbemerkungen

In unser Herangehensweise wird die gesamte Entwicklung, von den Sicherheitsanforderungen bis hin zur Implementation, innerhalb von Isabelle/HOL formalisiert. Die Vorteile dieser Methodologie sind zum ersten, dass die Spezifikation sehr abstrakt formuliert werden kann, mit der Betonung auf den Konzepten der Domänenmodellierung und nicht der Implementation, so dass die Zertifikation auf dieser Ebene die Validität der Sicherheitsanforderung diskutieren und früh eventuelle konzeptionelle Fehler offenlegen kann; zum zweiten können Beweise maschinell geprüft werden, so dass sich die Zertifikation auf diese Kernfragestellungen konzentrieren kann; und zum dritten werden durch den formalen Beweis von Eigenschaften oft versteckte Annahmen in Spezifikation und Implementation aufgedeckt. Es kann also formal bewiesen werden, dass die Implementierung wirklich für eine korrekte Ausführung der Sicherheitsfunktion Kollisionsvermeidung sorgt. Alle Annahmen über das Roboterverhalten, die Umgebung, oder auch das Ausführungsmodell des C-Codes sind explizit in Isabelle/HOL formuliert. Fehlende Annahmen werden offengelegt, wenn Beweise nicht vollendet werden können.

Ein Nachteil dieser Herangehensweise ist, dass formale Beweise vergleichsweise aufwändig sind. Das kann vielleicht dadurch aufgewogen werden, dass einmal bewiesene Theoreme für Projekte mit der gleichen Anwendungsdomäne wiederverwendet werden können. Ferner ist die Akzeptanz durch die Zertifizierungsbehörden noch nicht gegeben. Hier ist zu hoffen, dass das SAMS-Projekt, in dem die Zertifizierung durch den TÜV Süd durchgeführt wird, Brücken schlagen kann, denn letztendlich können alle Seiten (Zertifizierer, Implementierer und sogar Endnutzer) von den Vorteilen profitieren.

Literatur

[BCC+05] Lilian Burdy, Yoonsik Cheon, David Cok, Michael Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino und Erik Poll. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer*, 7(3):212– 232, June 2005.

[FM04] Jean-Christophe Filliâtre und Claude Marché. Multi-Prover Verification of C Programs. In *Sixth International Conference on Formal Engineering Methods (ICFEM)*, Jgg. 3308 of *Lecture Notes in Computer Science*, Seiten 15–29, Seattle, November 2004. Springer.

[MIS04] MISRA-C: 2004. Guidelines for the use of the C language in critical systems., 2004.

Positionsstatement: Zertifizierungsargumentationen mit mathematischer Präzision

Hardi Hungar

F&E-Bereich SC
OFFIS – Institut für Informatik
Escherweg 2
26121 Oldenburg
hungar@offis.de

Abstract: In gegenwärtiger Praxis haben Argumentationen, mit denen Zertifizierungen sicherheitskritischer Systeme begründet werden, die Qualität juristischer Beweise. Hier wird die Meinung zum Ausdruck gebracht, dass diese langfristig in einer stärkeren, mathematischen Präzision zu erbringen sein werden.

1 Ausgangslage

In den der Zertifizierung unterliegenden Bereichen erreicht die Qualität der Produkte unter Sicherheitsgesichtspunkten bereits heute eine sehr hohe Qualität: Unfälle sind weit seltener auf technisches Versagen zurückzuführen als auf Unzulänglichkeiten des Systemzusammenhangs oder der Bedienung. Gleichwohl ist vom rigorosen Standpunkt aus die Qualität der Zertifizierungszusicherung als eingeschränkt zu bewerten. Dies wird auch in der Formulierung von Normen deutlich, wo schwächere Argumentationen in Ermangelung anwendbarer formalerer Methoden zugelassen werden. Vor dem Hintergrund der enormen Fortschritte, welche es in der Anwendung rigoroser Techniken auf Systeme nichttrivialer Größenordnung in den letzten Jahrzehnten gegeben hat, wird deutlich, dass der Druck auf die Entwicklung, ein noch höheres Qualitätsargument zu liefern, zunehmen wird: Weil es möglich (und wirtschaftlich vertretbar) sein wird, werden in ferner, jedoch absehbarer Zukunft, ein großer Teil von Sicherheitsargumentationen im mathematischen Sinn formal geführt werden müssen.

2 Indikationen

Es gibt eine Vielzahl von Beispielen, welche den Fortschritt exakter Methoden bei der Sicherheits- oder Zuverlässigkeitsanalyse dokumentieren. So werden im Hardwarebereich formale Methoden schon routinemäßig angewendet. Ein herausragendes Beispiel ist die Verifikation des Prozessorkerns des Tricore 2 der Firma Infineon innerhalb des Verisoft-Projektes.

Aber auch auf Anwendungsebene haben Verifikationsmethoden Einzug gehalten, wie der erfolgreiche Einsatz von modelcheckbasierten Analysewerkzeugen etwa der Firma OSC für die Entwurfswerkzeuge Statemate, Stateflow und ASCET zeigt.

3 Wege zur Präzision

An der prinzipiellen Machbar formaler Verifikation besteht schon lange kein Zweifel mehr. Jedoch ist genauso klar, dass als unrealistisch anzusehen ist, mit Hilfe automatischer Beweiser die Ergebnisse heute gängiger Entwurfsprozesse behandeln zu können. Eine Vielzahl von Maßnahmen erscheinen erforderlich, um die Vision kompletter Verifikation Wirklichkeit werden zu lassen. Der Verfasser dieser Zeilen erhebt nicht den Anspruch, den Weg zu kennen und anderen aufzeigen zu können. Jedoch lassen sich etliche Maßnahmen aufzählen, die mit einiger Wahrscheinlichkeit einen wichtigen Beitrag liefern können.

- Verwendung abgesicherter Architekturen, etwa für Redundanzkonzepte, Zeitverteilung, zuverlässige Kommunikation
- Dedizierte Programmiersprachen und Modellierungswerkzeuge, welche die Beherrschbarkeit der Artefakte und nicht die Mächtigkeit und Vielfalt der Konzepte in den Vordergrund stellen.
- Hardwarearchitekturen und Betriebssysteme mit vorhersagbarem Ressourcenverbrauch und insbesondere zuverlässigem Zeitverhalten
- Komponenten mit zugesicherten Eigenschaften (Wiederverwendung, Modularisierung)
- Verzahnung automatischer und interaktiver Beweismethoden
- Evidenzproduzierende Verifikation (z.B. Modelchecker, welche Fehlerfreiheit nicht nur feststellen, sondern auch begründen)
- Verifikation von Codegenerierung und Übersetzung (z.B. Übersetzer**lauf**verifikation für Produktcode, im Gegensatz zu schwieriger zu verifizierenden Übersetzern)

4 Zeithorizont

Eine Voraussage zu treffen ist sicherlich wenig mehr als schlichtes Raten, jedoch erscheinen zwanzig Jahre als nicht unrealistischer Zeitraum für die weitgehende Ablösung heutiger Vorgehensweisen durch formal abgesicherte, wenn dieses Thema eine Gemeinschaft von Forschern findet, welche sich ihres annehmen und welche für ihre Arbeiten Rückhalt bei öffentlichen und privaten Organisationen finden.

Positionsstatement: Über die Rolle von formalem Beweis und Entwicklung im Zertifizierungsprozeß

Christoph Lüth

Deutsches Forschungszentrum für Künstliche Intelligenz, Bremen
christoph.lueth@dfki.de

1 Formaler Beweis

Unter einem *formalen* Beweis verstehen wir einen Beweis in formaler Logik, der vollkommen mechanisch durch die Anwendung der Schlußregeln der Logik mit Hilfe eines *Theorembeweiser* genannten Programmes in einem Rechner geführt werden kann. Ein formaler Beweis kann entweder durch vollautomatische Suchverfahren gefunden werden, oder muss mittels zusätzlicher Eingaben des Benutzers konstruiert werden (beispielsweise durch Angabe von Induktionsvariablen oder Invarianten). Die Korrektheit auch sehr großer Beweise folgt aus der Korrektheit der Implementation der Regeln der Logik in dem Theorembeweiser, welche sehr klein und damit einem sorgfältigen Review zugänglich ist. Nachdem die Korrektheit des Theorembeweisers einmal etabliert wurde, folgt die Korrektheit aller mit diesem Beweiser durchgeführten Beweise.

2 Anwendungen

Die Verfahren zum automatischen oder halbautomatischen Beweis von Behauptungen haben in den letzten zehn Jahren erstaunliche Fortschritte gezeigt. Wo vor einigen Jahren mathematische Trivialitäten mühsam gezeigt wurden, können heute Beweise mathematisch anspruchsvoller Sachverhalte geführt werden. In Verbindung mit der zunehmend verfügbaren Rechenleistung ergeben sich Anwendungen für diese Techniken im Bereich der Mathematik, Hardware- und Softwareverifikation.

Im Bereich der Mathematik hat sich das Flyspeck-Projekt [Hal] die vollständige Formalisierung des Beweises der seit Jahrhunderten offenen Keplerschen Behauptung (über die optimale Packdichte von Kugeln) zum Ziel gesetzt. Im Bereich der Hardware-Entwicklung, wo einzelne Fehler massive Kosten nach sich ziehen können, ist die maschinelle Verifikation bereits Stand der Technik, zum Beispiel bei der Fließkommaarithmetik der Pentium-Mikroprozessorfamilie der Firma Intel [KK03]. In dem vom Bundesministerium für Bildung und Forschung geförderten Projekt *Verisoft* [Ver] werden fünf konkrete, nichttriviale Anwendungen (u.a. ein Automobil-Notrufsystem) durchgängig, d.h. von der Hardware bis

zur Anwendung, formal verifiziert. Auch im industriellen Softwarebereich findet formale Verifikation und Entwicklung zunehmend Anwendung (z.B. in der Verifikation von Treibern für das Betriebssystem Windows [Mic]).

3 Formaler Beweis im Zertifizierungsprozess

Ein Theorembeweiser kann im Zertifizierungsprozess die Rolle einer unabhängigen Prüfinstanz für die Korrektheit der vorgebrachten Beweise spielen. Für den Entwickler hat dies den Vorteil, dass die Prüfung auf Korrektheit während der Entwicklung stattfindet, und Beweisfehler nicht erst während einer Prüfung aufgedeckt werden. Der Zertifizierer kann sich hingegen auf die Kernfragen des Zertifizierungsprozesses konzentrieren. Ein großer Vorteil formaler Beweise ist ferner, dass alle Seitenbedingungen explizit gemacht werden müssen — der Theorembeweiser wird keine stillschweigenden Annahmen akzeptieren. Dadurch kann sich der Zertifizierungsprozess auf die Angemessenheit und Rechtfertigung aller dieser getroffenen Nebenbedingungen (die beispielsweise die Einsatzumgebung des zu zertifizierenden Systems beschreiben), sowie die Angemessenheit und Vollständigkeit der Sicherheitsanforderungen fokussieren, während die mechanische Überpüfung der Korrektheit von Berechungen und Beweisen an einen Rechner delegiert wird.

Formale Beweistechniken können keinesfalls die bisherigen Methoden und Ansätze (wie Modellchecking oder Testen) vollständig ersetzen, aber sie können eine wichtige *Ergänzung* sein. Ein weiterer Vorteil ist die *Wiederverwendung* formaler Beweise: zum einen können einmal bewiesene Theoreme immer wieder verwendet werden, zum anderen sind formale Beweise wesentlich robuster gegenüber Änderungen. Muss beispielsweise die Anforderungsspezifikation geändert werden, wird der Theorembeweiser genau auf die Stellen im Beweis hinweisen, die dadurch ungültig werden; es ist unmöglich, dass kleine Änderungen unbemerkt große Wirkung nach sich ziehen, solange der Beweis formal geprüft wird. Werkzeuge zur Verwaltung von Änderungen in formalen Beweisen wurden in den letzten Jahren entwickelt [Hut00].

Literatur

[Hal] Thomas C. Hales. The Flyspeck Project Page. `http://www.math.pitt.edu/~thales/flyspeck/index.html`.

[Hut00] Dieter Hutter. Management of change in verification systems. In *Proc. 15th IEEE Int. Conf. on Automated Software Engineering, ASE-2000*, S. 23– 34. IEEE Comp. Society, 2000.

[KK03] Roope Kaivola und Katherine R. Kohatsu. Proof engineering in the large: formal verification of Pentium 4 floating-point divider. *Int. J. Software Tools for Technology Transfer*, 4(3):323–334, 2003.

[Mic] Microsoft Corp. Windows Driver Foundation. Web site at `http://www.microsoft.com/whdc/driver/wdf/default.mspx`.

[Ver] The VeriSoft project. Web site at `http://www.verisoft.de/`.

Schwachpunkte und Grenzen gegenwärtiger Methoden

Theodor Tempelmeier

Fakultät für Informatik
Hochschule Rosenheim
Hochschulstraße 1
83024 Rosenheim
tempelmeier@fh-rosenheim.de

Abstract: In der Entwicklung eingebetteter, sicherheitskritischer Systeme werden vor allem in der industriellen Praxis zum Teil Methoden eingesetzt, die nach Meinung des Autors Schwachpunkte aufweisen. Ziel dieses Beitrags ist die Überwindung dieser Schwachpunkte, oder wenigstens eine Schärfung des Bewusstseins bezüglich dieser Schwachpunkte, um eine Verbesserung der Situation zu erreichen.

1 Gegenwärtige Situation

Aktuell ist die Entwicklungs- und Nachweissituation computergestützter sicherheitskritischer Systeme stark geprägt durch Ansätze wie die modellgetriebene Entwicklung und die automatische Code-Generierung sowie durch kontinuierliche Verbesserungen im Bereich der mathematischen, auch automatischen, Korrektheitsbeweise. Selbstverständlich sind diese Ideen, ohne dass weitere Argumente nötig wären, auf das entschiedenste zu bejahen. Aber dennoch zeigt die industrielle Praxis Schwachpunkte, deren Überwindung relativ kurzfristig eine Verbesserung der Situation ermöglichen könnte. Und bezüglich der mathematischen Korrektheitsbeweise sind deren inhärente Grenzen zu beachten.

2 Modellgetriebene Entwicklung und automatische Code-Erzeugung

Sofern aus Modellen sicherheitsrelevanter Code generiert werden soll, ist für die Modellierungssprache eine *klar definierte Semantik* zu fordern; dies ist bedauerlicherweise nicht immer gegeben. Zur automatischen Code-Erzeugung muss deutlich gemacht werden, dass der Grundsatz „Von nichts kommt nichts“ auch hier gilt. Im Grunde wird nichts generiert, sondern nur der (grafische) Ausgangscode in Zielcode transformiert. Dabei fließen die womöglich unzureichende Semantik des Modells sowie eine Vielzahl weiterer Informationen ein. Diese *Transformation* muss deshalb ebenfalls *präzise definiert* sein. Es erscheint abwegig, wenn eine Reihe von Modellierungselementen vermieden werden muss, damit korrekter Zielcode entsteht [Te07], oder wenn die Code-Erzeugung durch ca. 50 Schalter (entsprechend 2^{50} Möglichkeiten) gesteuert werden kann [Te07]. Schließlich ist noch die Modellierung selbst korrekt anzuwenden. Die Mo-

delle müssen einen höheren *Abstraktionsgrad* als die Implementierung aufweisen, ansonsten wäre nichts gewonnen. Beispiele für eine *Modellierungsinversion* (d.h. das Modell ist auf demselben oder einem schlechteren Abstraktionsniveau als die Implementierung) sind Oder-Gatter im grafischen Modell oder auch UML-Packages für Zielcode mit einem höherwertigen Paketkonzept wie z.B. in der Programmiersprache Ada.

3 Programmiersprachen

Trotz der vorherrschenden Codeerzeugungseuphorie soll das Thema Programmiersprache nicht unbeachtet bleiben. Die Sprachen *C und C++* können für sicherheitskritische Anwendungen nur als *hoffnungslos ungeeignet* eingestuft werden. Die geringfügigen Verbesserungen in C++ (gelegentlich als „objektorientierter Assembler" verspottet) reichen nicht aus. *Java* ist aufgrund seiner dynamischen Laufzeitaspekte *ebenfalls ungeeignet*. Man kann wohl Java, vielleicht auch C++, so weit abändern, dass sie für sicherheitskritische Anwendungen geeignet werden. Damit würde aber nur eine längst geleistete Arbeit nachvollzogen. Denn die Programmiersprache *Ada* ist für sicherheitskritische Anwendungen von vornherein *hervorragend geeignet*, auch wenn sie außerhalb der Luft- und Raumfahrtindustrie weitgehend ignoriert wird. Die Stärken von Ada sind ein überlegener Synchronisationsmechanismus (CSP-ähnlich vs. Monitor-ähnlich in Java), ein überlegenes Package-Konzept (wichtig für „teile und herrsche"), das sog. Ravenscar-Profil für sicheres Tasking, integrierte Festpunkttypen sowie eine strenge Typisierung (wie in der IEC 61508 vorgegeben). Gerade vor dem letztgenannten Punkt ist der weitverbreitete Einsatz von C als erschreckend und beunruhigend abzulehnen.

4 Korrektheitsbeweise

Strenge Korrektheitsbeweise von Programmen sind als zusätzliche Absicherung absolut zu bejahen. Man muss jedoch auch deren Grenzen erkennen. *Die Beweise gelten nur in Bezug auf die getroffenen Annahmen.* Als Beispiel könnten Veränderungen in der Umwelt, z.B. verschüttete Schmierseife, den Bremsweg eines autonomen Roboters vergrößern, so dass die „bewiesene" Systemsicherheit gefährdet wäre. Oder eine Flugzeugsteuerung könnte nach der Landung den Umkehrschub verweigern, weil sich die Räder des Fahrwerks wegen Aquaplaning nicht drehen; der Korrektheitsbeweis unter der falschen Annahme „Räder drehen sich nicht → kein Bodenkontakt" ist dann nutzlos. Abhilfe bringt hier nur eine extreme Sorgfalt in der Berücksichtigung und Beurteilung aller Voraussetzungen und Annahmen des Gesamtsystems. Leider hat nämlich *Albert Einstein* recht: *„Insofern sich die Sätze der Mathematik auf die Wirklichkeit beziehen, sind sie nicht sicher, und insofern sie sicher sind, beziehen sie sich nicht auf die Wirklichkeit."*

Literaturverzeichnis

[Te07] Tempelmeier, T.: Untersuchung verschiedener gängiger Modellierungswerkzeuge und Modellierungsmethoden, Interner Bericht, Hochschule Rosenheim, 2007.

Ergebnisse des Workshops

„Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme“

Hardi Hungar[1], Erwin Reyzl[2]

[1] OFFIS, F&E-Bereich Sicherheitskritische Systeme
Escherweg 2, 26121 Oldenburg
hungar@offis.de

[2] Siemens AG, CT SE 1,
Otto-Hahn-Ring 6, 81730 München
erwin.reyzl@siemens.com

1 Generelles

Etwa 30 Teilnehmer setzten sich in reger Diskussion mit neun Beiträgen sowie sechs Positionsstatements zu Erfahrungen und Herausforderungen der Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme auseinander. Vertreten waren vor allem Verkehrs- und Automatisierungstechnik, dabei kamen die Teilnehmer aus Industrie und Forschung.

2 Vorträge

Nach der Einführung und dem Vortrag von Hr. Prof. Parnas zum Thema „Certification: What, how, and how Confidently“ wurden in drei Sitzungen die Themen „Werkzeuge und Programmierung“, „Entwurfsparadigmen“ sowie „Formale Methoden“ angesprochen.

Bei „Werkzeuge und Programmierung“ zeigte sich, wie bereits heute modellbasierte Werkzeugen mittels qualifizierter/zertifizierter Codegeneration und –übersetzung zumindest Teile der Entwicklung sicherheitskritischer Software verbessern. Mit der Spezifizierung und Standardisierung eines echtzeit- und sicherheitsfähigen Java dürfte in Zukunft auch Java neben C/C++ und Ada Anwendung finden. Die Wichtigkeit domänenspezifischer Modellierung in der Praxis der Stellwerkstechnik sowie die Unverzichtbarkeit eines sicheren (secure) Codings für sichere (safe) Systeme war Thema unter „Entwurfsparadigmen“. Die abschließenden Vorträge zeigten nicht nur die Perspektiven „Formaler Methoden“ im Umfeld sicherheitsrelevanter Systeme, sondern belegten, wie diese bereits heute in der Praxis der Zertifizierung Einsatz finden können.

3 Positionsstatements

In den Positionsstatements zeigten sich unterschiedliche, teilweise konträre Einschätzungen insbesondere der Tragweite modellbasierter und formaler Ansätze. Bereits heute in speziellen ausgewählten Situationen praktisch genutzt, wurde ihnen einerseits das Potential zugesprochen, eine ingenieursgerechte und industrielle Softwareentwicklung zu befähigen. Benannte Perspektiven waren fundierte Nachvollziehbarkeit, frühere und damit kostengünstigere Prüfung auf höherer Abstraktionsebene, Ersatz traditioneller Tests durch Korrektheitsnachweise, sowie zuletzt die Überwindung juristischer Argumentation und betriebswirtschaftlicher Überlegung durch mathematisch belegbare Gewissheit.

Andererseits wurden derzeit, und möglicherweise zukünftig weiter ungelöste, Beschränkungen und Unzulänglichkeiten gesehen, die einer wirklich generellen Anwendung derartiger Ansätze entgegenstehen. Unzulänglichkeiten der vorwiegenden Programmiersprachen, Semantikschwächen gängiger Modellierungssprachen, Hürden bei Einführung und Anwendung formaler Sprachen, Unbeherrschbarkeit von Variabilität und Komplexität. Darüberhinaus stelle die Langlebigkeit der Systeme mit lange Wartungsverpflichtungen neue Werkzeuge und Methoden vor ebenso hohe Verpflichtungen.

4 Diskussion

In der Diskussion wurden auch die Positionsstatements nochmals aufgegriffen. Die Vielfalt der oft konträren Stellungnahmen findet sich in der Folge frei und unkommentiert aufgelistet.

Vor- und Nachteile der Programmiersprachen waren erneut Thema: sicheres Ada, performantes und weitverbreites C/C++, programmierfreundliches Java.

- Java wird bereits bei der Mars Rover Mission Einsatz finden. Gute Gründe für JAVA bestehen z.B. bei sicherheitskritischen Komponenten, deren Umfeld selbst in JAVA realisiert ist.

- Weiterhin limitierte Rechenressourcen erlaubten auch heutzutage oft nur C; ein ADA Frachtsystem für das Spaceshuttle musste schlicht in C nachprogrammiert werden.

- Inakzeptabel ist andererseits, dass grundlegende Probleme mit Integer-Overflow und Arraygrenzen bei C/’C++ im Gegensatz zu Ada nicht abgefangen werden, wo dies doch mit heutigen leistungsfähigen Prozessoren leicht lösbar sein sollte.

- Nur durch strikte Beschränkung des Sprachumfangs werden C/C++ (MISRA) sowie Java (SCSJ) safety-tauglich werden.

Bisher fanden formale Methoden wenig Eingang in die Zertifizierung selbst. Durch formale Verifikation wird eher eine zusätzliche Prüfungsmöglichkeit als eine Eliminierung von Tests erwartet. Verschiedene Prüfungen werden damit erst machbar, allerdings kann dadurch ein leichterer Sicherheitsbeweis trotzdem nicht erwartet werden. Ein weit verbreiteter Irrtum ist zudem, dass durch wie auch immer bessere Prüftechniken Sicherheit selbst hineingeprüft wird.

- „Sicherheit kann nicht in ein System hineingetestet werden." Sicherheit ist ein Konzept, das im Entwurf behandelt werden muss. Sicherheit kann nicht durch noch so massives Testen erreicht werden.

- Formale Methoden bringen zusätzliche Qualität. Dennoch bleiben Tests unverzichtbar. Niemand würde sich mit gutem Gefühl in ein Fahrzeug (Auto, Zug, Flugzeug) setzen, das nicht auch getestet wurde.

- Formale Prüfungen basieren auf Modellen, beruhen auf Abstraktion und erfassen damit naturgemäß nicht das gesamte System. Abstraktion und Bewertung erfordern selbst Sorgfalt und Sachverstand, um nicht Wesentliches unberücksichtigt zu lassen.

- Anforderungen und ihre vollständige Erhebung sind die Grundlage, auf dem das Sicherheitskonzepts und seine Prüfung basiert. Neben Domänenkompetenz bedarf es eines systematischen Vorgehens.

- Mitunter sind Systeme so komplex, dass sie sich einem vernünftigen und vollständigen Spezifizieren entziehen. Am Ende der Entwicklung (Systemintegration) treten oft eine Diskrepanz zwischen Kundenintention und Resultat zu Tage.

Techniken und Tools allein werden nicht zum Erfolg führen, ohne den Faktor Mensch zu integrieren. Werkzeuge und Methoden werden aber oft überschätzt. Fundierte Ausbildung und industrielle Erfahrung, Kompetenz und Sorgfalt, sowie organisierte Abläufe sind unverzichtbare Vertrauensgrundlage:

- Man sollte nie vergessen, dass Zertifizierung dem Vermitteln von Vertrauen dient und berechtigtes Vertrauen voraussetzt. Alle Regulierungen setzen Sorgfalt voraus und sollen hauptsächlich sicherstellen, dass keine Aspekte übersehen werden.

- Eine der großen Herausforderung ist es, berechtigtes Vertrauen zukünftig durch nachvollziehbare Gewissheit zu ersetzen; kurz gesagt Mathematik statt Jura.

- Weitgehend formale Beweise im mathematischen Sinne könnten in fernerer Zukunft (20 Jahre?) die juristischen Argumentationen ersetzen.

- Wichtig ist eine präzise Kommunikation zwischen den Beteiligten vom Kunden/Auftraggeber bis zur Abnahme/Zertifizierung.

- Der Drang nach Tools wie DOORS kann mit dem Griff nach dem Strohhalm verglichen werden. Man hat damit ein Werkzeug, mit dem man Anforderungen aufschreiben kann, aber mehr auch nicht.

- UML wird als interdisziplinäres Verständigungsmittel oft weit überschätzt. Fachrichtungsübergreifend funktioniert es weit weniger als beim Austausch zwischen Softwareexperten.

- Formale Methoden werden oft überbewertet. Ohne ingenieurstechnischen Sachverstand wird sich ihr Potential nicht erschließen.

- Generierung und Automatisierung manueller Schritte werden Softwareentwicklung von ihrem derzeitigen Manufaktur-/Kunststatus auf den Level industrieller Fertigung heben.

- Angesichts katastrophaler Fehler stellt sich die Frage, wie Experten befähigt werden können, Konsequenzen aus Ketten- und Mehrfachfehlern früher und besser vorherzusehen und damit im Sicherheitskonzept wirklich berücksichtigen zu können.

- Die Erzeugung direkter persönliche Betroffenheit bewirkt erstaunliche Fortschritte. China verpflichtete Manager des Jahr-2000-Problems zum Fliegen zum Jahreswechsel, und bewältigte damit den schleppenden Fortschritt.

- Auch das Space-Shuttle-Programm setzt auf das persönliche Element. Hier müssen sich die Manager den direkten Vertrauensfragen der Astronauten stellen: „Haben Sie wirklich alles für meine sichere Rückkehr getan?“.

- Systems Engineering kann man genauso selten studieren wie Software Engineering, oder noch seltener.

Unumstritten ist, dass es eines konsequenten System- und Software-Engineerings bedarf, beginnend mit den Anforderungen bis hin zur Freigabe, um Qualität und Vertrauenswürdigkeit kritischer Systeme auch in Zukunft sicherzustellen. Der Trend zur Integration auch Web-basierter Techniken in kritischen Anlagen macht fachübergreifende Zusammenarbeit zwischen funktionaler (Safety) und informationstechnischer Sicherheit (Security) unumgänglich.

Insbesondere im Zuge wachsender Komplexität und eines zunehmenden Softwareanteils werden modellbasierte Entwicklung, formale Ansätze für Design und Verifikation, Code-Generierung und Test-Automatisierung an Bedeutung gewinnen. Dies kann zwar nie Sorgfalt und Fachkompetenz ersetzen. Aber nur so werden die verantwortlichen Experten die steigenden Anforderung an Nachvollziehbarkeit, Dokumentation, Tests und Prüfungen mit vertretbarem Aufwand und Kosten erfüllen können.

5 Fazit

Die sich an die Vorträge und Positionsstatements anschließende lebhafte Diskussion zeigte ein reges Interesse aus dem Teilnehmerkreis. Zutage traten viel Übereinstimmung in grundlegenden Fragen, manche abweichende, pointierte Meinung und insgesamt einen Konsens, dass sich das Gebiet in einer schnellen Entwicklung befinde, wo sich noch kein Ende abzeichne. So erscheint eine Fortführung der Veranstaltung in den kommenden Jahren, vielleicht dann auch unter Beteiligung der dieses Mal nicht vertretenen Industriesparten und insbesondere der Prüfungs- und Zulassungsbehörden, als sehr wünschenswert

Testmethoden für Software - Von der Forschung in die Praxis

Inhaltsverzeichnis

2. Workshop Testmethoden für Software - Von der Forschung in die Praxis 353
Fevzi Belli, Christof J. Budnik, Andreas Spillner

Modellbasierter Test mit FitNesse – Ein Erfahrungsbericht aus der Praxis 355
Dehla Sokenou

Plattformübergreifende funktionale Tests durch GUI-Abstraktion 359
Benno Markiewicz, Holger Flemig, Karsten Weicker

Vergleich einer graphenbasierten Methode mit der Klassifikationsbaummethode für die Testfallermittlung anhand einer automotiven Fallstudie 367
Friedrich Beidinger, Axel Hollmann, Markus Kleinselbeck, Wolf Ritschel

A TTCN-3-based Web Service Test Framework 375
Edith Werner, Jens Grabowski, Stefan Troschütz, Benjamin Zeiss

Die Einführung eines wieder verwendbaren Testframeworks in der Sozialversicherung 383
Armin Beer

Test Case Structuring and Execution Control in an Integration Framework for Heterogeneous Automatic Software Tests 387
Andreas Ganser, Holger Schackmann, Horst Lichter, Heinz-Josef Schlebusch

Ein Framework für die Testautomatisierung bei Fahrer-Assistenz-Systemen 395
Vladimir Entin, Klaus Meyer-Wegener

Using UML Environment Models for Test Case Generation 399
Maritta Heisel, Denis Hatebur, Thomas Santen, Dirk Seifert

Glassboxtest zur Testsuite-Optimierung 407
Rainer Schmidberger

Automatic Generation of Optimized Integration Test Data by Genetic Algorithms 415
Florin Pinte, Francesca Saglietti, Norbert Oster

Automatic Test Case Generation from UML Models and OCL Expressions 423
Stephan Weißleder, Dehla Sokenou

A Domain Specific Language for Uncovering Abstract Protocols and Testing Message Scenarios 427
Tim Reichert, Dominikus Herzberg

Path Constraint Construction with Lookahead 431
Ralf Gerlich

Simplifying the Description and Application of Tests 435
Daniel Brenner, Colin Atkinson

Statistical Testing within the Scope of the System Validation of a RIS/PACS System 439
Stefan Holpp, Helmut Goetz, Winfried Dulz

Static Bug Detection Through Analysis of Inconsistent Clones 443
Elmar Juergens, Benjamin Hummel, Florian Deissenboeck, Martin Feilkas

2. Workshop Testmethoden für Software - Von der Forschung in die Praxis

Fevzi Belli[1], Christof J. Budnik[2], Andreas Spillner[3]

[1] Fachgebiet Angewandte Datentechnik, Universität Paderborn,
Warburger Str. 100, 33098 Paderborn
belli@upb.de

[2] Siemens Corporate Research, Inc., Software Engineering Department
755 College Road East, Princeton, NJ 08540, USA
christof.budnik@siemens.com

[3] Zentrum für Informatik und Medientechnologien, Hochschule Bremen,
Flughafenallee 10, 28199 Bremen
spillner@informatik.hs-bremen.de

Folgekosten fehlerhafter Software-Produkte demonstrieren eindrucksvoll, dass die systematische Entwicklung von Software in einer für den Einsatz ausreichenden Qualität auch nach Jahrzehnten Forschung nicht die Regel darstellt. Eine Ursache ist sicherlich in den Projekten selbst zu finden; häufig werden bekannte Qualitätssicherungsmaßnahmen nicht, oder nicht in der notwendigen Form eingesetzt, um Termin- und Budgetanforderungen gerecht zu werden. Sicherlich ist aber auch eine Ursache, dass ein Transfer von der Forschung in die Praxis nur unzureichend erfolgt. Als Beispiel seien hier die vielfältigen Forschungsaktivitäten im Bereich systematisches Testen genannt und die sehr eingeschränkte Anwendungen entsprechender Testmethoden in der Praxis. Gründe für einen unzureichenden Transfer von der Forschung in die Praxis können sein:

- Unzureichender Einblick in die Anforderungen aus der Praxis
- Unzureichende Übermittlung/Verbreitung und somit Akzeptanzprobleme vorhandener formaler Testmethoden
- Unzutreffende Annahmen beim Einsatz der Testmethoden
- Hohe Komplexität der Testmethoden und -werkzeuge
- Inadäquate Werkzeugunterstützung bei großen Projekten bzw. Systemen
- Ungenaue Kostenabschätzungen
- Vernachlässigung von Schulungen und Fortbildungen
- Fehlende Management-Unterstützung und ungenügende Ressourcenbereitstellung
- Fehlende, theoretisch untermauerte Testmethoden spezieller Domänen

Ziel des Workshops war die stärkere Verzahnung zwischen Forschung und der Praxis im Bereich Softwaretesten. In diesem ganztägigen Workshop stellten Teilnehmerinnen und Teilnehmer aus Forschung und Praxis ihre Arbeiten einem breiteren Fachpublikum vor und tauschten Erfahrungen aus, was das Potential einer fruchtbaren Kooperation durch einen Wissenstransfer in beide Richtungen fördert. Wir erhielten erfreulich viele Beiträge aus Firmen, Hochschulen und Forschungseinrichtungen. Jeder Beitrag wurde von mindestens drei Mitgliedern des Programmkomitees begutachtet, diese sind:

Colin Atkinson, Uni Mannheim

Sami Beydeda, ZIVIT Bonn

Mario Dal Cin, Uni Erlangen-Nürnberg

Klaus Didrich, Siemens AG

Dimitris Dranidis, CITY College, GR

Christof Fetzer, TU Dresden

Sabine Glesner, TU Berlin

Jens Grabowski, Uni Göttingen

Wolfgang Grieskamp, Microsoft Research

Karl-Erwin Großpietsch, Fraunhofer-Institut (IAIS)

Volker Gruhn, Uni Leipzig

Walter Gutjahr, Uni Wien

Uwe Hehn, method park Software AG

Maritta Heisel, Uni Duisburg-Essen

Thomas Illgen, Carmeq GmbH

Oliver Jack, FH Jena

Bruno Legeard, LEIRIOS Technologies

Michael Leuschel, Uni Düsseldorf

Peter Liggesmeyer, Uni Kaiserslautern

Tilo Linz, imbus AG

Zoltan Adam Mann, sd&m AG

Dirk Meyerhoff, Schüco Internat. KG

Ralf Reussner, Uni Karlsruhe

Francesca Saglietti, Uni Erlangen

Franz Schweiggert, Uni Ulm

Dehla Sokenou, GEBIT Solutions GmbH

Andreas Ulrich, Siemens AG

Joachim Wegener, Berner & Mattner Systemtechnik GmbH

Mario Winter, FH Köln

Eric Wong, UT Dallas

Die konstruktiven Kritiken der Gutachter führten zweifelsohne zur Verbesserung der eingereichten Beiträge. Daher danken wir den Mitgliedern des Programmkomitees. Wir danken natürlich auch den Teilnehmern und Teilnehmerinnen des Workshops und den Organisatoren (insbes. Herrn Walid Maalej, TU München); ohne sie wäre dieser Workshop nicht zustande gekommen. Unser besonderer Dank gilt Herrn Axel Hollmann, Uni. Paderborn, der uns (als Bindeglied zur Tagungsleitung SE 2008) tatkräftig unterstützt hat.

Fevzi Belli, Christof J. Budnik, Andreas Spillner

Modellbasierter Test mit FitNesse – Ein Erfahrungsbericht aus der Praxis

Dehla Sokenou
GEBIT Solutions, Koenigsallee 75b, D-14193 Berlin
dehla.sokenou@gebit.de

Abstract: Das Papier stellt ein modellgetriebenes Softwareprojekt vor, in dem systematisch mit Hilfe des Testframeworks FitNesse getestet wurde. Dabei werden Vor- und Nachteile des Werkzeugs im praktischen Einsatz beleuchtet.

1 Motivation

In großen Projekten stellt sich meist die Frage nach einem Testwerkzeug, da manuelle Tests in ausreichender Zahl nicht durchgeführt werden können. Einerseits soll dieses Testwerkzeug flexibel genug sein, um es auf die speziellen Bedürfnisse des Projekts anzupassen. Andererseits soll es möglichst einfach in seiner Bedienung sein. Zusätzliche Anforderung in unserem Projektkontext war die Anwendbarkeit einmal entwickelter Testtechniken auch in anderen Projekten, die mit dem gleichen modellgetriebenen Framework für Softwareentwicklung implementiert werden. Das Papier stellt ein modellgetriebenes Projekt vor, in dem systematisch mit Hilfe des FitNesse-Framework [Fit] getestet wurde. Zunächst wird das Projekt vorgestellt, anschließend der Einsatz von FitNesse im Projekt beschrieben. Nach einer Diskussion verwandter Arbeiten wird zum Schluss eine Zusammenfassung unserer Erfahrungen sowie ein Ausblick auf aktuelle und zukünftige Projekte gegeben.

2 Das Projekt

Im Projekt wurde eine Java Enterprise-Anwendung entwickelt, die auf einem JBoss-Applikationsserver läuft und über einen Apache-Webserver Zugriff per Web erlaubt. Die Webseiten werden mit Hilfe von JSF erzeugt. Zusätzlich werden verschiedene mobile Geräte per WLAN eingebunden. Der Projektumfang umfasste bei einer Entwicklungszeit von 2 Jahren ca. 10 Personenjahre (davon ca. 10% für FitNesse-Tests), ca. 2000 Java-Klassen (davon 120 Business-Objekt-Klassen), ca. 220.000 Lines of Code und etwa 100 Datenbanktabellen. Die Anwendung wurde auf der Basis des GEBIT-eigenen TREND-Frameworks [GEB] für modellgetriebene Entwicklung realisiert, dabei wurden knapp 100 Use Cases umgesetzt. Das Projekt wurde –auch aufgrund der FitNesse-Tests– im vorgegebenen Zeitrahmen abgeschlossen und ist inzwischen produktiv im Einsatz.

3 FitNesse im Projekt

Bereits zu Beginn des Projekts wurde nach einem neuen Testwerkzeug gesucht, das sich gut in den modellgetriebenen Softwareentwicklungsprozess einbinden ließ. Dabei erschien FitNesse als geeignet, zumindest einen großen Teil der Anforderungen zu erfüllen.

Warum FitNesse? FitNesse ist durch sein offenes Konzept sehr flexibel. Testfälle werden mit Hilfe eines Wikis als Tabellen definiert. Die Auswertung erfolgt mit Hilfe des dahinterliegenden Fit-Frameworks [MC05], entweder über das Wiki oder mit Hilfe eines Testrunners, der z.B. aus Eclipse heraus gestartet werden kann. Die Testfälle werden mit Hilfe sogenannter Fixtures ausgewertet, kleinen Stücken von Programmcode, die die Verbindung zwischen den zunächst abstrakten Testfällen und dem zu testenden System implementieren. Neben einigen rudimentären Build-in-Fixtures erlaubt FitNesse die Definition eigener Fixtures. Diese Fixtures können entweder produktspezifisch definiert werden oder auch in allgemeiner Form. FitNesse ist nicht beschränkt auf eine Art des Tests, sondern unterstützt durch unterschiedliche Fixture z.B. sowohl Oberflächen- als auch Backend-Tests.

Dies sprach für den Einsatz von FitNesse, da das Framework flexibel erweiterbar genug ist, um eine generische Anbindung zu ermöglichen, die nicht auf das aktuelle Softwareprodukt beschränkt ist. Stattdessen wurden die Fixtures so implementiert, dass sie geeignet sind, alle Produkte zu testen, die modellgetrieben mit TREND implementiert sind. Es wurden etwa 30 generische Fixtures im Projektkontext implementiert, Beispiele sind: *Starten von Use Cases oder Aktivitäten*, *Ausführen von Aktivitätsschritten*, *Prüfen von Werten in einem Objekt oder der Datenbank*. Zusätzlich gibt es projektspezifische Fixtures. Der Großteil der Tests ($>$80%) wurde jedoch mit den generischen Fixtures realisiert.

Im Projekt war das Testen zunächst ein Randthema. Sehr schnell wurde jedoch erkannt, dass die Qualität einer Software mit der vorliegenden Komplexität nur durch entwicklungsbegleitende automatisierte Tests nachhaltig zu sichern ist. Deshalb wurde nach etwa einem halben Jahr ein Testspezialist in das Projekt geholt, der die wenigen bereits vorhandenen Fixtures in umfassender Weise erweiterte und ergänzte und systematisch die Funktionen der Software auf Basis der Modelle und der Spezifikation durch Tests überprüfte. Die Erstellung erfolgte zwar manuell, aber auf Basis der Modelle.

Es wurden zwei Arten von Testsuiten erstellt. Der “rote Faden” erreicht eine 100% Überdeckung von Use Cases, Aktivitäten und Aktivitätsschritten, aber als Black-Box-Test systembedingt keine 100% Überdeckung der Implementierung. Er umfasst 17450 Prüfungen in 350 Tests, die mit einer Laufzeit von 630 Sekunden durchgeführt werden. Eine weitere Testsuite überprüft spezielle Funktionen, z.B. bisher nicht erreichten oder kritischen Code. Diese umfasst 15000 Prüfungen in 220 Tests mit einer Laufzeit von 940 Sekunden. Im Laufe des Projekts waren zwangsläufig Änderungen an den Testfällen erforderlich. Diese waren bedingt durch Änderungen an den Modellen und Änderungen der anderen implementierten Teile. Die Testfälle wurden innerhalb der Projektlaufzeit etwa 500 Mal angepasst, wobei eine Anpassung meist nur einen oder wenige Testfälle betraf. Bei der Menge der Testfälle ist eine Anpassung in dieser Größenordnung aus unserer Erfahrung normal und wird z.B. bei Einsatz von Werkzeugen zum reinen Oberflächentest noch übertroffen.

Nach Einführung der systematischen Tests durch einen speziell dafür eingesetzten Exper-

ten verbesserte sich die Qualität der sich in der Entwicklung befindlichen Software rapide. Durch die gute Abdeckung der Funktionen durch Tests wurden neue Fehler schnell gefunden. Besonders gut geeignet waren sie für den Regressionstest, d.h. sie verhinderten das Einbringen neuer Fehler in bestehenden Softwareteilen, die bereits eine gute Qualität aufwiesen. Ziel war stets der fehlerfreie Testdurchlauf. Liefen die FitNesse-Testfälle ohne Fehler durch, so bestand so viel Vertrauen in die Stabilität der Software, dass dem Kunden auch außerhalb der Releases ein Prototyp zur Verfügung gestellt werden konnte. Von den ca. 800 Fehlern im Issuetracking-System sind ca. 500 Fehler durch die FitNesse-Tests initial aufgedeckt worden. Die anderen Fehler wurden z.T. durch Entwicklertests, z.T. vom Kunden in den Prototypen und Releases gefunden.

4 Verwandte Arbeiten

Abbot, ein klassisches Capture-Replay-Werkzeug, wird ebenfalls in einigen unserer Projekte eingesetzt. Abbot wurde in den 4 Jahren seines Einsatzes in unseren Projekten kontinuierlich um TREND- und Projektspezifika erweitert. Im Gegensatz zu FitNesse ist Abbot für den Tester zunächst einfacher zu bedienen. FitNesse bietet jedoch ein abstrakteres, technisch einfacher aufgebautes Framework für die Implementierung eigener Fixtures. Die Testfälle sind mit FitNesse abstrakter definiert, da nicht einzelne Klicks auf der Benutzeroberfläche protokolliert werden, sondern eine Anweisung im Testfall erst durch die dahinterliegende Fixture ausgewertet wird.

Fallstudien zu TTCN-3 (vgl. [BSSG04, DGNP04]) wurden meist in der Domäne der Telekommunikationssysteme durchgeführt. Die Ergebnisse dieser Fallstudien lassen sich in der Regel nicht auf die Domäne der betrieblichen Anwendungssysteme übertragen, die nicht den reaktiven Character von Telekommunikationssystemen haben. Zudem benötigen modellbasierte Testverfahren, die auf TTCN-3 basieren, meist spezielle Testmodelle, modelliert z.B. mit dem UML Testing Profile [UTP04]. Unsere Tests basieren ausschließlich auf den vorhandenen Designmodellen. Auch für den Leirios Test Designer [BBC+06], einem Werkzeug für den modellbasierten Test, ist die Definition spezieller Testmodelle nötig, ebenfalls in UML, doch ohne das UML Testing Profile zu nutzen.

5 Zusammenfassung und Ausblick

FitNesse überzeugt durch seine offene Architektur. Zusätzlich zu den Fixtures haben wir das Wiki (Syntax und neue Funktionen wie Anlegen neuer Testsuiten) erweitert. Die generische Entwicklung von Fixture ist relativ einfach möglich, so dass FitNesse leicht an andere Frameworks angebunden werden kann oder für den Test der verschiedensten Anwendungen geeignet ist. Gleichzeitig muss man auf die Nachteile von FitNesse hinweisen. FitNesse ist durch die Definition von Testfällen im Wiki weniger intuitiv bedienbar als viele andere Testwerkzeuge. Zum Testen muss immer zunächst der FitNesse-Server gestartet werden.Es gibt wenig Modularisierungs- und Strukturierungsmöglichkeiten. Testfälle

können zwar von anderen Testsuiten referenziert werden, die Testfälle sind jedoch streng hierarchisch strukturiert und ihre Reihenfolge nur durch ihre Benennung festgelegt. Integrationstests lassen sich nur schwierig realisieren, wenn das Zusammenspiel mehrerer Prozesse getestet werden soll. Der Test von Oberflächen ist relativ mühselig, da alle Schritte manuell in Wiki-Tabellen beschrieben werden müssen, es fehlt hier noch an geeigneter Werkzeugunterstützung. Eines der größten Mankos für die Einarbeitung in FitNesse ist die mangelnde Dokumentation des offenen Quellcodes, da dies ein Hindernis bei der (für den praktikablen Einsatz notwendigen) Erweiterung des Testframeworks darstellt.

Nach den guten Erfahrungen in dem beschriebenen Pilotprojekt für den FitNesse-Einsatz wird FitNesse auch in aktuellen Projekten eingesetzt. Wir haben dabei gute Erfahrungen gemacht, wenn mindestens ein Entwickler mit Testerfahrung in einem Projekt ausschließlich für die FitNesse-Tests zuständig ist und Testen nicht nur "nebenbei" passiert. Neben der Erstellung der Testfälle an sich ist dieser auch für die Erweiterung der vorhandenen Fixtures verantwortlich. Aktuell werden wenn möglich FitNesse-Tests direkt von Beginn eines Projekts eingesetzt, was ein hohes Qualitätsniveau bereits vom Projektstart an gewährleistet, ohne den Gesamtaufwand signifikant zu erhöhen. Ergänzend werden mit JUnit [BG00] implementierte Entwicklertests definiert. Zukünftig soll auch die Erstellung von Testfällen unterstützt werden durch ein zusätzliches TREND-Werkzeug zur Aufzeichnung von FitNesse-Testfällen und die automatische Generierung von FitNesse-Testfällen aus den Modellen, wobei bereits bestehende Techniken verwendet werden [Sok06].

Literatur

[BBC+06] E. Bernard, F. Bouquet, A. Charbonnier, B. Legeard, F. Peureux, M. Utting und E. Torreborre. Model-Based Testing from UML Models. In *MOTES-Workshop, Informatik 2006*, number P-94 in LNI, Seiten 223–230, Dresden, Germany, 2006.

[BG00] K. Beck und E. Gamma. Test Infected: Programmers Love Writing Tests. In Dwight Deugo, Hrsg., *More Java Gems*, Seiten 357–376. Cambridge University Press, 2000.

[BSSG04] M. Born, I. Schieferdecker, P. Santos und H.-G. Gross. Model-Driven Development and Testing - A Case Study. In *1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, Enschede, Netherlands, 2004.

[DGNP04] Z. R. Dai, J. Grabowski, H. Neukirchen und H. Pals. From Design to Test with UML - Applied to a Roaming Algorithm for Bluetooth Devices. In *TestCom'2004*, number 2978 in LNCS, Oxford, UK, 2004. Springer-Verlag.

[Fit] FitNesse Test Framework. http://fitnesse.org/.

[GEB] GEBIT TREND Framework for Java. http://www.gebit.de/trend.

[MC05] R. Mugridge und W. Cunningham. *Fit for Developing Software - Framework for Integrated Tests*. Prentice Hall, 2005.

[Sok06] D. Sokenou. *UML-basierter Klassen- und Integrationstest objektorientierter Programme*. Dissertation, Technische Universität Berlin, Germany, 2006.

[UTP04] *UML 2.0 Testing Profile Specification, V. 1.0*. OMG, *http://www.omg.org*, 2004.

Plattformübergreifende funktionale Tests durch GUI-Abstraktion

Benno Markiewicz[1] Holger Flemig[2] Karsten Weicker[1]

[1] Fachbereich Informatik, Mathematik und Naturwissenschaften, HTWK Leipzig
Postfach 301166, 04251 Leipzig
bmarkiew,weicker@imn.htwk-leipzig.de

[2] itCampus Software- und Systemhaus GmbH
Nonnenstr. 42, 04229 Leipzig
h.flemig@itcampus.de

Abstract: Sowohl Unit-Test-basierte Ansätze als auch Capture/Replay-Techniken (C/R) stehen vor großen Herausforderungen, wenn plattformunabhängiges GUI-Testen gefordert wird, da Testskripte in der Regel nicht für verschiedenen GUI-Frontends geeignet bzw. konvertierbar sind. Das Ziel der hier vorgestellten Studie ist die Demonstration, dass über GUI-Widget- und Event-Abstraktion portierbare Testskripte erstellt werden können. Ferner werden die Grenzen und möglichen Probleme eines solchen Ansatzes aufgezeigt.

1 Motivation

In der Softwareentwicklung steigt der Testaufwand mit zunehmender Systemkomplexität sowohl in der Funktionslogik als auch in der Benutzungsoberfläche. Dadurch kann der Testanteil der Entwicklungskosten auf 50% ansteigen [Me02].

Wird zudem mit unterschiedlichen Instanzen der Benutzungsoberfläche gearbeitet, bedeutet dies häufig mehrfachen Testaufwand für die Präsentationsschicht. Dies kann beispielsweise bei der Portierung eines Fat-Client-Systems zum Web-Client, der nachträglichen Unterstützung eines anderen Betriebssystems oder der bewussten Entscheidung für unterschiedliche Technologien auf verschiedenen Plattformen auftreten. Bei gezielter plattformübergreifender Entwicklung bieten einige GUI-Toolkits entsprechende Unterstützung, aber bei nachträglichen Entscheidungen oder der parallelen Entwicklung Web- und Client-basierter Versionen besteht in der Regel keine Möglichkeit, im Rahmen der Testautomatisierung Testskripte wiederzuverwenden.

In dieser Arbeit wird vorgestellt wie durch Widget-Abstraktion mit unveränderten Testskripten über GUI-Toolkits hinweg gearbeitet werden kann. Dabei konzentrieren wir uns zunächst im Rahmen der Machbarkeitsstudie auf manuell an der Software durchgeführte GUI-Tests, die aufgezeichnet und später auf

unterschiedlichen Plattformen automatisiert wiederholt werden. Der hier vorgestellte Ansatz basiert zu großen Teilen auf der Arbeit [Ma07].

2 Einordnung des Ansatzes

Aufgrund ihrer eventorientierten Natur besitzen GUIs nicht nur eine enorme Vielzahl an Zuständen sondern Events ändern den Systemzustand beständig durch Seiteneffekte. Nicht zuletzt durch große Freiheiten bei der Systembedienung ist rein manuelles Testen inakzeptabel und sehr fehleranfällig. Für die Automatisierung unterscheidet man Unit-Test-Ansätze und Capture/Replay-Systeme.

Der erste Ansatz mit Unit-Tests ist meist auf einzelne GUI-Toolkits beschränkt. In den Testskripten wird entweder direkt der Zustand der Widgets manipuliert (HL, High-Level-Ansatz) oder es werden Maus- und Tastatur-Ereignisse (LL, Low-Level-Ansatz) ausgelöst. Daneben gibt es noch Mischformen. Als Prüfpunkte dienen Assertions, d.h. Ist-Soll-Vergleiche der Widgeteigenschaften. Besonders für das Java-basierte Swing-Toolkit gibt es eine Vielzahl von Frameworks (z.B. Jemmy, JFCUnit, UISpec4J oder Abbot.) Problematisch ist die fehlende Kompatibilität der Testskripte zwischen den Frameworks ebenso wie ein hoher Initialaufwand.

Die Capture/Replay-Systeme (C/R) werden in die Kommunikation zwischen GUI und dem zu testenden System eingeklinkt. Dort angemeldete Funktionen (Eventlistener) warten auf auftretende Ereignisse und Testskripte können dann durch Interaktion eines menschlichen Testers mit der Software erstellt werden. Wie im obigen Low-Level-Ansatz werden diese Aktionen beim Abspielen des Testskripts wieder auf der Software durchgeführt. Die Skripte sind entweder compilierbarer/interpretierbarer Quellcode oder in einem XML-Format gespeichert. Einige kommerzielle Systeme unterstützen zwar mehrere GUI-Toolkits, aber die erstellten Testskripte sind bis auf Ausnahmen nicht kompatibel zueinander. Die Entwicklung von C/R-Systemen erfolgte von positions- und bildschirmfotobasierten bis hin zu den heute üblichen widgetbasierten Ansätzen. Der positionsbasierte Ansatz ist kritisch bei Änderungen in Layout oder Bildschirmauflösung. Widgetbasierte Ansätze nutzen heuristische Algorithmen oder eindeutige Benennungen zur Identifikation von Widgets. Hierbei werden die Ereignisse pro Widget erfasst bzw. ausgelöst. Einmal gefundene Widgets können mittels High- oder Low-Level-Events gesteuert und deren Zustandseigenschaften ausgelesen werden. Testskripte sind hierbei nicht so stark von Layoutänderungen abhängig wie bei dem positionsbasierten Ansatz.

Abbildung 1 zeigt die Vorgehensweise des hier präsentierten Ansatzes. Im Gegensatz zu derzeit im Einsatz befindlichen Systemen sind die Testskripte unabhängig vom GUI-Toolkit – trotz der Unterstützung mehrerer GUI-Toolkits sind i.d.R. die aufgenommenen Testskripte toolkitspezifisch, wodurch Eigenentwicklungen oder Widgets aus Komponentenbibliotheken nicht unterstützt werden. Die hierfür notwendige Widget-Abstraktion wird in den folgenden Abschnitten ausführlich behandelt. Die Testskripte enthalten Zusicherungen, die im Rahmen

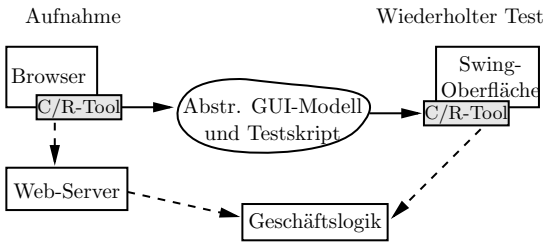


Abbildung 1: Konzept des hier vorgestellten plattformübergreifenden Testansatzes

des Aufnahmevorgangs vom Benutzer angegeben werden – sie bestimmen letztendlich Erfolg bzw. Misserfolg des Tests.

Andere aktuelle Themen der Testautomatisierung wie Modellierung durch Zustandsautomaten [MPS01], vollständige Automatisierung [MX04] oder formale GUI-Spezifikation [BRRP05] bleiben im Rahmen der zunächst einfachen Umsetzung unberücksichtigt.

3 Widgetabstraktion

In [Ma07] wurden die verfügbaren Widgets von 12 GUI-Toolkits (MFC, wxWidgets, GTK+, Swing, AWT, SWT, WinForms, QT, XUL, WPF sowie HTML und XForms als Beschreibungssprachen) gegenübergestellt und die einfachen Widgettypen in Tabelle 1 als kleinsten gemeinsamen Nenner definiert, welche den Kern der Abstraktion bilden.

Technologisch stützt sich unser Ansatz auf Beschreibungssprachen zur UI-Generierung. Der *UI Markup Language* (UIML) [Ph00, Ha07] sind die Arten und Typen der Widgeteigenschaften entlehnt. Die *User Interface Automation* (UIA) [Ha05] lieferte das Steuermusterprinzip, bei dem die funktionalen Eigenschaften der Widgets durch Interfaces beschrieben werden.

Die wichtigsten unterstützten Steuermuster sind: ExpandPattern (Beschreibung einer ausklappbaren Funktion wie in ComboBoxen oder TreeNodes), ImagePattern (Grafiken und Icons), InvokePattern (ausführbare Elemente wie Buttons), SelectionPattern (selektierbare Elemente wie Elemente in ListBoxen oder Tabellenzellen), TextValuePattern (Elemente mit Texteigenschaft wie Textfelder, Labels, Buttons), TextEditPattern (Elemente deren Texteigenschaft veränderbar ist), TogglePattern (umschaltbare Elemente wie Check- oder RadioBoxen), RangePattern (Elemente mit einer fortlaufenden Auswahl von Zahlenwerten, wie Trackbars, Fortschrittsbalken oder Scrollbars) und WindowPattern (Fensterelemente). Eine freie Kombinierbarkeit und beliebige Hierarchien erlauben auch die Beschreibung komplexer zusammengesetzter Widgets. Statt der High-Level-Events in UIML und UIA wurde hier mit Low-Level-Events gearbeitet, da nur diese durch die Browser-Anbindung Meex der Firma itCampus unterstützt wurden

und darüberhinaus mit Blick auf JavaScript und AJAX auch alle Events im Browser tatsächlich ausgelöst werden sollten.

Zwei Beispiele sollen das Konzept illustrieren. Abbildung 2 zeigt oben eine ComboBox, die das ExpandPattern unterstützt und deren Kindelemente Text enthalten (TextPattern), anwählbar sind (SelectionPattern) und hier zusätzlich mit einer CheckBox versehen sind (TogglePattern). Komplexere Widgets wie eine Baumstruktur in Abbildung 2 (unten) sind ebenfalls abbildbar. Die Unterelemente besitzen jeweils ein Textlabel und ein Icon, sind selektierbar, umschaltbar und im Falle weiterer Unterelemente auch noch ausklappbar.

Tabelle 1: Verfügbarkeit von plattformübergreifenden Widgets (Auswahl)

		HTML 4.01	XForms 1.1	MFC 7.0	wxWidgets 2.8.0	GTK+ 2.11.0	Swing 1.6	AWT 1.1.x	SWT	WinForms	QT 4.2	XUL (M1.8; FF1.5)	WPF/ XAML
Aktion													
	Button	+	+	+	+	+	+	+	+	+	+	+	+
	Button mit Graphik	+	-	+	+	+	+	-	+	+	+	+	+
Anzeige													
	Animation	+	+	+	+	+	+	-	-	+	+	+	+
	Fortschrittsbalken	-	-	+	+	+	+	-	+	+	+	+	+
	Graphik	+	+	+	+	+	+	-	+	+	+	+	+
Text	Textanzeige	+	+	+	+	+	+	+	+	+	+	+	+
	Label	+	+	+	+	+	+	+	+	+	+	+	+
	HTML-Viewer	+	+	?	+	?	+	-	+	+	+	+	+
Auswahl													
Einfach	ComboBox	-	-	+	+	+	+	-	+	+	+	+	+
	ComboBox (readonly)	+	-	+	+	+	+	+	+	+	+	+	+
	ListBox	+	+	+	+	+	+	+	+	+	+	+	+
	RadioButton	+	-	+	+	+	+	+	+	+	+	+	+
Mehrfach	CheckBox	+	-	+	+	+	+	+	+	+	+	+	+
	ListBox (N)	+	+	+	+	+	+	+	+	+	+	+	+
Zahl	Slider/ Scale	-	-	+	+	+	+	?	+	+	+	-	+
Basis-Widget													
	Vaterelement aller Widgets	-	-	+	+	+	+	+	+	+	+	-	+
Composites													
Widgets	Gruppierung	+	+	+	+	+	+	+	+	+	+	+	+
	Panel mit Scrollleisten	+	-	?	+	+	+	+	+	+	+	+	+
	Statusleiste	+	-	+	+	+	-	-	-	+	+	+	+
	Toolbar	-	-	+	+	+	+	-	+	+	+	+	+
	Toolbar-Elemente	-	-	+	+	+	+	-	+	+	+	+	+
Dialog													
Allgemein	Fenster	+	-	+	+	+	+	+	+	+	+	+	+
	MessageDialog (modal)	+	-	+	+	+	+	-	+	+	+	+	+
Datei	„Datei öffnen"-Dialog	-	-	+	+	+	+	+	+	+	+	+	+
	„Datei speichern"-Dialog	-	-	+	+	+	+	+	+	+	+	+	+
Eingabe													
Text	Passwort	+	+	+	+	-	+	+	+	+	+	+	+
	RTF-Textfeld	-	-	+	+	-	+	-	+	+	+	+	+
	Text (einzeilig)	+	+	+	+	+	+	+	+	+	+	+	+
	Text (mehrzeilig)	+	+	+	+	+	+	+	+	+	+	+	+
Hilfe/ Feedback/ Fehlermeldung													
	Hint / Tooltip	+	+	+	+	+	+	-	+	+	+	+	+
Komplexe Widgets													
Baum	Baum-Struktur	-	-	+	+	+	+	-	+	+	+	+	+
ListView	ListView	-	-	+	+	+	-	-	+	+	+	+	+
Tabelle	Tabelle (kein Layout)	+	-	-	+	+	+	-	+	+	+	+	?
Navigation													
	Menu-Struktur	-	-	+	+	+	+	+	+	+	+	+	+
	Menu-Elemente	-	-	+	+	+	+	+	+	+	+	+	+
	Menu-Leiste	-	-	?	+	+	+	+	+	+	+	+	+
	Registerreiter	-	+	+	+	+	+	-	+	+	+	+	+
	Scrollleiste einzeln	-	-	+	+	+	+	+	+	+	+	+	+
Sonstiges													
	Separator (Linien)	+	+	+	+	+	-	-	+	+	+	+	+

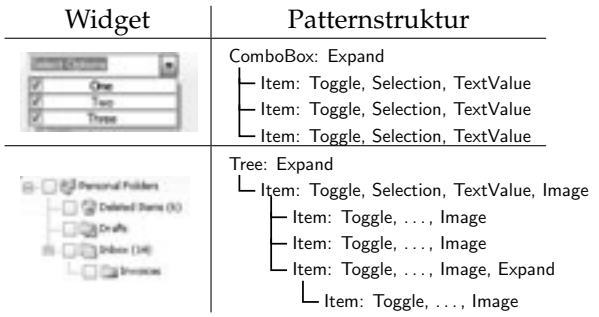


Abbildung 2: Vergleich der Widgets mit ihrer Patternstruktur: Combobox und Baum

Abbildung 3: Screenshot der Demoanwendungen (Mozilla und Swing)

4 Prototypische Umsetzung

Im Protoyp werden das Swing-Toolkit sowie einfache dynamische Webanwendungen unterstützt – eine Testanwendung ist in Abbildung 3 dargestellt. Der Zugriff auf die Widgets innerhalb einer Webanwendung erfolgt über eine einheitliche API, die eine von itCampus entwickelte Bibliothek zum Ansprechen der Mozilla- und IE-Engine bietet. Hierbei wird die Mozilla-Gecko-Engine bzw. die IE-ActiveX-Komponente in einem eigenen Browser integriert, so dass globale Eventlistener gesetzt werden können und der DOM-Baum verfügbar ist. Auf die Widgets und Events einer Swing-Anwendung kann mit Swing-Bordmitteln des JDK zugegriffen werden. Voraussetzung ist die Ausführung des Testobjektes in der gleichen virtuellen Java-Maschine wie das Testsystem.

Beim Aufnahmeprozess ist das Abfangen von Events mit den genannten Anbindungen möglich. Beim Eintreffen eines Events werden Informationen über das Widget, auf dem das Event ausgelöst wurde, mitgeliefert. Dies wird verwendet, um eine spätere Zuordnung von Widget und Event und so die Widgetfindung zu gewährleisten. Für den Prototyp müssen Widgets eindeutig benannt sein. Es folgt die Umwandlung der Eventinformationen zu einem abstrakten Eventformat. Die-

ses wird mit Daten zur Identifikation des Widgets verknüpft. Zur Laufzeit können zusätzlich Zusicherungen eingeführt werden, die ebenfalls als abstrakte Events gespeichert werden. Es resultiert ein XML-Testskript.

Beim Wiedergabeprozess wird das Testskript schrittweise ausgeführt. Aus jedem einzelnen Event werden die Informationen zur Widgetidentifikation extrahiert und das passende Widget in der Liste der verfügbaren Widgets gesucht. Das abstrakte Event wird in ein toolkitspezifisches Event umgewandelt und auf dem Widget ausgeführt. Bei gespeicherten Zusicherungen findet ein Ist-Soll-Vergleich statt.

Die einzelnen Widgets können in verschiedenen GUI-Toolkits unterschiedlich realisiert sein, was mit einer Tabelle in Swing und HTML verdeutlicht werden soll. Die Swing-Tabelle wird mit einem Datenarray gefüllt und die impliziten Elemente werden durch das Tabellenwidget selbst dargestellt. Bei der HTML-Tabelle sind die Elemente direkt durch `<tr>`- und `<td>`-Tag adressierbar. Für jedes zu unterstützende Widget jedes Toolkits ist eine programmtechnische Abbildung der Struktur auf ein Widgetobjekt mit den gewünschten Unterelementen und Steuermustern nötig.

Während die Hauptelemente der Oberfläche über einen eindeutigen Bezeichner identifiziert werden, wird bei Listen und ComboBoxen ein Index, bei Tabellen eine zweidimensionale Indizierung sowie bei Bäumen eine um Hierarchieinformation erweiterte x-Dimension benötigt. Abbildung 4 zeigt eine Liste, eine Tablle, ein

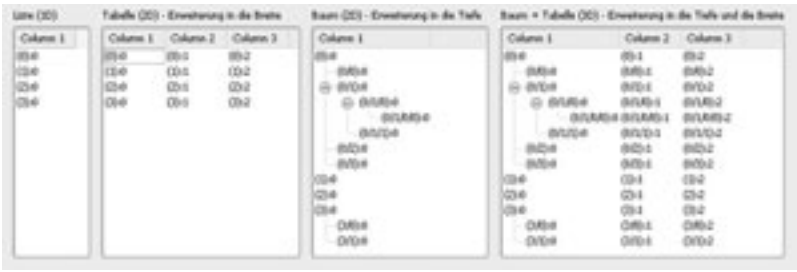


Abbildung 4: Identifikatoren in zusammengesetzten Widgets

Baum sowie eine Baumtabelle. Jedes Unterelement ist dort mit seinem eindeutigen Identifikator beschriftet. Mit diesem Hilfsmittel sind einfache sowie komplexere Unterwidgetstrukturen adressierbar. Tabellen als Unterelemente in Tabellen wurden dabei im Prototyp ausgeschlossen.

5 Erfahrungen

Im ersten Prototypen wurden einige, für ein vollständig funktionstüchtiges C/R-Werkzeug wichtige Funktionen nicht berücksichtigt. So wird beispielweise das stark systemspezifische Drag&Drop innerhalb und zwischen Anwendungen nicht

unterstützt. Dasselbe gilt für spezifische vom Fenstermanager abhängige Funktionen wie Mini-/Maximieren sowie mehrere parallele modale Fenster. Bedingt durch die Einschränkung des Prototypen auf ein Toolkit zur Aufnahmezeit, sind Fremddialoge wie *Datei öffnen* auch nicht aufzeichenbar, da kein Zugriff auf die Widgetinterna möglich ist.

Wie bereits dargelegt sind die unterschiedlichen inneren Strukturen der Widgets in den Toolkits kritisch, da funktional identische Widgets durch Aufsplittung und Neugruppierung von Unterelementen vielfach modellierbar sind. Für die Kompatibilität der Testskripte müssen hier in gewisser Weise Abbildungen bzw. Normalisierungen auf feste Typen durchgeführt werden. Dies ist aber kontraproduktiv zum Ansatz, Widgets über Steuermuster frei zu typisieren. Es kommt hinzu, dass bei nicht klar definierten Strukturen die Identifikation von Unterelementen sehr schwierig wird. Die zuvor vorgestellten Identifikatoren auf Indexwert-Basis sind nicht stabil genug, um größere Änderungen in der Struktur korrekt abbilden zu können. Durch die Einschränkung der Verschachtelungstiefe auf Eins sind u.a. unbenannte Layout-Tabellen (z.B. in HTML) nicht abbildbar. Ein weiteres Problem neben der Vereinheitlichung der unterstützten Patterns pro Widget ist die Laufzeitabhängigkeit von Patterns. Diese können je nach Widgetzustand verschiedene sich ändernde Patterns unterstützen, was die Fragilität der Testskripte beeinflusst.

Ein weiterer kritischer Punkt sind die Low-Level-Events. Sie sind systemspezifischer als zunächst angenommen. Besonders die Abhängigkeit und das Timingverhalten der Events zum Auslösen von nachfolgenden High-Level-Events ist stark spezifisch. Es sind für jedes Toolkit und jedes System von Neuem Anpassungen nötig. Aber auch die Maus-Events mit zu einem Widget relativen Koordination sind problematisch. So kann nicht garantiert werden, dass das Verhalten des Nutzers bei der Aufnahme 1:1 wiedergegeben werden kann. Von einem abstrakten Widget kann nicht eindeutig auf die korrekten Koordinaten geschlossen werden. Insbesondere nicht, wenn der Abspielvorgang in einem anderen Toolkit mit anders dargestellten Widgets (und -strukturen) durchgeführt wird. Der Ansatz stellt insofern eine Mischform von widgetbasierten HL-Test über Steuermuster und positionsbasierten LL-Events dar. Denkbar wäre ein Einsatz von High-Level-Events, was in einer plattformübergreifenden Umsetzung von UIA münden würde. Aber mit der Aufnahme und dem Auslösen von High-Level-Event wird von der korrekten Nutzerinteraktion weggegangen. Hierbei wird die dahintersteckende GUI-Logik aufgerufen und nicht der Nutzer mit seinen Eingabegeräten simuliert.

6 Fazit

Für den plattformübergreifenden Einsatz von Testskripten in C/R-Systemen wurde eine Widget- und Eventabstraktion konzipiert. User Interface Automation und das generische Vokabular von UIML stellten hierfür die Basis dar. Größter Vorteil des Ansatzes ist die freie Typisierung von Widgets mittels der Kombination von Steuermustern, die jeweils einen einzelnen funktionalen Aspekt kapseln. Theore-

tisch sind beliebige GUI-Strukturen und zusammengesetzte Widgets abbildbar. Die Interaktionsmöglichkeiten der Widgets wurden mit Low-Level-Maus- und Tastaturevents abgebildet. Im Anschluss an die Konzeption wurde die Implementation eines in großen Teilen vollständigen Capture/Replay-Systems vorgenommen. Der Prototyp unterstützt exemplarisch eine Untermenge von Widgets des Swing-Toolkits und von Web-Formularelementen (HTML). Problematisch sind dabei die uneinheitlichen inneren Strukturen von GUIs und Widgets in den Toolkits, so dass es kritisch ist, untergeordnete Widgets zu adressieren. Ebensowenig sind die Laufzeitabhängigkeit der Steuermuster sowie die starke Systemnähe von LL-Events gelöst. Diese Problematik muss in den Griff bekommen werden. Die Abkehr von LL-Event und die Zukehr zu HL-Events ist zwar eingeschränkt möglich, aber so kann der korrekte funktionale Test über die GUIs – über die Nachahmung des Nutzerverhalten – nicht garantiert werden. Überdacht werden muss ebenfalls die Unterstützung von komplexen zusammengesetzten Widgetstrukturen. Wir werden mit Interesse beobachten, ob das Vorbild UIA auch bald für die Testautomatisierung instrumentalisiert wird und inwieweit Abstraktion von GUI-Komponenten Einzug in kommerzielle C/R-Systeme hält.

Literatur

[BRRP05] Berstel, J., Reghizzi, S. C., Roussel, G., und Pietro, P. S.: A scalable formal method for design and automatic checking of user interfaces. *ACM Trans. Softw. Eng. Methodol.* 14(2):124–167. 2005.

[Ha05] Haverty, R. New accessibility model for Microsoft Windows and cross platform development. online http://msdn2.microsoft.com/. 2005.

[Ha07] Harmonia. Generic UIML vocabulary specification. online http://www.uiml.org/. 2007.

[Ma07] Markiewicz, B.: Abstraktion von GUI-Widgets zur Optimierung der Plattformunterstützung von C/R-Testautomatisierungen. Master's thesis. HTWK Leipzig, FbIMN. Leipzig. 2007.

[Me02] Memon, A. M.: GUI testing: Pitfalls and process. *Computer*. 35(8):87–88. 2002.

[MPS01] Memon, A. M., Pollack, M. E., und Soffa, M. L.: Hierarchical GUI test case generation using automated planning. *IEEE Trans. Softw. Eng.* 27(2):144–155. 2001.

[MX04] Memon, A. M. und Xie, Q.: Empirical evaluation of the fault-detection effectiveness of smoke regression test cases for GUI-based software. In: *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*. S. 8–17. Washington, DC, USA. 2004. IEEE Computer Society.

[Ph00] Phanouriou, C.: *UIML: A Device-Independent User Interface Markup Language*. PhD thesis. Virginia Tech. Blacksburg, VA, USA. 2000.

Vergleich einer graphenbasierten Methode mit der Klassifikationsbaummethode für die Testfallermittlung anhand einer automotiven Fallstudie

Friedrich Beidinger[1], Axel Hollmann[2], Markus Kleinselbeck[1], Dr. Wolf Ritschel[1]

[1] Hella KGaA Hueck & Co.,
Rixbecker Straße 75, 59552 Lippstadt
{Markus.Kleinselbeck, Wolf.Ritschel}@hella.com, fbeidinger@googlemail.com

[2] Fachgebiet Angewandte Datentechnik, Universität Paderborn,
Warburger Str. 100, 33098 Paderborn
hollmann@adt.upb.de

Abstract: Mechanische Steuerkomponenten, wie die in modernen Automobilen, werden mehr und mehr durch elektronische verdrängt. Diese sind meistens mit einem Steuergerät gekoppelt. Um die Funktionsfähigkeit dieser Steuergeräte zu gewährleisten, ist ein systematisches Testen notwendig. Die Automatisierung der Tests soll nicht nur die Qualität steigern, sondern auch die Effizienz erhöhen, um die Kosten zu reduzieren. Während die eigentliche Durchführung der Tests mittlerweile einen hohen Automatisierungsgrad erreicht hat, werden die Testfälle oft manuell erstellt. In diesem Beitrag wird der Einsatz der Ereignis-Sequenz-Graphen-basierten Methode untersucht, die aus einer formalen Beschreibung eines Steuergerätes die Erzeugung von Testfällen ermöglicht. Diese Methode wird dann mit der Klassifikationsbaummethode anhand einer Fallstudie verglichen.

1 Einführung und Motivation

Die Komplexität der elektronischen Systeme im Automobil wächst bei gleichzeitiger Verringerung der Entwicklungszeit. Die Integration der Steuergeräte zu einem funktionierenden Gesamtsystem wird immer schwieriger, da die Abhängigkeiten der Geräte voneinander immer höher werden. In diesem Beitrag werden die Erfahrungen bei der Anwendung einer graphenbasierten Methode (*Ereignis-Sequenz-Graphen*) zur Ermittlung von Testschritten auf ihre Anwendbarkeit im Testprozess für automotive Anwendungen bei der Firma Hella betrachtet. Zudem erfolgt ein Vergleich der Leistungsfähigkeit mit der gegenwärtigen Methode des manuellen Testens mit der Klassifikationsbaummethode. Dieses wird anhand eines Fallbeispiels eines Fahrerassistenzsystems zur automatischen Abstandsregelung durchgeführt.

In dem Fallbeispiel kommt ein Testsprozess zur Anwendung wie in Tabelle 1 dargestellt. Das betrachtete Steuergerät hängt direkt von der Korrektheit der Signale von vier weiteren Steuergeräten ab.

Etwa 2000 Anforderungen werden hierbei zu 400 Testzielen zusammengefasst, die in 4000 Testschritten umgesetzt werden. Die in der Tabelle angegebenen Aufwände sind empirisch für das Projekt ermittelt worden. Die Vermutung liegt daher nahe, dass eine Optimierung der Anzahl der Testschritte im Testentwurf direkt Zeit bei der Umsetzung der folgenden Prozessschritte einspart.

Prozess	Durchführung	Aufwand
Testplanung	manuell	gering 5%
Testspezifikation	manuell / geringe Toolunterstützung	mittel 20%
Testentwurf	manuell	mittel 20%
Testimplementierung	manuell	hoch 30%
Testdurchführung	automatisch	gering 5%
Testauswertung	manuell / geringe Toolunterstützung	mittel 20%

Tabelle 1: Testprozess und Aufwände

Der Beitrag ist wie folgt strukturiert: Im folgenden Kapitel werden Methoden zum Testentwurf vorgestellt. In Kapitel 3 wird das Testobjekt und seine Umgebung vorgestellt, sowie die Modell- und Testfallerstellung beschrieben. In Kapitel 4 werden die Ergebnisse des ESG-Ansatzes mit manuellem Test verglichen. Ein Ausblick schließt den Beitrag ab.

2 Methoden zum Testentwurf

Weit verbreitet in der Praxis ist immer noch die *intuitive und erfahrungsbasierte Testfallermittlung*. Dabei werden vom Testingenieur die Eingabegrößen so variiert, dass die Ausgangsgrößen die zu erfüllenden Kriterien aufweisen. Der Erfolg dieser Vorgehensweise sowie die Anzahl und Effizienz der aufgestellten Testschritte und -fälle hängen stark von der Erfahrung des Testingenieurs ab. Sie sollte eher dazu benutzt werden, um die nach systematischen Methoden aufgestellten Testfälle sinnvoll zu ergänzen [SL05]. Zudem wird das Aufstellen von Negativ-Testfällen [BL07] meistens vernachlässigt, da die Anforderungen in der Regel nur positiv formuliert sind.

Ebenfalls weit verbreit ist die *Klassifikationsbaummethode* [GG93]. Sie ermöglicht über eine Klassifizierung aller Aspekte des Testobjekts die Komplexität zu reduzieren. Die Grundlage für diese Methode bildet die Systemspezifikation. Bei der Erstellung eines Klassifikationsbaums wird zunächst die Komplexität des in der Spezifikation verankerten Problems zerlegt. Durch die Analyse der Spezifikation werden einzelne für den Test relevante Aspekte (Klassifikationen) der Problemstellung herausgefiltert. Mehrere Klassifikationen können zu einer Komposition zusammengeführt werden. Der Wertebereich einzelner Klassifikationen wird betrachtet und in sinnvolle Klassen zerlegt.

Die Klassen werden so gewählt, dass alle Werte aus diesen als äquivalent betrachtet werden können (Äquivalenzklassen). Diese Äquivalenzklassen können nach der Betrachtung zusätzlicher Bedingungen weiter klassifiziert werden. Durch die wiederholte Bildung von Unterklassifikationen wird der Klassifikationsbaum erstellt.

Beschreibt ein Klassifikationsbaum ein dynamisches Verhalten eines Systems, so können die Blätter als Ereignisse aufgefasst werden. Diese sind gleichzeitig der Kopf einer Kombinationstabelle, wobei jede Zeile einen Testfall spezifiziert.

Ereignis-Sequenz-Graphen (ESG) [Be01] werden benutzt, um das Verhalten eines Systems und die Interaktion mit einem Benutzer bzw. der Umgebung zu modellieren. Ereignisse sind Eingaben des Benutzers und der Umgebung oder stellen Antworten des Systems dar. Ein Ereignis-Sequenz-Graph $ESG=(V,E)$ besteht aus einer endlichen Menge von Ereignissen V und einer endlichen Menge von Kanten $E \subseteq V \times V$. Eine Kante $(v,v') \in E$ gibt an, dass das Ereignis v' dann ausführbar ist, wenn das Ereignis v ausgeführt wurde. Die Pseudoknoten $[$ und $]$ markieren zulässige Start- bzw. Endereignisse. Der Graph beinhaltet somit alle gültigen Ereignissequenzen. Der inverse Graph $\overline{ESG} = (V, \overline{E})$ ist gegeben durch die Menge der Ereignisse V und die Kantenmenge $\overline{E} = (V \times V) \setminus E$. Er repräsentiert das ungültige Verhalten des Systems und wird zur systematischen Generierung von Negativtests benutzt. In Abbildung 1 ist beispielhaft der Graph $ESG=(V,E)$ mit $V=\{a,b,c\}$ und $E=\{(a,b),(a,c),(b,c),(c,b)\}$, der inverse Graph $\overline{ESG}$ mit $V=\{a,b,c\}$ und $\overline{E} = \{(a,a),(b,a),(b,b),(c,a),(c,c)\}$ und der komplettierte Graph gegeben.

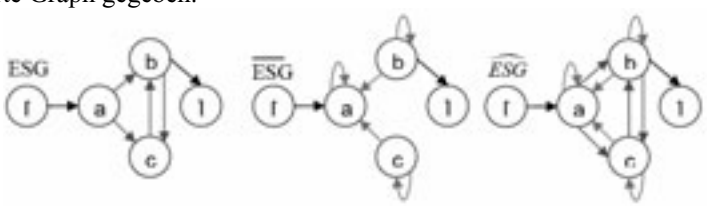


Abbildung 1: Ereignis-Sequenz-Graph, inverser und komplettierter Graph

3 Bewertungskriterien und Fallbeispiel

3.1 Anforderungen an die Testmethode

Die Anwendbarkeit der ESGs wurde bereits für das Testen von graphischen Oberflächen erfolgreich nachgewiesen [BHN07]. Das Testen von Steuergeräten besitzt andere Rahmenbedingungen. Die Anforderungen sind nur in textueller Form festgeschrieben. Die Steuergeräte sind physikalisch über CAN-Bus, Spannungsversorgung und mechanischer Halterung mit dem Fahrzeug verbunden.

Die Sensorik und die Einbindung des Steuergerätes in die Umwelt werden in dem Fallbeispiel nicht betrachtet, sondern nur der Teil, der für die Fahrzeugumgebung relevant ist. Da der Test der Fahrzeugintegration für fast alle Steuergeräte ähnlich ist, kann hier sehr gut die gegenwärtige Testmethode für den Vergleich mit der Anwendung von ESGs herangezogen werden.

3.2 Auswahl des Testobjekts

Das Steuergerät (ein Radarsensor auf 24GHz Technik) ist über den CAN-Bus mit fünf weiteren Steuergeräten und so indirekt mit mehr als 7 Sensoren gekoppelt und wird über die Standardspannungsversorgung mitversorgt. Von dem Gesamttest wurde ein Minimalfall (8 Zustände, abhängig von ca. 4 Eingangsgrößen) und ein Maximalfall (ca. 20 sog. Test-Targets, abhängig von über 20 Eingangsgrößen, zentraler Zustandsautomat) ausgewählt.

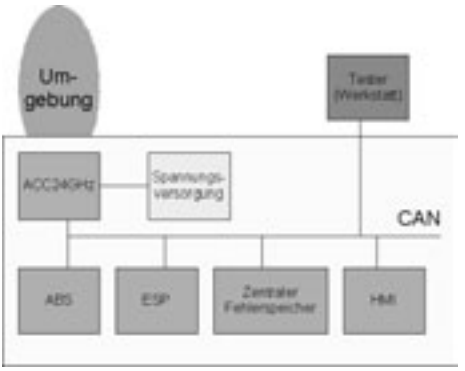


Abbildung 2: Abhängigkeit der Steuergeräte über das CAN-Netzwerk

Abbildung 2 zeigt die physikalischen Abhängigkeiten des Testobjekts (ACC24GHz:Adaptive Cruise Control, 24GHz Radar) zu den übrigen Steuergeräten. Der zentrale Zustandsautomat ist ein besonders komplexes und umfangreiches Testobjekt, da er in direkter Beziehung zu den Signalen aller aufgezeigten Steuergeräte steht. Aufgrund der sich kontinuierlich ändernden Anforderungen während der Entwicklung wurden zahlreiche Erweiterungen vorgenommen, die sich mit einem Zustandsautomaten nicht konsequent modellieren lassen („regelbasiertes Verhalten“). Abbildung 3 zeigt eine Zusammenfassung der für den Zustandsautomaten relevanten Signale. Grundsätzliche Randbedingung für die Durchführung der Tests war, dass nicht auf interne Signale oder Daten zugegriffen werden soll.

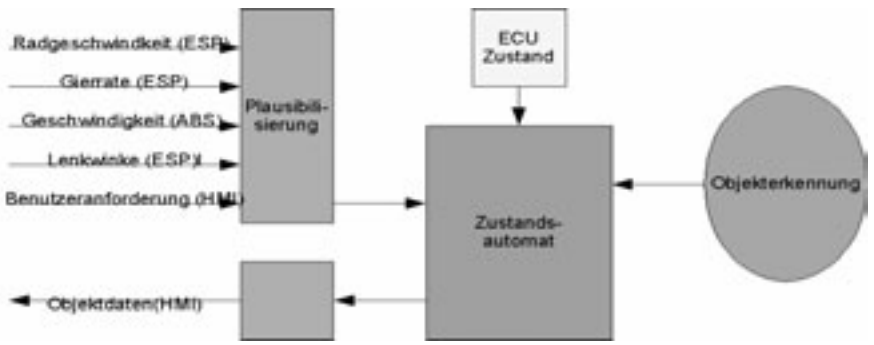


Abbildung 3: Ein- und Ausgangssignale

3.3 Testumgebung

Um die Durchführung der Integrationstests automatisieren zu können, wurde ein Integrationstestplatz eingerichtet. Mit Hilfe des Integrationstestplatzes werden unterschiedliche Situationen, die auf der Strasse auftreten können, simuliert und in den Test eingebunden. Mit Hilfe der SPS und programmierbaren Netzgeräte werden Kurzschlüsse, Fehlpotentiale sowie unterschiedliche Versorgungsspannungskurven am Bus sowie am zu testenden Steuergerät realisiert. Der gesamte Testablauf wird von einem Hostrechner mit der Softwarekomponente CANoe überwacht und durchgeführt. Außer der Überwachung und Steuerung des Testprozesses wird mit Hilfe von CANoe die imaginäre Vernetzung des Steuergerätes im Auto (Restbussimulation) ohne Vorhandensein von reellen Steuergeräten, die unmittelbar mit dem Prüfling über CAN-Bus gekoppelt sein müssen, realisiert. Am Ende eines Tests erstellt CANoe einen Testreport im *.xml oder *.html Format, in dem alle durchgeführten Testschritte von einzelnen Testfällen mit kurzen Beschreibungen aufgelistet sind. Der Testreport dient als Grundlage für die spätere Testauswertung.

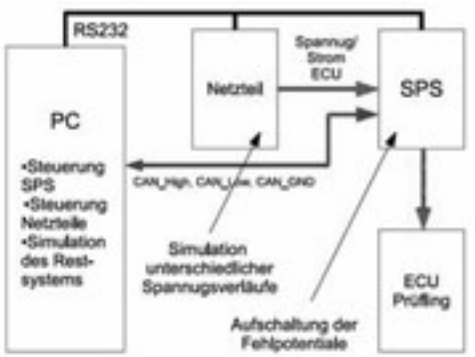


Abbildung 4: Schematischer Aufbau des Integrationstestplatzes

3.4 Anwendung der ESG-Methode und Testfallgenerierung

Die ESG-Methode wurde im Rahmen einer Diplomarbeit auf ihre Anwendbarkeit untersucht [Bei07]. Dazu wurden die Testfälle mit Hilfe der ESG-Methode aufgestellt und auf der beschriebenen Testumgebung durchgeführt. Dabei wurden die Original-Lastenhefte für die Erstellung der Testspezifikation mit herangezogen.

Als Grundlage für die Knoten des ESGs dienen die system- und testumgebungsspezifischen Ereignisse. Die systemspezifischen Ereignisse sind die Ereignisse, die auf der Kommunikationsebene (Netzwerkebene) geschehen. Damit sind die Signale gemeint, dessen Eingang bzw. Ausgang als Ereignis interpretiert wird. Die testumgebungsspezifischen Ereignisse beschreiben dagegen die äußeren Einflüsse auf das System. Zu solchen Ereignissen gehören die Aufschaltungen von Fehlpotentialen sowie Kurzschlüsse im Gesamtsystem. Jedem Ereignis wird ein Programmcode zugeordnet, der beim Auftreten des Ereignisses ausgeführt wird.

Um die Zusammenhänge zwischen einzelnen Ereignissen besser verstehen zu können, müssen einzelne Punkte der Systemspezifikation mit Hilfe von Kombinations- oder Zustandsübergangstabellen zusammengeführt und damit die Aufgabenstellung präzisiert werden. Anhand dieser Tabellen wird dann entschieden, ob einem Ereignis ein oder mehrere Knoten im ESG zugeordnet werden müssen. Die Kantenbildung erfolgt anhand der Systemspezifikation und/oder durch logische Zusammenhänge. Die Erkenntnisse, die bei der Präzisierung der Aufgabenstellung (Aufstellung der Kombinations-, Zustandsübergangstabellen usw.) gewonnen wurden, müssen bei der Kantenbildung berücksichtigt werden. Ausgehend von einem Knoten (Startknoten bevorzugt) im ESG werden zwischen ihm und allen seinen direkten Nachfolgern Kanten angelegt. Diese Prozedur wird solange wiederholt, bis alle Knoten mit ihren direkten Nachfolgern verbunden sind. Beim Anlegen der Kanten dürfen einzelne Knoten nicht isoliert, sondern müssen im Zusammenhang mit allen ihren Vorgängern betrachtet werden. Abbildung 5 zeigt ein Beispiel. Aus den Modellen wurden automatisch Testfälle generiert, so dass sämtliche Ereignispaare (Kanten) überdeckt werden. Diese Testsequenzen wurden mittels des *Chinese Postman Problems* [Ah91] minimiert. Zudem wurden Testfälle aufgestellt, um die ungültigen Ereignispaare des inversen Graphen abzudecken.

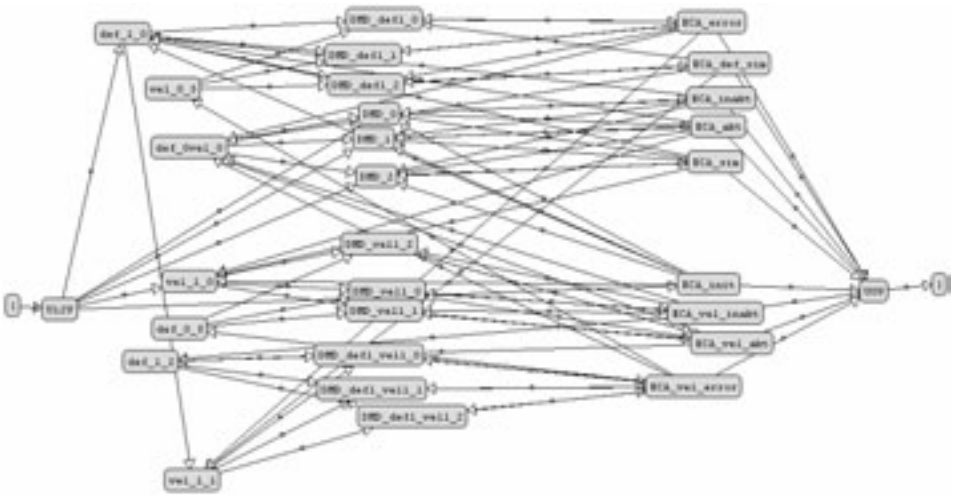


Abbildung 5: Beispiel eines Ereignis-Sequenz-Graphen zur Überprüfung der Zustandsübergänge

3.5 Manueller Testprozess mit Hilfe der Klassifikationsbaummethode

Zur selben Zeit hat ein Team von zwei Personen einen vollständigen Test nach der manuellen Methode, basierend auf Klassifikationsbäumen (aktionsorientierte Variante [HL02]), spezifiziert und implementiert. Ein Beispiel ist in Abbildung 6 gegeben. Bei der Erstellung der Testspezifikation zeigte sich, dass die Findung der Testziele relativ schnell und zügig möglich war. Aufwändig war dagegen die Verfeinerung der Testziele in einzelne Testschritte und die Pflege bei Änderungen. Die Zusammenhänge waren nicht mehr intuitiv verfolgbar.

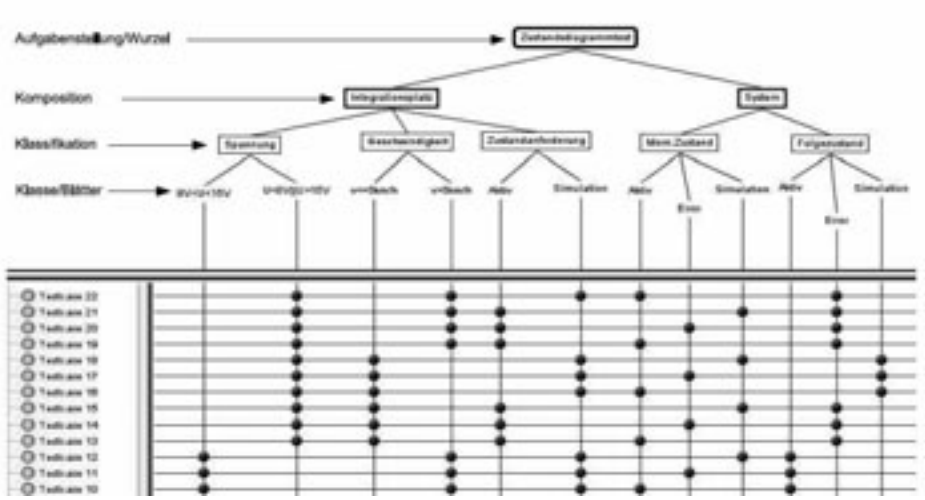


Abbildung 6: Beispiel Klassifikationsbaum

4 Bewertung der Anwendbarkeit der ESG-Methode und Ausblick

Die Anzahl der benötigten Stunden, die für die Erstellung der Testspezifikation gebraucht wurden, nahm mit der ESG-Methode ca. 26 % mehr Zeit in Anspruch (s. Tabelle 3). Die Anzahl der Testfälle konnte bei der ESG-Methode jedoch mehr als halbiert werden – die Anzahl der Testschritte sogar fast um den Faktor vier.

	Manuelle Methode	ESG
Stunden zur Aufstellung der Testspezifikation	120	152
Anzahl Testfälle	26	11
Anzahl Testschritte	240	62

Tabelle 3: Stundenvergleich

Mit Hilfe der Ereignissequenzgraphen wurden zudem alle Fehler gefunden, die mit dem manuellen Testprozess gefunden wurden. Zusätzlich zu diesen Fehlern wurde ein weiterer, erheblicher (da sicherheitskritisch) Fehler in der Implementierung des zentralen Zustandsautomaten gefunden. (Fehler Nr. 3). Tabelle 4 gibt einen Überblick über die mit der ESG-Methode gefundenen Fehler.

1	Befindet sich das System im Fehlerzustand, die Geschwindigkeit sinkt unter 5km/h und es erfolgt eine Simulationsaufforderung, bleibt das System im Fehlerzustand anstatt in den Simulationszustand zu wechseln.
2	Beim Aufschalten des Fehlers wechselt das System nicht rechtzeitig in den Fehlerzustand.
3	Das System befindet sich im Fehlerzustand und die Fehlerbedingung verschwindet. Dann wechselt das Gerät nicht rechtzeitig in den angeforderten Zustand Aktiv (Fehler gefunden mit Hilfe von negativen Testfällen).
4	Wenn sich das System im Fehlerzustand befindet und die Fehlerbedingung verschwindet, wechselt das Gerät nicht rechtzeitig in den angeforderten Zustand Inaktiv (Fehler gefunden mit Hilfe von negativen Testfällen).
5	Das System wechselt nicht rechtzeitig aus dem Zustand Init in Aktiv oder Inaktiv.

Tabelle 4: Fehlerliste

Abschließend lässt sich sagen, dass die Zeit für die Implementierung der Testfälle und die Kosten, die durch die Testdurchführung entstehen mit Hilfe von der ESG-Methode erheblich reduziert wurden. Mit dem Ereignis-Sequenz-Graphen ist eine Methode gegeben, die die Automatisierung der Testvorbereitung ermöglicht und zudem das Regressionstesten vereinfacht. Weitere Schritte zur Effizienzsteigerung sind zudem möglich, und zwar durch die Entwicklung eines Rahmenprogramms, das die generierten Testfälle aus den ESGs in ein von CANoe lesbares Format wie z.B. XML umsetzt.

Literaturverzeichnis

[Ah91] Aho, A. V. et.al.: An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours. IEEE Transactions on Communications, Vol. 39(11), S. 1604-1615, 1991.

[Be01] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE 01), S. 34-43, IEEE CS, 2001.

[BHN07] Belli, F.; Hollmann, A.; Nissanke, N.: Modeling, Analysis and Testing of Safety Issues - An Event-Based Approach and Case Study. Computer Safety, Reliability, and Security, LNCS 4680, Springer Berlin, 2007.

[BL07] Belli, F.; Linschulte, M.: 'Negativ'-Tests interaktiver Systeme und ihre Automatisierung. Lecture Notes in Informatics, Vol. 106, S. 35-44, Software Engineering 2007 - Beiträge zu den Workshops, Gesellschaft für Informatik, Bonn, 2007.

[Bei07] Beidinger, F.: Bewertung einer graphenbasierten Methode für den Testentwurf. Diplomarbeit Universität Paderborn, Fachgebiet Angewandte Datentechnik, 2007.

[BN02] Broekman, B.; Notenboom, E.: Testing Embedded Software. Addison-Wesley-Verlag, 2002.

[GG93] Grochtmann, M.; Grimm, K.: Classification Trees for Partition Testing. Software Testing, Verification and Reliability, 3(2), S. 63-82, John Wiley Verlag, 1993.

[HL02] Hanisch, J.; Letzel, J.: Automatisierung von Regressionstests eines Programms zur Halbleiter-Strukturanalyse. Diplomarbeit HU Berlin, 2002.

[SL05] Spillner, A.; Linz, T.: Basiswissen Softwaretest. 3. Auflage, Dpunkt Verlag, 2005.

A TTCN-3-based Web Service Test Framework

Edith Werner, Jens Grabowski, Stefan Troschütz, Benjamin Zeiss
Software Engineering for Distributed Systems Group,
Institute for Computer Science, University of Göttingen,
Lotzestr. 16–18, 37083 Göttingen, Germany.
{ewerner,grabowski,zeiss}@cs.uni-goettingen.de,
stefan.troschuetz@stud.informatik.uni-goettingen.de

Abstract: The increased usage of Web services for critical applications introduces a growing need for efficient testing approaches to assure their quality. The *Testing and Test Control Notation* (TTCN-3) is a standardised testing language that is well suited for black-box testing of distributed systems such as Web services. Also due to its abstract test specification methodology, it allows easy adaptation to different Web service frameworks or platforms. This paper presents a mapping from the *Web Service Description Language* (WSDL) to TTCN-3 and a corresponding automated translator.

1 Introduction

Web services are a standards-based approach to distributed communication that utilises the *Hypertext Transfer Protocol* (HTTP) and the *Extensible Markup Language* (XML) as technological foundation. Usually, a Web service is expected to have an interface described in a machine-processable format, e.g. the *Web Service Description Language* (WSDL), and to communicate to other systems using SOAP messages via HTTP. Due to the progressing adoption of Web services in business-critical applications, the demand for efficient testing approaches increases.

The *Testing and Test Control Notation* (TTCN-3) is a test specification and test implementation language standardised by the *European Telecommunications Standards Institute* (ETSI) and the *International Telecommunication Union* (ITU). Several authors already recognised the suitability of TTCN-3 for Web service testing and also introduced the idea of deriving abstract test interfaces from a Web services' WSDL description.

In this paper, we present a specification of a mapping between WSDL and TTCN-3 *Abstract Test Suite*s (ATSs) that is both unambiguous and modularly structured. The mapping is implemented and has also been integrated into the commercial TTCN-3 *Integrated Development Environment* (IDE) TTworkbench.

This paper is structured as follows: We start by giving an overview on related research in Section 2. In Section 3, we describe the foundations of SOAP, WSDL, and TTCN-3. Our mapping of WSDL to TTCN-3 is presented in Section 4. Afterwards, in Section 5, the implementation of our mapping and its integration into the development environment TTworkbench are described. In Section 6, we conclude with a summary and an outlook.

2 Related work

There are various tools from industry dealing with Web service testing, many of which are language-specific and sometimes even bound to a specific test platform. For example, in [JA] the authors present how to automate Web service testing using technologies such as JUnit, Apache Commons HttpClient, and XMLUnit, whereas in [McC], the authors automate ASP.NET Web service testing using means provided by the .NET Framework. However, these tools and approaches mostly do not sufficiently cope with the language and platform-independency of Web services.

In [XPS05], the advantages of Web service testing based on TTCN-3 are discussed. The ATS is developed using the knowledge of the WSDL descriptions, models, and the Web service source code while the adapter concept of TTCN-3 enables testing using different client-side implementation languages and different Web service client toolkits by simply switching the TTCN-3 execution platforms and adapters. Anyhow, there is no automation of the mapping involved.

In [SS03], the authors discuss the automated testing of SOAP based Web services by use of TTCN-3 and present a mapping between XML data descriptions and TTCN-3 data to enable the automated derivation of test data. The paper is the first proposal of an automated mapping between *Document Type Definition* (DTD) [W3C06] or *XML Schema Definition* (XSD) types and TTCN-3 type definitions. Nonetheless, the automated generation of TTCN-3 types associated with test components as well as generic test implementation and execution are not considered properly. Additional work on importing XSD types into TTCN-3 [Jea04, JDR04] became the foundation of the forthcoming TTCN-3 Standard Part 9 [ETS07], which was finalised as draft in October 2006 and is going to standardise the use of XSD with TTCN-3.

In [SVR06], the import of WSDL descriptions into TTCN-3 as well as mapping rules between WSDL and TTCN-3 definitions are discussed. It enables the automated derivation of basic TTCN-3 test suites from WSDL descriptions where only specific test data must still be entered manually. Although the approach is evidenced by means of a simple case study, the presented mapping between WSDL and TTCN-3 is incomplete, e.g. it is not considered how TTCN-3 identifiers derived from WSDL can be ensured to be unique.

3 Foundations

In the following, we provide an overview on the Web service technologies SOAP and WSDL as well as key concepts of the test specification language TTCN-3. Language concepts from WSDL, SOAP and TTCN-3 will be highlighted using *slanted type*.

SOAP [W3C03] is a protocol for exchanging XML-based messages in a distributed environment and constitutes the leading standard for the interaction with Web services. The SOAP protocol assumes that every message has a sender, a receiver, and an arbitrary number of so-called intermediaries that process the message and route it to the receiver. The *Body* element carries the core information of a SOAP message and its content must be

processed by the receiver. The *Header* element is used to add functionality or additional information to a SOAP message without modifying the core of the message. For example, the header could contain a session identifier or security credentials.

WSDL [W3C01] is an XML-based format for the description of Web services that specifies the exposed interface and location of a Web service as well as how to access it. The overall structure of WSDL documents can be divided into two parts. The abstract part consisting of the *types*, *message*, and *portType* elements describes the interface of the Web service, i.e. the exposed operations and data types. The instantiation part comprising the *service* and *binding* elements describes a concrete implementation of the service interface at a network endpoint, i.e. where the Web service is located and how its operations can be accessed.

TTCN-3 is a multipart standard [ETS07] which consists of a core language (part 1 of [ETS07]), a number of representation formats (parts 2 and 3), the operational semantics (part 4), the *TTCN-3 Runtime Interface* (TRI) for the connection to the *System Under Test* (SUT) (part 5), the *TTCN-3 Control Interface* (TCI) for the connection to the test environment (part 6) as well as several language bindings (parts 7–9). The TTCN-3 core language is similar to typical programming languages, but provides additional features dedicated to testing such as built-in data matching, test verdicts, timer handling, or the concurrent execution of test components. The test behaviour is specified by *test cases*, *functions*, and *altsteps*. Functions and altsteps are used to specify and structure test behaviour. A *default* is an altstep that when activated is implicitly included at the end of alternative behaviour and is typically used for handling errors and timeouts. A *test case* is a special kind of function that always returns a test verdict. The different TTCN-3 elements are structured into *modules*. Besides the test definitions, a module can contain a control part that is the behavioural entry point when a TTCN-3 module is executed. To increase the flexibility of the modularisation, a module can also import definitions from other modules.

4 A mapping of WSDL to TTCN-3

To test a software system using a test specification language like TTCN-3, an ATS has to be created which is then compiled and finally executed on the test system. In the case of a Web service, an ATS skeleton can be derived from the interface definition that is given in WSDL as depicted in Figure 1. For the test execution, a SUT adapter and codec have to be provided to instantiate the ATS for the target system. The specification of abstract test specifications simplifies reusability and portability, but also implies that the mapping is involved at two different levels in a TTCN-3 test system: First, the information by the provided WSDL documents must be translated to TTCN-3 and the TTCN-3 type system to enable the development of Web service test suites in the TTCN-3 core language. Second, the TTCN-3 adaptation layer must implement the reversed mapping to realise compatibility with the original interface definition. Thus, our mapping is a detailed rule set that fulfils the requirement to be unambiguous in both directions. In this paper, we only highlight the interesting aspects of the mapping.

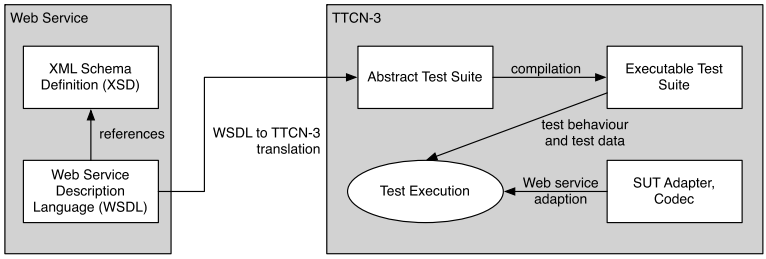


Figure 1: The Web service test framework

4.1 Communication paradigm

An essential choice that influences the mapping rules is whether the TTCN-3 ATS should use message-based or procedure-based communication for the interaction with Web services. In the related research work, both communication ways are used almost equally often. The mapping presented in this paper generates a TTCN-3 ATS that uses message-based communication. This decision is based on the practical observation that Web services typically use message-based communication themselves. Also, the input, output, and fault messages described by the WSDL document can be mapped directly to a message type definition in TTCN-3. Lastly, the *call* statement, a procedure-based communication operation, is not allowed to reference *altsteps* and during its evaluation, all active *defaults* are ignored. Therefore, common behaviour such as the reaction on inactivity of the system under test must be specified repeatedly.

4.2 Identifier conversion

Each of the two languages, WSDL and TTCN-3, define a set of rules and naming conventions for identifiers to assure their uniqueness and readability. In our setting, the WSDL identifiers are translated into TTCN-3 identifiers for the generation of the ATS and have to be reverted when the test suite is instantiated and executed. To allow this, TTCN-3 identifiers are derived from the name attribute of WSDL elements. However, as WSDL allows a wider range of characters in identifiers than TTCN-3, these additional characters are replaced by reversible escape sequences to obtain a TTCN-3 compatible name.

In WSDL, scoping and overloading of identifiers is widely used, e.g. it is possible to import elements with equivalent names from another document or from another XSD. To generate unique TTCN-3 identifiers, the WSDL identifiers are prefixed with their type and scope. The prefixes employed are partly suggestions made by [WDT+05] which are supplemented with a set of prefixes closely bound to the domain of Web service testing.

4.3 Mapping of WSDL language constructs

Our mapping rules make wide use of modularisation, which enhances the readability, reusability and maintainability of the resulting test suite. In addition, the modularisation reduces the number of rules that target the elimination of name ambiguities.

The WSDL *types* element encloses data definitions, usually given using the XSD, that are relevant for the exchanged messages. The mapping of XSD data type definitions to TTCN-3 is defined in the forthcoming TTCN-3 standard part 9 [ETS07], which is adopted with some minor changes and additions. For example, rather than coding a unique qualified form for each identifier, a module is used to represent a target namespace of any enclosed XSD instances. This as well improves the overall test suite structure. The components defined by each XSD are mapped to TTCN-3 type definitions, which are located in the module that has been created for each corresponding target namespace of the schema.

The WSDL *portType* element is also mapped to a module. Within the module, a TTCN-3 *group* is defined for each operation element of the respective WSDL port type. Inside of each group, a TTCN-3 *record* type is defined for every input, output, and fault element of the respective WSDL operation. At each representation of an input element, the first field of the according TTCN-3 *record* type is named soapbinding and is of the TTCN-3 SoapBinding type, so that each input message can be accompanied with the proper SOAP binding information upon test execution. The SoapBinding type is predefined inside a separate module containing base definitions. The other fields of the TTCN-3 *record* types are derived from the WSDL message that is referenced by the input, output, or fault element to which they correspond. Finally, each TTCN-3 module representing a WSDL port type contains a TTCN-3 *port* type which uses message-based communication and allows sending and receiving of the defined *record* types.

For each WSDL *port* and its associated *binding* element, a TTCN-3 module is created which contains TTCN-3 constants with SOAP binding information for every described operation. Furthermore, additional local constants are created inside this module that hold information common to all operations defined by the respective WSDL port, e.g. SOAP version and *Uniform Resource Identifier* (URI). For each operation, a record of type SoapBinding is declared which is composed of references to the constants holding common SOAP binding information and the invocation details specified by the attributes of the SOAP binding elements.

The WSDL *service* element is mapped to a collection of modules. Each module contains basic test behaviour and execution control for testing a WSDL port type referenced by the service via port and binding and a component type definition that specifies the associated WSDL port type. In addition, test cases and necessary templates for testing the operations defined by the WSDL port type are created. The module control part executes the defined test cases for each WSDL port of the service which references the WSDL port type.

The complete mapping including all technical details and an extensive example can be found in the master's thesis of Stefan Troschütz [Tro07].

5 Implementation

In order to execute our generated test suites, we used the commercial tool TTworkbench [TT] as development environment and test system. In addition, we implemented an integration into the Eclipse-based TTworkbench making our translation tool easy to use. As of October 2007, the TTworkbench tool provides WSDL support via an own plug-in, which was developed concurrently to the one presented here. As far as we know, the main difference in the mapping is the used communication paradigm as well as the modular organisation of our mapping.

The automated mapping starts with the mapping of the WSDL *types* elements. The XSDs contained in those elements are retrieved and mapped to TTCN-3 modules. To ease dealing with the possible interdependencies of the XSD components, all schemas are mapped at once. Next, the WSDL *portType* elements are processed. For every WSDL port type defined in the currently mapped WSDL document, a corresponding TTCN-3 module is generated. Finally, the WSDL *service* and *port* elements are mapped. For each *service* element, the enclosed port elements are associated with the corresponding port type definitions. During the mapping procedure, TTCN-3 identifiers are generated mainly by applying rewriting rules on patterns matched by regular expressions.

5.1 Test adapter and codec

In order to run the *Executable Test Suite* (ETS) generated by TTworkbench's TTthree compiler against an actual SUT, a specific SUT adapter and codec have to be implemented. Both SUT adapter and codec inherit from the abstract implementations provided by TTworkbench and override the methods of the TRI and the TCI that are invoked by the test system when executing a test suite. In order to deal with SOAP messages and communicate with a Web service under test, the SUT adapter and codec make use of the *Application Programming Interface*s (APIs) provided by the Apache Axis distribution [Axi]. The codec uses the implementation of the identifier mappings for proper encoding and decoding of SOAP messages and for the recovery of element and attribute names from TTCN-3 identifiers.

5.2 Test composer

Since a WSDL document specifies neither semantics nor behavioural context for the described operations, complex test scenarios cannot be derived from it. Therefore, the automatically generated ATS contains only simple test cases for testing the Web service operations individually. To ease the specification of new and more complex test cases, we provide a dialogue-based means to freely combine the generated operation calls with user-defined data templates and adequate error- and verdict-handling.

First, the user has to select a TTCN-3 module representing a WSDL service where the

new test scenario should be inserted. The test component on which the behaviour for testing the corresponding WSDL port type is executed is determined and analysed and the associated operations are presented grouped per port. From this, the user can specify the sequence of Web service operations that should be invoked in the course of the test scenario. In the second step, the user has to specify identifiers for the TTCN-3 definitions used by the new test scenario, including a test case, templates for all input, output, and fault messages of the selected Web service operations, and a group that encloses the former test scenario definitions. All specified identifiers have to be unique within the context of the target module and are required to adhere to the naming conventions to enable their translation into WSDL. Finally, the new test scenario is created and inserted into the target module. For that purpose, a syntax tree representing the test scenario group and all enclosed TTCN-3 definitions is constructed, transformed into TTCN-3 core notation, and inserted into the target module at the place initially specified by the user.

The behaviour of composed test cases is determined as follows. For each operation, first the template for the input message is transmitted via the corresponding port. If the operation is a one-way operation, i.e. the Web service under test will not respond, the test verdict is set to *pass*. Otherwise, the timer owned by the test component is started to safeguard the test behaviour against inactivity of the Web service under test. After starting the timer, an *alt* statement specifies several alternatives. The first alternative is executed if the template defined for the output message of the Web service operation is duly received. It will stop the timer and set the test verdict to *pass*. If the operation specifies fault messages, further alternatives are defined for the receipt of their corresponding templates. Each of those alternatives will also stop the running timer, but set the test verdict to *fail*. At last, a *default* is activated in order to catch the receipt of unexpected messages or a timeout.

6 Summary and outlook

By utilising the WSDL description of a Web service, it is possible to derive an ATS in TTCN-3 that provides the necessary kind of abstraction from the actual implementation platform and the communication details. With the comprehensive mapping specification that we have created, Web service testing in TTCN-3 becomes more reliable, comprehensible, consistent and efficient. The automated translation enables the test developer to concentrate on actual Web service test development from the start and avoids typical humanly errors. Our automated WSDL to TTCN-3 translator is integrated into the TTCN-3 development environment TTworkbench as an Eclipse plug-in.

The presented mapping for Web service testing is one ingredient of a fully-fledged Web service development cycle that we are currently developing. It will support Web service development from design to deployment. We also plan the attempt to couple the existing test case composer with a process description like the *Business Process Execution Language* (BPEL) to automatically generate more advanced test cases.

References

[Axi] Web Services - Axis. Apache <Web Services/> Project, http://ws.apache.org/axis/. [January 11th 2008].

[ETS07] ETSI Standard (ES) 201 873 V3.2.1: The Testing and Test Control Notation version 3; Parts 1-9. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation series Z.140, 2007.

[JA] M. Jadhav and M. Ahmed. Automate Web service testing, Part 2: Test a Web service with XMLUnit. IBM developerWorks, http://www-128.ibm.com/developerworks/edu/ws-dw-ws-soa-autotest2.html [January 11th 2008].

[JDR04] D.-M. Jeaca, G. Din, and A. Rennoch. Importing XMLSchema datatypes into TTCN-3. In *Proceedings of the 3rd Workshop on System Testing and Validation (SV04)*, 2004.

[Jea04] D.-M. Jeaca. XML Schema to TTCN-3 Mapping: Importing XMLSchema datatypes into TTCN-3. Master's thesis, University Politechnica Bucharest, Department of Computer Science, Bucharest, Romania, 2004.

[McC] J. McCaffrey. Test Run: Automate Your ASP.NET Web Services Testing. MSDN Magazine, http://msdn.microsoft.com/msdnmag/issues/05/03/TestRun/ [January 11th 2008].

[SS03] I. Schieferdecker and B. Stepien. Automated Testing of XML/SOAP based Web Services. In *Proceedings of the 13th Fachkonferenz der Gesellschaft für Informatik (GI) Fachgruppe "Kommunikation in Verteilten Systemen" (KiVS)*, 2003.

[SVR06] Ina Schieferdecker, Diana Vega, and Cosmin Rentea. Import of WSDL Definitions in TTCN-3 Targeting Testing of Web Services. In *Proceedings of the 9th International Conference on Integrated Design and Process Technology (IDPT)*, 2006.

[Tro07] Stefan Troschütz. Web Service Test Framework with TTCN-3. Master's thesis, ZFI-BM-2007-14, University of Göttingen, Institute for Computer Science, Göttingen, Germany, 2007.

[TT] Testing Technologies TTworkbench. http://www.testingtech.de/products/ttwb_intro.php. [January 11th 2008].

[W3C01] Web Services Description Language (WSDL) 1.1. World Wide Web Consortium (W3C) Note, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[W3C03] SOAP Version 1.2 Part 0: Primer. World Wide Web Consortium (W3C) Recommendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 2003.

[W3C06] Extensible Markup Language (XML) 1.0 (Fourth Edition). World Wide Web Consortium (W3C) Recommendation, http://www.w3.org/TR/2006/REC-xml-20060816/, 2006.

[WDT+05] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz. *An Introduction to TTCN-3*. Wiley, New York, 2005.

[XPS05] P. Xiong, R. L. Probert, and B. Stepien. An Efficient Formal Testing Approach for Web Service with TTCN-3. In *Proceedings of the 13th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2005.

Die Einführung eines wieder verwendbaren Testframeworks in der Sozialversicherung

Armin Beer

Siemens IT Solution and Services PSE
Gudrunstrasse 11
A-1100 Wien
armin.beer@siemens.com

Abstract: Einen gut durchdachten und wirksamen Test für ein Softwareprojekt aufzusetzen ist anspruchsvoll und aufwendig. Daher wurde im Rahmen einer Beratung für einen Sozialversicherungsträger ein wieder verwendbares Testframework für einen iterativen Entwicklungsprozess eingeführt. Dessen wichtigste Merkmale sind eine enge Kooperation zwischen Analytikern, Entwicklern, Anwendern und Testern, der Aufbau von Test-Know-How im ganzen Team und Maßnahmen zur frühen Fehlerfindung durch automatisierte und systematische Tests auf Basis des V-Modells. Die Anwendung des Frameworks in einem Pilotprojekt ermöglichte die planmäßige Lieferung einer Applikation mit zufriedenstellender Qualität und war die Voraussetzung für dessen generelle Verwendung.

1 Einleitung

Bei einem Sozialversicherungsträger, der die gesetzliche Kranken- und Unfallversicherung für die österreichischen Beamten und Vertragsbediensteten durchführt werden sukzessive bestehenden Host-Applikationen durch moderne J2EE Applikationen abgelöst. Im Zuge dessen sollte ein für alle Software-Entwicklungsprojekte verbindliches Testframework, mit dem Ziel die Qualität der dort realisierten Produkte zu sichern, eingeführt werden. Dazu wurden Test-Fachleute von SIS PSE als Berater angefordert.

Folgende Anforderungen waren zu erfüllen:

- Eignung des Testframeworks für eine iterative Entwicklung
- Aufbau einer Testmanagementgruppe, die unabhängig von der Entwicklung Tests durchführt oder einzelne Projekte unterstützt
- Aufbau von Test-Know-how und *Skills* bei Entwicklern und Testern
- Anwendung von Tools – vorzugsweise *Open Source Software*

2 Realisierung des Testframeworks

Es wurde ein Framework, dessen Betonung auf einem klaren methodischen Vorgehen nach ISTQB [SL05] beruht, realisiert. Insbesondere wurden systematische und explorative Testfall-Entwurfsverfahren zur Reduktion des zu testenden Umfangs unter Beibehaltung des Anspruchs auf hohe Fehler-Aufdeckungs-Kapazität angewendet.

Das Testframework umfasst:

- Tool-gestütztes Testmanagement u. a. mit definierten Rollen, Abläufen und Metriken
- *Workflow* mit durchgängiger Verfolgbarkeit (Traceability) von den Anforderungen zu Use-Case-Beschreibungen und weiter zu Testfällen
- Richtlinien für methodisches Testfall-Design (zustandsbasiert, Äquivalenzklassen, exploratives Testen etc.)
- Curriculum für Fachtester, Testleiter und Entwickler (ISTQB-Certified Tester-Prüfung, Schulungen in Methoden und Tools, *Coaching* durch Berater etc.).
- Support verschiedener Projekte durch eine Testmanagement-Gruppe

Das dem Testvorgehen zugrundeliegende V-Modell wurde für die iterative Entwicklung adaptiert. Alle Rollen im Projekt-Team sind genau definiert. Zum Beispiel unterstützt der Testdaten-Administrator auch die Entwickler bei der Erstellung ihrer Testdaten. Zur besseren Team-Bildung und Förderung der Motivation für den Test werden Analytiker, Fachtester, Entwickler etc. in Workshops und durch begleitendes *Coaching* in der Anwendung der Methoden und Tools geschult. Der Know-How-Transfer erfolgte anhand von Test-Richtlinien, die das Rollen-spezifische Wissen berücksichtigen.

Automatisierte Regressions-Tests auf allen Testebenen und Last- und Performance-Tests werden regelmäßig durchgeführt. Um die Transparenz über den Status der einzelnen Arbeitspakete zu erzielen, können z.B. die Test-Ergebnisse jedes *Nightly Builds* von jedem Entwickler oder Fachtester über einen *Intranet-Link* abgefragt werden. Die Tätigkeiten des Testprozesses nach ISTQB (Planung und Steuerung, Analyse und Design, Realisierung und Durchführung, Auswertung und Bericht, Abschluss) werden durch Tools unterstützt. Für das Testmanagement werden das Tool *SiTEMPPO*[1], für das Anforderungs- und Fehlermanagement das Tool *Bugzilla*[2], für die Testautomatisierung die Tools HP WinRunner und *QuickTestPro*[3], für Lasttests die OpenSource Software *OpenSTA*[4] und für die Komponenten-Tests die OpenSource Tools[5] *JUnit, HttpUnit, CruiseControl* und *DbUnit* verwendet. Das Framework und die Dokumente sind so gestaltet, dass sie für andere Entwicklungsprojekte wieder verwendbar sind.

---

[1] Tool von SIS PSE (SiTEMPPO: Test Execution, Managing, Planning and rePorting Organizer).
[2] bugzilla.mozilla.org
[3] HP QuickTest Professional Software
[4] http://www.opensta.org/ .
[5] http://www.junit.org/; http://www.torsten-horn.de/techdocs/java-httpunit.htm; http://cruisecontrol.sourceforge.net/; http://www.realsolve.co.uk/site/tech/dbunit-quickstart.php

3 Anwendung und Evaluierung des Testframeworks im Projekt P1

Das Framework wurde zum ersten Mal bei einem mittleren Entwicklungsprojekt eingeführt und evaluiert. Das Projekt P1 beschäftigte sich mit der Entwicklung einer Applikation zur Verwaltung von Stammdaten von sozialversicherten Personen. Als Projektlaufzeit war ein Kalenderjahr vorgesehen. Das Projektteam umfasste ca. 30 Personen, zusammengesetzt aus Mitarbeitern des Sozialversicherungsträgers und Mitarbeitern von zwei großen Softwaredienstleisterfirmen.

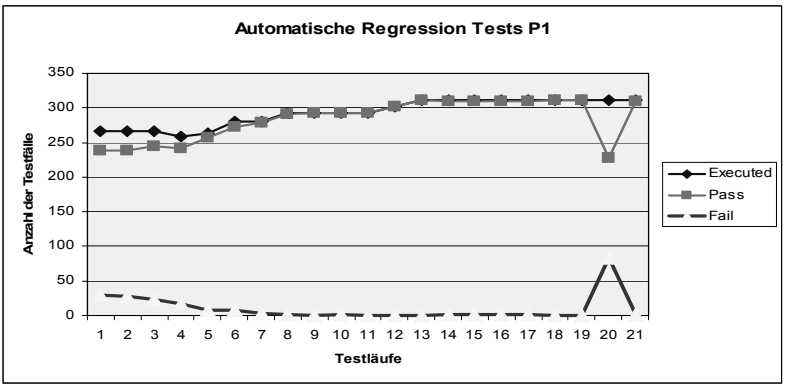


Abbildung 1: Ergebnisse der Regressionstests

Die Testmanagement-Gruppe wurde bereits am Beginn der in drei Iterationen geplanten Entwicklung involviert. Nach jeder Iteration wurde das Feedback der Anwender berücksichtigt. Das hatte zum Teil Rückwirkungen auf Funktionalität früherer Iterationen. Da ein Tool-gestütztes Anforderungs-Management vorhanden war und die Testfälle systematisch aus den Use-Case Beschreibungen abgeleitet wurden, konnte der Testfall-Pool aktuell gehalten werden. Insgesamt wurden 1200 Testfälle durch das Fach-Personal realisiert, wobei 10%-20% davon explorative Testfälle waren. 25% davon wurden mit dem Tool HP WinRunner automatisiert.

Zur Auswertung der Tests wurden Testfortschritts-Metriken und Fehlermetriken verwendet. Abb. 1 zeigt die Ergebnisse der Regressionstests. Ab dem 6.Lauf wurde die Qualität besser, beim 13.Lauf erfolgte die Abnahme, in der anschließenden Wartungsphase war die Qualität gut, bis auf dem Testlauf 20, vor dem ein *Refactoring* durchgeführt wurde. Die Fehlertrendanalyse in Abb. 2 zeigt die monatlichen Auswertungen aus Bugzilla. Zum Zeitpunkt M4 stieg die Anzahl der neuen Fehler stark an und die Gesamtzahl der offenen Fehler erreichte einen Höchstwert. Das Absinken der neu entdeckten und vor allem auch der insgesamt noch offenen Fehler zeigt dann die positive Entwicklung zum Produktivtermin hin. [KA03]

Zur Bewertung des Reifegrads des Testprozesses wurde das TPI-Modell [PK00] in P1 angewendet. Die Bewertung ergab, dass der Testprozess auf Basis des vorgestellten Frameworks insgesamt als „beherrschbar“ einzustufen ist. Einige Kernbereiche haben sogar bereits die Stufe „effizient“ erreicht.

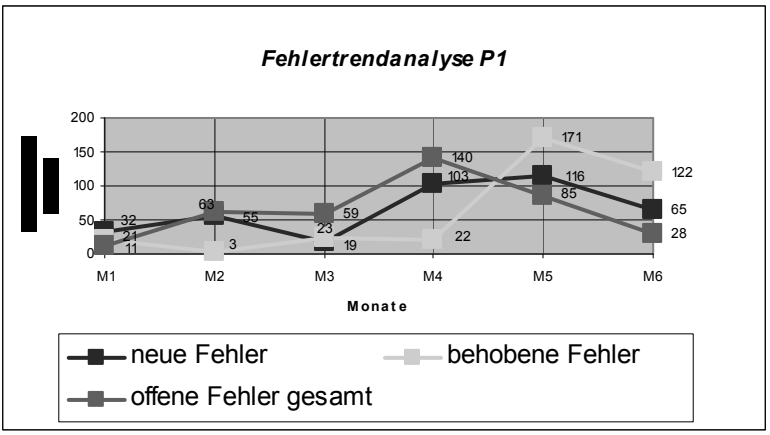


Abbildung 2: Fehlertrendanalyse P1

Eine große Herausforderung bestand darin, die Anwendbarkeit des Frameworks im Projekt P1 nachzuweisen, wobei dieses wie geplant erfolgreich im Jänner 2006 abgeschlossen wurde. Die anschließende Wartungsphase zeigte, dass keine gravierenden Fehler mehr auftraten und die geplanten Release-Zyklen eingehalten werden konnten. Inzwischen ist das Testvorgehen für alle Projekte verbindlich gemacht worden.

4 Zusammenfassung

Die wichtigsten Erfolgsfaktoren für den Einsatz des vorgestellten Testframeworks in der Sozialversicherung waren: (1) Aufbau von Test-Wissen und Erfahrung im Laufe einer iterativen Entwicklung durch spezifisches *Coaching* z.B. für Domänen-Experten und Entwickler, (2) Realisierung eines Testfall-Pools bestehend aus systematischen und explorativen Testfällen, (3) regelmäßige automatisierte Regressions-Tests für alle Teststufen und Last- und Stress-Tests. Die Wiederverwendung wird durch das einheitliche Testvorgehen nach ISTQB, den Know-How-Transfer innerhalb der Organisation, Metriken und *Templates* unterstützt. Als weitere Optimierungs-Maßnahme wird gerade die Einführung statischer Analyse-Werkzeuge vorbereitet.

Literaturverzeichnis

[KA03] Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley, 2003.
[PK00] Pol, M.; Koomen, T.; Spillner, A.: Management und Optimierung des Testprozesses. 1. Auflage, dpunkt.-Verlag, Heidelberg, 2000.
[SL05] Spillner, A.; Linz, T.: Basiswissen Softwaretest – Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB-Standard. 3. Auflage, dpunkt-Verlag, 2005.

Test Case Structuring and Execution Control in an Integration Framework for Heterogeneous Automatic Software Tests

Andreas Ganser[1], Holger Schackmann[1], Horst Lichter[1], Heinz-Josef Schlebusch[2]

[1]RWTH Aachen, Research Group Software Construction, Ahornstr. 55, 52074 Aachen
{Ganser|Schackmann|Lichter}@swc.rwth-aachen.de

[2]KISTERS AG, Charlottenburger Allee 5, 52068 Aachen
Schlebusch@kisters.de

Abstract: Dependencies between automatic test cases are considered as problematic, since they impair understandability and maintainability of these test cases. However, dependencies can not be fully avoided, when test cases require a time-consuming setup of the test fixture. This paper describes the handling of dependencies between test cases within an integration framework for heterogeneous automatic test tools that unifies test case administration, test execution and reporting of the test results. Two types of dependencies between test cases are identified. A simple approach is presented how one of these dependency types is utilized within the framework to gain control during test execution in order to reduce test execution time and unnecessary output.

1 Introduction

To develop and maintain complex software products different tools for automatic software tests are used. On the one hand this is caused by the need to pursue different kinds of tests like system tests, unit tests, or tests of non-functional qualities. On the other hand this is caused by the heterogeneity of the software under test in terms of implementations languages, technologies or underlying platforms. Hence, substantial know-how is necessary for applying these tools with a correct setup of the test environment, as well as for developing and maintaining the test cases. Moreover, test results can become scattered into several reports of different tools. As a consequence, this may lead to considerable overhead in the testing process that limits the acceptance and benefits of automatic testing. This was the motivation for the development of an integration framework for heterogeneous automatic test tools that unifies test case administration, test execution, and reporting of the test results [SLH07]. This framework was developed in cooperation with KISTERS AG. KISTERS AG offers software solutions for the energy markets and the management of the natural resources water and air, based on core software technologies for time series management of measurement values, modeling and forecasts. Due to the large volume of data needed to test these high capacity systems under realistic conditions this application

domain imposes additional challenges. To complete a test suite run in reasonable time, tests cases must share the same fixture or even be executed as chained tests [Mes07]. The resulting dependencies between test cases complicate test case maintenance. Thus, it is crucial to handle these dependencies within the integration framework.
In this paper several patterns for the integration of test cases into the central repository of the test framework will be presented. Then a characterization of dependencies between test cases is given. Further on, an approach will be presented how to utilize certain test case dependencies to reduce the run time of test suites and ease the analysis of test results. Finally, the capabilities of the framework will be discussed based on practical experiences with the integration of a representative test tool.

2 Structure of the Test Framework

This section briefly describes the basic concepts of the framework and introduces several recurring patterns of test structures. It is important to keep in mind that terms like test case and test plan have a slightly different meaning in the context of different test automation tools, as well as in the context of the test framework as described below.

2.1 Basic concepts

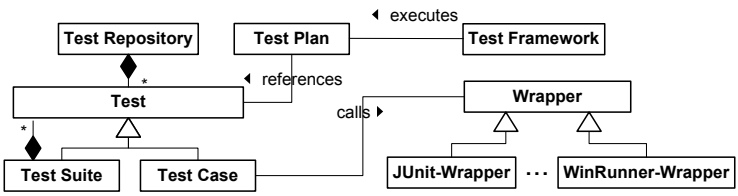


Figure 1: Basic concepts of the test framework

A sketch of the framework's basic concepts is given in Figure 1. The integration of an automatic test tool in the test framework is realized by a *wrapper* that executes an automatic test by a call to the external tool. Moreover, the wrapper is responsible for collecting the results of the tests. General information on the test result, like the test verdict or the execution time, is then delivered to the framework in a unified format. The wrapper can provide optional comments and links to additional files. The framework will merge this information into an overall report. The granularity of the results depends on the results reported by the wrapper. A wrapper must at least report a verdict like "passed" or "failed".
Tests are hierarchically organized into a *test repository* that is given by a directory structure of the file system and is put under version control. A *test suite* is represented by a directory that contains a script which defines pre- and post-processing actions that describe the

common setup and cleanup respectively. A *test case* is represented by a call to a wrapper within the test suite's script. While the test repository represents a static organization of all test cases, a test run is defined by the *test plan*. The test plan lists all related tests with their path within the test repository. Additionally pre- and post-processing sections can be defined. To ease portability and configurability of test cases, a test plan can define global variables. During a test run the tests will be executed in the order given in the test plan. All test results reported by the wrappers will be collected in an xml file and imported into a quality reporting system. This system archives each test run in a database, offers a unified presentation of the results, and enables access through a web interface.

2.2 Patterns of Test Structures

The following patterns are applied within the test framework in order to enable the integration of existing tests from different test tools and facilitate the maintenance of the tests. These patterns can be subdivided into patterns which describe the static structure of tests during design time, and patterns which describe the dynamic structure in order to control test execution.

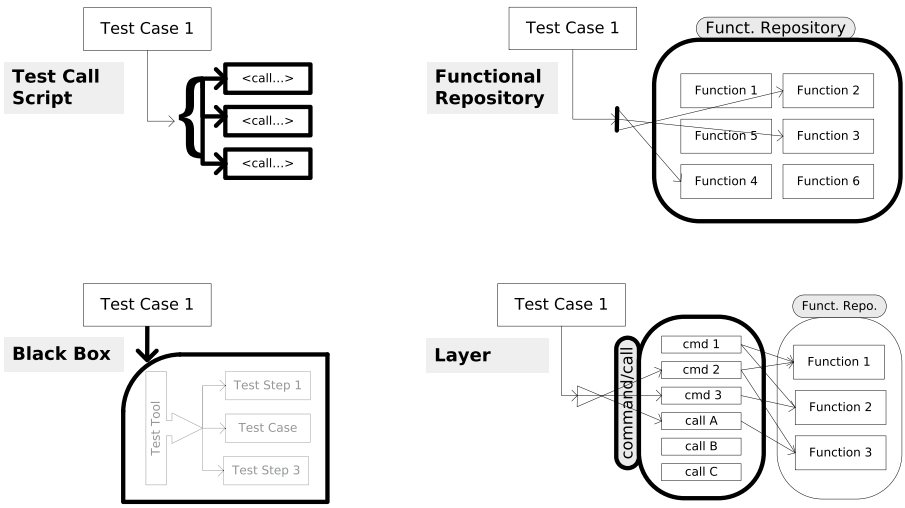


Figure 2: Patterns of static test structures

The patterns of static test structures comprise what we call *test call script*, *black box*, *functional repository* and *layer* pattern. Some sketches of the patterns are given in Figure 2. The names and ideas of the patterns are taken from literature [FG99] [Bin00] [Vig05].
A *test call script* implements one or more test cases by calls to the wrappers. Moreover it can encapsulate the static structures mentioned after.
In case the framework regards a test tool as a *black box*, it calls the test tool which takes on responsibility for the entire execution, the further selection of test cases or test suites,

and, most important, bears responsibility for the entire reporting and for the verdict.
A pattern of *functional repository* comprises a repository of functions from which test cases can be constructed. This repository works as a tool box with reusable functions and delivers no evaluation of the functions. Accordingly, the wrapper is responsible for the reporting and the verdict.
The *layer* pattern works similarly to the functional repository but includes the evaluation of the test. The main difference between both patterns is the abstraction the layer represents. The idea is to encapsulate complex calls in simple names and, furthermore, provide an abstract code for tests.
The only pattern of a dynamic structure is the *test plan* as introduced in section 2.1. This pattern builds upon test call scripts.

3 Dependencies between Tests

A test case sometimes has dependencies to other test cases, for example *chained tests*, where a subsequent test case relies on data outputs or the system state that has been created by the preceding test. Such dependencies are considered as problematic, since understandability and maintainability of these test cases is impaired [Mes07] [Bla03]. But there are pragmatic reasons to run some tests as chained tests. If the setup of a test's fixture is very time consuming, like importing large data volume of measurement values, the setup time can be saved by using the data output or system state created by another test.
Hence we had to face the problem how to handle these dependencies within the test framework. Further investigation showed that two types of dependencies between tests can be identified (see section 3.1). In the following we introduce these types of dependencies and describe how the existing framework was enhanced with a lightweight controlling mechanism with respect to a verdict dependency between tests.

3.1 Chained-Test Dependency and Verdict Dependency

Examining dependencies, the point of view and the order of tests need to be mentioned. Firstly, tests are regarded how they are presented to the framework. This means a test case is a call of a wrapper as illustrated in Figure 1 and, therefore, the smallest element the framework is able to recognize and control. Notwithstanding, it might later be necessary to inspect test cases of a test tool which are called by the wrapper. Secondly, test cases are always ordered in a linear sequence. Therefore the perspective from a single test case can be forwards and backwards which describes the preconditions and the postconditions respectively. No matter what perspective is taken the relationships between two test cases are subsumed and referred to as dependency. A backwards perspective is usually described as follows:

$$\text{if } X \text{ then } proceed \text{ else } omit$$

But what is condition X? Simply put, it can be the status of the system or an aggregation of the test's outputs, e.g. the test verdict.

Whenever the status of the system is the condition, the tests are chained tests. This means the test executed beforehand produced the preconditions for the upcoming test. This needs to be avoided because chained tests are difficult to maintain and understand [Mes07]. Nevertheless those chained tests are found in reality very often. This is because bringing the entire system in the required status is often very tedious and time-consuming. This means test chains are intended to save time and money. The other side of the coin is, the overall status of the system can not be assured due to the amount of data you have to ensure.
In case the condition is an aggregation or interpretation of some output, the most condensed form is a verdict like “passed” or “failed”. Such verdicts are the pieces of interest in this paper, because the decision to execute or to omit a test case will be made based on the verdicts. A verdict is usually determined after the execution of a test case by comparing actual outputs with expected outputs. Hence the status of the system is not involved with how the decision is made.

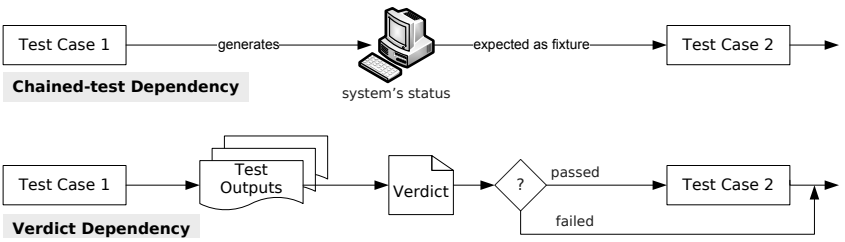


Figure 3: Dependencies between test cases

Talking in terms of dependencies, there are two distinguishable kinds as shown in Figure 3. To begin with, the *chained-test dependency* subsumes all conditions based on the status of the system; next, the *verdict dependency* comprises all conditions concerning verdicts.

Merging chained-test dependencies and verdict dependencies, the latter is a subset of the former. Indeed testers often like implying from verdicts to the status of the system. Hence chained-test dependencies are often applied in the context of test case design without evaluating the overall status of the system, but simply using some verdicts.

3.2 Verdict Dependencies in the Test Framework

The verdict dependencies need to be examined at different levels of test design. First, a look at the level of test cases is needed and, second, a look inside a test case is needed to regard test steps. At the level of test cases the verdict dependencies are quite useful. In fact verdicts open up an elegant way to gain control about test runs. This control can be done with a little help of knowledge about previously run tests. So, it might be possible to deduce whether the upcoming test case is likely to succeed. If there is no way it will succeed it can be omitted. For example, a failed login in the first test case makes a second useless which logs in and tries to change the user’s name. Omitting the second test case reduces the execution time and keeps the tester away from inspecting outputs which deliver

no further insights. Therefore, the verdict dependencies became the idea the framework was improved with.
Since the framework bases test runs on test plans (section 2.1) and should evaluate verdicts during run time, the verdict dependencies need to make an impact on the dynamic pattern mentioned in section 2.2. Hence, the description of a test element was extended with a keyword "requires_passed" as shown in Figure 4. This keyword models the verdict dependency of the current test case and takes a list of test cases which define its precondition.

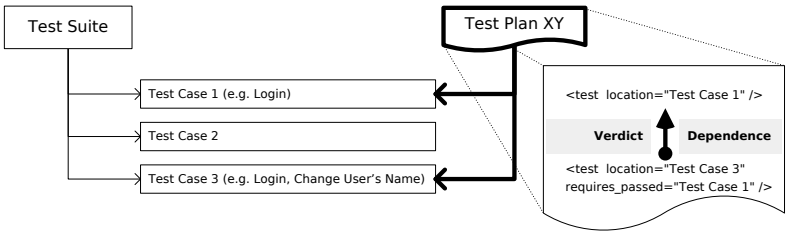


Figure 4: Sketch of a verdict dependency in a test plan

4 Integration of a Representative Testing Approach

The main intention for considering verdict dependencies lies in optimizing test runs and outputs. But testing approaches need to be well structured and designed to gain full benefits from these dependencies. Therefore, a representative testing approach was integrated into the framework in order to assess the usefulness of the concept. A model for evaluating tests in some sort is presented in [ZVS+07], which bases on the ISO model for software quality [ISO]. An appropriate language to name problems, in terms of test smells, was found in the area of object oriented xUnit test patterns [Mes07]. The above mentioned chained tests therefore suffer from interacting tests in terms of test smells.
Such and alike problems were investigated during integrating an application specific test tool and the existing test suites into the framework [Gan07]. This tool implements a typical testing approach referred to as *reference version approach* [FG99]. It compares expected outputs, the references, with the obtained outputs in order to find differences. Whenever a difference is found, the test is assumed to have failed. Therefore the expected outputs need to be prepared beforehand. These references can be of many different types: text files, screen shots, dumps of databases et cetera.
With respect to the above mentioned patterns, the approach was transformed from a black box pattern to a test call script pattern. In detail the testing approach was split up in controllable parts, what is one call of the application specific test tool per test case. This enabled the integration of this approach in the framework with the test call scripts necessary for the wrapper calls. Moreover, the verdict dependencies could be applied to the testing approach, and the results of the comparison were improved by parsing these results into xml-formatted documents. These documents can now be imported to the company's

reporting system which increases the acceptance of the framework.
Regarding the quality model, the evaluation was done slightly altered, because the evaluation was not of a test specification but a testing tool. Most important were the strong improvements of understandability, analyzability and the improvement of the test repeatability. By contrast, the time behavior is affected negatively due to the overhead the framework imposes. Besides the usage is more complex since the framework requires more configuration settings. But we think that the improved controllability, the embedding of the framework in the application domain and the better understandability of the test outweigh the mentioned disadvantages.
In terms of test smells, we were able to eliminate a smell called "manual result verification" in consideration of condensing the test run to a single verdict. This was achieved because on the one hand the verdicts exist at every stage of abstraction, from test steps over test cases up to test suites and test plans. On the other hand a parser transforms variances in the comparisons into verdicts. Moreover, the framework's ability to ease automatic test runs and result analysis reduced the danger of "infrequently run tests".

Over all the improved controllability of the testing approach finally has facilitated the usage of the verdict dependencies and, the benefits and the simplicity of the concept lead to a positive acceptance. Therefore, the verdict dependencies are expected to be employed increasingly in test suites with tedious fixtures.

5 Experiences and Outlook

With the initial assumption of side-effect free test cases, some existing tests could only be integrated as black box. The usage of verdict dependencies enables the integration on a more fine-grained level and facilitates detailed control of the test case execution by the test framework. Describing verdict dependencies as an optional precondition for a test makes existing dependencies explicit within a test plan. Hence understandability of the test plan is substantially improved.
Moreover, verdict dependencies can be utilized to run very coarse-grained tests as regression tests, especially of high capacity systems. But in case something fails, more details about the defect should be found out. This is what we call a *top-down test strategy*. For this purpose we introduced an attribute named "requires_failed" which works the opposite way of the "requires_passed". That is the test is only executed in case the named tests in the attribute failed or were not executed. Of course both attributes can be combined.

Further development might involve verdict dependencies for parallelizing the execution of test plans; in other terms "load-balancing" the test plan or test run. Therefore our verdict dependencies are important as a constraint how it is reasonable to distribute the execution of test cases. For example, consider a given test plan with a very long run time, e.g. one day. But this test plan must be done within four hours. In case the test plan can be distributed along verdict dependencies on several machines, maybe four hours will be enough.
Note that parallelizing test plans is different from running test cases for a distributed system. An enhancement of the framework that enables the description of such distributed test scenarios has been implemented as a prototype [Ada07].

6 Summary

In a nutshell, we introduced a framework for automated testing in a heterogeneous testing environment and showed how this approach on high-level test tool controlling can be optimized. In order to come up with a clean concept, we examined possible dependencies between tests and distinguished between chained-test dependencies and verdict dependencies. Though chained-test dependencies are a common way to reduce effort in fixtures, there is no way to gain proper control for the framework. Instead the verdict dependencies open up the opportunity for controlling a test run. For this purpose a test plan must include information about preconditions in terms of verdicts about previously run tests. In case the preconditions are not fulfilled, the execution of a test case is omitted. Using verdict dependencies relies on a certain fine-grained structure of a test case with respect how the test tool works. Therefore some structures were examined and some hints about how to include a testing approach were given.

References

[Ada07] Sofia Adamanova. Distributed execution of heterogeneous automatic software tests. Master Thesis, RWTH Aachen, 12 2007.

[Bin00] Robert V. Binder. *Testing Object-Oriented Systems*. Addison-Wesley, 2000.

[Bla03] Rex Black. *Critical Testing Processes: Plan, Prepare, Perform, Perfect*. Addison Wesley, 2003.

[FG99] Mark Fewster and Dorothy Graham. *Software Test Automation - Effective use of test execution tools*. Addison Wesley Logman Limited, ACM New York, 1999.

[Gan07] Andreas Ganser. Testdesign für automatische Softwaretests. Diplomarbeit, RWTH Aachen, 11 2007.

[ISO] ISO/IEC 9126. *Software engineering Product quality Part 1: Quality model, Part 2: External metrics, Part 3: Internal metrics, Part 4: Quality in use metrics*. International Standards Organization.

[Mes07] Gerard Meszaros. *XUnit Test Patterns*. Addison-Wesley, 2007.

[SLH07] Holger Schackmann, Horst Lichter, and Veit Hoffmann. An integration framework for heterogeneous automatic software tests. In Bleek, Schwentner, and Züllighoven, editors, *Software Engineering (Workshops)*, volume 106 of *LNI*, pages 107–112. GI, 2007.

[Vig05] Uwe Vigenschow. *Objektorientiertes Testen und Testautomatisierung in der Praxis*. dpunkt Verlag, 2005.

[ZVS+07] Benjamin Zeiß, Diana Vega, Ina Schieferdecker, Helmut Neukirchen, and Jens Grabowski. Applying the ISO 9126 Quality Model to Test Specifications. In *Software Engineering 2007 (SE 2007, Lecture Notes in Informatics (LNI), volume 105)*, pages 231–242. Köllen Verlag, Bonn, März 2007.

Ein Framework für die Testautomatisierung bei Fahrer-Assistenz-Systemen

Vladimir Entin, Klaus Meyer-Wegener

Department Informatik, Lehrstuhl für Informatik 6 (Datenbanksysteme)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
vladimir.entin@informatik.uni-erlangen.de
klaus.meyer-wegener@informatik.uni-erlangen.de

Abstract: In dieser Arbeit wird ein Framework für die Testautomatisierung bei Fahrer-Assistenz-Systemen vorgestellt, das im Rahmen eines INI.FAU-Projekts (www.ini.fau.de) prototypisch realisiert worden ist. Es wird auf zwei Probleme der Testautomatisierung eingegangen: Testprozessbeschreibung und Schaffung einer geeigneten Architektur zur Testprozessausführung. Der vorgestellte Ansatz ist ebenfalls auf andere Bereiche anwendbar, in denen eine heterogene Tool-Landschaft und eine sehr große Menge aufgezeichneter Messwerte als Test-Eingangsdaten eingesetzt werden.

1 Testvorgänge bei der Entwicklung von Fahrer-Assistenz-Systemen

Die modellbasierte Funktionsentwicklung in der Automobilindustrie hat in den letzten Jahren breite Akzeptanz gefunden. Etablierte Entwicklungs- und Testverfahren wie Modell-basierte Simulation mit künstlich erzeugten Eingangsdaten sind jedoch nicht imstande, reale Testfahrten vollständig zu ersetzen. Die zu entwickelnde Funktion wird dafür in Form eines C-Codes auf die Steuereinheit übertragen. Während anschließender Testfahrt beobachtet der Funktionsentwickler die Verhaltensweise seines Algorithmus. Im Bereich Vorentwicklung von Fahrer-Assistenz-Systemen (FAS, z.B. Adaptive Cruise Control) wird der Funktionscode nicht ausschließlich aus Modellen abgeleitet, sondern es werden komponentenbasierte Entwicklungsumgebungen zu seiner Erzeugung eingesetzt. Dazu gehört beispielsweise das Automotive Data And Time Triggered Framework (ADTF). Es erlaubt es dem Benutzer, die entwickelten Software-Module in die Bus- und Hardware Architektur von Versuchsfahrzeugen zu integrieren [Sc07]. Die Testfahrten stellen einen hohen Kostenfaktor dar und können deshalb nicht jedes Mal durchgeführt werden, wenn ein neuer Software-Stand der zu testenden Funktion vorliegt. Aus diesem Grund werden Messwerte wie beispielsweise CAN-Botschaften im Binärformat abgespeichert. Die Größe einer solchen Datei kann bis zu siebzig Gigabyte pro Stunde Messfahrt betragen. Grund dafür sind Videoaufzeichnungen, die zum Zweck der Bildverarbeitung unkomprimiert abgespeichert werden müssen. Die Messdateien, beziehungsweise zeitliche Ausschnitte (Szenarien) davon, werden in einer Szenarien-Datenbank verwaltet [En06]. Durch die Aufzeichnung aller Messwerte ist der

Funktionsentwickler imstande, Systemtests [Li06] nach dem Blackbox-Prinzip im Labor durchzuführen. Die charakteristische Eigenschaft solcher Tests besteht in der Art der Testfall-Definition. Diese geschieht, indem eine Menge von Fahrmanövern festgelegt wird, die während einer Messfahrt bei bestimmten Wetterbedingungen und Fahrbahnzuständen abgefahren werden müssen. Bei den klassischen Systemtests setzt man hingegen auf automatisierte oder manuelle Ableitung von Testfällen samt zugehöriger Test-Eingangsdaten von der Systemspezifikation [Li 06]. Zur Testdurchführung sind Konfigurationsschritte notwendig, die mit steigender Anzahl an Messdateien einen hohen manuellen Aufwand für den Funktionsentwickler bedeuten. Ohne einen automatisierten Ansatz ist also ein umfangreicher Systemtest in der Vorentwicklung von Fahrer-Assistenz-Systemen (FAS) kaum möglich.

2 Realisierung der Testautomatisierung

Zur Realisierung des automatisierten Testablaufs sind grundsätzlich zwei Probleme zu lösen. Zum einen muss eine flexible Prozessbeschreibung geschaffen werden, die es erlaubt, die unterschiedlichsten am Testablauf beteiligten Anwendungen zu parametrisieren und die Datenflüsse zwischen diesen darzustellen. Zum anderen muss eine geeignete Softwarearchitektur entworfen werden, die die Ausführung des im ersten Schritt beschriebenen Testprozesses erlaubt.

2.1 Testprozessbeschreibung

Gerade bei einem automatisierten Ablauf mehrerer Anwendungen, die Daten untereinander austauschen, ist es notwendig, im Vorfeld zahlreiche Parametrisierungen vorzunehmen. Oftmals bestehen gewisse Abhängigkeiten zwischen den Konfigurationsparametern, die eine schrittweise Vorgehensweise bei der Parametrisierung erfordern. Beispielsweise können keine Szenarien-IDs aus der Datenbank automatisiert ausgelesen werden, wenn man davor die Zugangsparameter wie beispielsweise Port oder Serveradresse für die letztere nicht spezifiziert hat. Außerdem weisen unterschiedliche Anwendungen verschiedenste Parametersätze auf, weshalb vor der Parametrisierung zunächst festgelegt werden muss, welche Anwendungen am Testvorgang beteiligt sind. Um der variablen Testprozess-Zusammensetzung gerecht zu werden, aber auch Integrierbarkeit diverser Anwendungen in den automatisierten Testvorgang zu gewährleisten, wurde der Ansatz der Process-Driven Architecture (PDA) gewählt [Mu07]. PDA unterscheidet zwischen domänenspezifischer Prozessmodellierung, systemspezifischer Prozessmodellierung und plattformspezifischer Modellierung. Durch schrittweise Abbildung domänenspezifischer Modelle auf systemspezifische und anschließend systemspezifischer auf plattformspezifische wird der Testprozess soweit beschrieben, dass er vom entsprechenden Framework ausgeführt werden kann (siehe Abschnitt 2.2). Bei der domänenspezifischen Prozessmodellierung geht es darum, das Testprozessmodell mithilfe von abstrakten Typen (z.B. Entwicklungsumgebung, Testdaten-Speicherort) sowie deren Verbindung untereinander

durch Datenflüsse darzustellen. Bei der Abbildung auf das systemspezifische Prozessmodell müssen für jeden abstrakten Typ die Anwendungsbezeichner angegeben werden. Anschließend geht es darum, das systemspezifische Prozessmodell zu erstellen, indem die anwendungsabhängigen Parametrisierungssätze teilweise in mehreren Schritten angegeben werden. Letztlich muss das systemspezifische Prozessmodell um plattformspezifische Eigenschaften ergänzt werden. So müssen u.a. für jede Anwendung Framework-Plugins samt Pin-Verbindungen (siehe Abschnitt 2.2) angegeben werden. Somit entsteht das plattformspezifische Prozessmodell. Anschließend wird das plattformspezifische Modell vom Testframework [Iß07] zur Ausführung gebracht.

2.2 Framework-Architektur zur Testprozessausführung

Da der automatisierte FAS-Testprozess weitgehend ohne Interaktion mit dem Entwickler verläuft und auf Dateiaustausch zwischen den beteiligten Anwendungen basiert, wurde als Architekturgrundlage für das Framework das Pipes-and-Filters-Muster gewählt [Bu00], wobei Filters als Plugins realisiert worden sind. Dadurch wird zum einen die bereits erwähnte Integrierbarkeit der Anwendungen in den Testprozess garantiert. Zum anderen wird eine variable Testprozess-Zusammensetzung ermöglicht, indem bestimmte Plugins mehrfach in unterschiedlichen plattformspezifischen Prozessmodellen verwendet werden können. Das Framework implementiert drei wesentliche Mechanismen (Abb. 1).

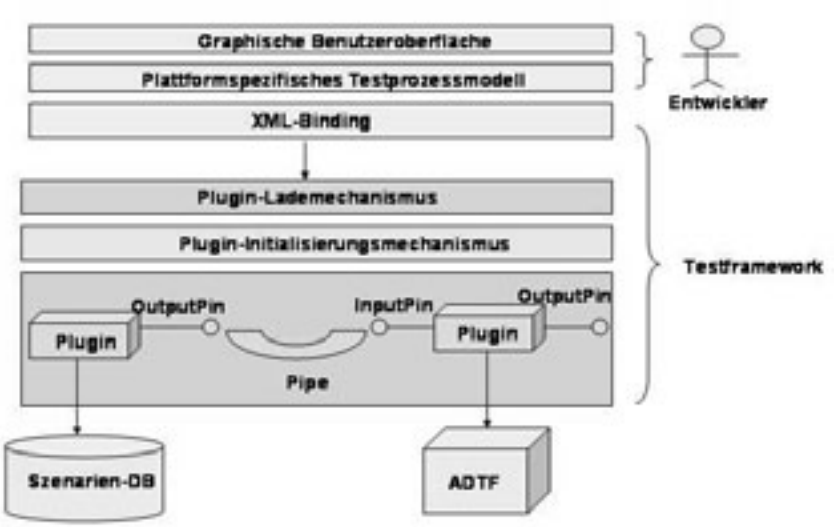


Abb. 1 Testframework-Architektur

Zunächst muss der Entwickler durch eine entsprechende graphische Oberfläche in mehreren Schritten das plattformspezifische Testprozessmodell erzeugen. Dieses wird als XML-Dokument dem Testframework übergeben. Letzteres liest über den XML-Data-Binding-Mechanismus [Xb07] zunächst die Informationen über die beteiligten Plugins aus. Anschließend werden die benötigten Plugins als Assemblies geladen. Bei der

Plugin-Initialisierung geht es darum, die Pipes zwischen den Plugins zu erzeugen. Im letzten Schritt wird die eigentliche Ausführung gestartet. In dieser werden Schritt für Schritt über entsprechende Plugins die am Testvorgang beteiligten Anwendungen angesteuert.

3 Zusammenfassung und Ausblick

Es wurde ein bei AUDI AG prototypisch umgesetztes Framework zur Testautomatisierung bei FAS in der Vorentwicklung vorgestellt. Dieses erlaubt, unterschiedliche Konstellationen von Anwendungen zu einem automatisierten Testvorgang zu bringen. Aktuell wird das Framework im Bereich der Bildverarbeitung insbesondere bei der Verfolgung von Fahrzeugen verwendet. Die ersten Erfahrungen haben gezeigt, dass die Auswahl einer minimal notwendigen Szenarienmenge entscheidend für die Akzeptanz des Frameworks ist. Außerdem hat sich eine formularbasierte graphische Oberfläche zur Erstellung der Testprozessbeschreibung als wenig flexibel erwiesen. In der Zukunft soll eine auf Kombination aus Workflow und Formular basierende graphische Oberfläche geschaffen werden. Der vorgestellte Ansatz ist keineswegs auf den automobilen Bereich beschränkt. Der Einsatz eines solchen Frameworks wäre überall dort denkbar, wo eine heterogene Tool-Landschaft und eine sehr große Menge aufgezeichneter Messwerte zum Testen und Entwickeln von Software eingesetzt werden, wie etwa in der Medizin.

Literaturverzeichnis

[Bu00] Buschmann, F.: Pattern-orientierte Software-Architektur. Addison-Wesley. München, 2000.

[En06] Entin, V.: Datenbank zur Verwaltung von Fahrszenarien. Diplomarbeit. Universität Erlangen-Nürnberg, Audi Electronics Venture GmbH. Gaimersheim, 2006.

[Iß07] Ißbruecker, M.: Konzeption und Erstellung einer Umgebung zur Anfertigung und Ausführung von Testprozessen für Fahrer-Assistenzsysteme. Diplomarbeit. Westsächsische Hochschule. Zwickau, 2007.

[Li06] Liggesmeyer, P.: Software-Qualität. Spektrum-Verlag Heidelberg. Berlin, 2002.

[Mu06] Müller, S.: Modellbasierte IT-Unterstützung von wissensintensiven Prozessen. Dissertation. Universität Erlangen-Nürnberg, Erlangen 2007.

[Sc07] Schabenberger, R.: ADTF: Framework für Fahrerassistenz-und Sicherheitssysteme. Audi Electronics Venture GmbH. VDI 13. Internationaler Kongress “Elektronik im Kraftfahrzeug” Baden Baden, 2007.

[Xb07] Bourret, R.: XML Data Binding Resources. www.rpbourret.com/xml/XMLDataBinding.htm Letzter Aufruf: 16.12.2007.

Using UML Environment Models for Test Case Generation

Maritta Heisel[1], Denis Hatebur[1,2], Thomas Santen[3], and Dirk Seifert[4]

[1] University Duisburg-Essen, Faculty of Engineering, Department of Computational and Cognitive Sciences, Working Group Software Engineering, Germany,{denis.hatebur, maritta.heisel}@uni-duisburg-essen.de
[2] Institut für technische Systeme GmbH, Germany, d.hatebur@itesys.de
[3] Department of Computer Science, Technische Universität Berlin, Berlin, Germany, santen@cs.tu-berlin.de
[4] LORIA – Université Nancy 2, Campus Scientifique BP 239, F-54506 Vandœuvre lès Nancy cedex, dirk.seifert@loria.fr

Abstract. We propose a new method for system validation by means of testing, which is based on environment models expressed as UML state machines. A sun blind control case study serves to illustrate the method.

1 Introduction

Model-based software development proceeds by setting up *models* of the software to be constructed. This approach has proven useful, because it allows developers to first elaborate the most important properties of the software before proceeding with the implementation. Often, software models are also used for code generation. In this case, however, a problem arises: it does not make sense any more to test the software against its models, because these were already used to generate it. We therefore propose to test the software not only against its *specification* (i.e., against the models), but also against its *requirements*, which describe the how the *environment* should behave in which the software will be operating (acceptance testing). For this purpose, we have to set up a model of the environment, too.

In this paper, we describe how UML state machines (with a corresponding support tool TEAGER [SS06]) can be used to realize the described approach in the area of reactive and/or embedded systems. For this kind of system, state machine models are particularly useful. We elaborate on two different testing approaches:

On-the-fly testing: Here, generating and executing test cases is intertwined. This has the advantage that state explosion is not a problem, but the disadvantage that for non-deterministic systems the tests may not be repeatable.

Batch testing: Here, test cases are generated and stored for later execution. This has the advantage that regression tests become possible but the disadvantage that all possible behavior variants must be computed.

In Sect. 2, we introduce Jackson's terminology [Jac01], which clarifies the notions used in the rest of the paper. The sunblind example is presented in Sect. 3. In Sect. 4, we describe our testing approach, presenting different test architectures. In Sect. 5, we

present state machine patterns that help to set up environment models. Some experimental results are given in Sect. 6. Sect. 7 discusses related work, and Sect. 8 summarizes our contributions.

2 Terminology

Jackson's [Jac01] terminology serves to clearly distinguish the different notions that have to be taken into account when developing software:

Machine is the thing we are going to build; it may consist of software and hardware.
Environment is the part of the real world where the machine will be integrated.
System consists of the machine *and* its environment.[5]
Requirements are *optative* statements; they describe how the *environment* should behave when the machine is in action.
Specifications are *implementable* requirements; they describe the machine and form the basis for its construction.
Domain knowledge is needed to transform requirements into specifications. It is expressed as *indicative* statements. We distinguish between facts and assumptions: **Facts** describe what holds in the environment, no matter how we build the machine. **Assumptions** describe things that cannot always be guaranteed, but which are needed to fulfill the requirements, e.g., rules for user behavior.

The domain knowledge D consists of both the facts F and the assumptions A: $D \equiv F \wedge A$. The relation between requirements and specifications is $S \wedge D \Rightarrow R$, i.e., we have to show that if we build the machine such that it satisfies the specification S and integrate it into an environment for which D holds, then the requirements R are satisfied.

3 Example

We illustrate our testing approach with the example of a sunblind control system. The task is to write software that controls a sunblind, taking into account user commands, wind, and sunshine: The sunblind can manually be lowered or pulled up. It is automatically lowered on sunshine for more than one minute. The sunblind can be destroyed by heavy wind, which should be avoided. The environment consists of user, sun and wind.

To illustrate the difference between requirements and specifications and to stress the importance of explicitly modeling the environment, we transform one requirement concerning the sunblind control problem into a specification, making use of domain knowledge.

R1 *The sunblind is not destroyed by wind.*

To make this requirement implementable, we must know when wind can destroy the sunblind, and how a destruction can be avoided:

[5] In the literature, the term "system" is used interchangeably for the software (i.e., the machine) as well as for the machine in its environment. Jackson's terminology establishes a clear distinction.

F1 *Heavy wind for more than 30 sec is destructive.*
A1 *Heavy wind for less than 30 sec is not destructive.*
F2 *If the sunblind is up, it cannot be destroyed by wind.*

Using this domain knowledge, we can replace $R1$ by

R1' *The sunblind is up if there is heavy wind for more than 30 sec.*

because $F1 \wedge\ A1 \wedge F2 \wedge R1' \Rightarrow R1$. Next, we use

F3 *It takes less than 30 sec to pull up the sunblind.*

to obtain

R1" *If there is heavy wind and the sunblind is not up, it is pulled up.*

because $F3 \wedge R1'' \Rightarrow R1'$. Using the facts

F4 *There is heavy wind if and only if the wind sensor generates more than 75 pulses per sec.*
F5 *Turning the motor left pulls up the sunblind.*

we finally obtain the specification

S1 *If the wind sensor generates more than 75 pulses per sec and the last signals to the motor have not been* turn left*, followed by* motor left blocked *and* stop motor*, then the* turn left *signal is sent to the motor.*

(because $F4 \wedge F5 \wedge S1 \Rightarrow R1''$), which is quite different from the requirement we started out with. All in all, we have shown $F1 \wedge F2 \wedge F3 \wedge F4 \wedge F5 \wedge A1 \wedge S1 \Rightarrow R1$.

4 Test approach

How would the sunblind control software (SUT, system under test) be tested? Usually, conformance with the specification would be checked. In our example, we would have to verify that the machine generates the *turn left* signal. However, if the specification was not correctly derived from the requirements, the SUT passes the test nevertheless.

We therefore propose to test the SUT against the requirements. This means that we check whether the sunblind can enter a state where it would be destroyed. Besides detecting errors made in transforming requirements into specifications, testing against requirements allows us to verify that customer needs are satisfied (acceptance test).

In order to test the SUT against its requirements, we need a model of the environment, because the requirements refer to the environment and not to the machine. Much like the SUT, the environment can be modeled using UML state machines. The model explicitly contains the facts and the assumptions about the environment. The environment model consists of adapters and the input event generator: Adapters transform abstract events such as *pull up sun blind* into concrete ones, such as *turn motor left*. The input event generator produces abstract events. To capture stochastic properties of the environment probabilistic state machines of TEAGER can be used. This reduces the number of inadequate test cases.

The requirements are translated into state machines, too. These state machines serve to inform the tester whether a requirement is violated. They observe the stimuli and SUT outputs at an adequate level of abstraction. As shown in Fig. 1, the Test Case Generator component of the tool TEAGER can be used to simulate the environment model and to check the requirements. To calculate test cases, for each tick (1) an abstract stimulus is generated by the Input-Event-Generator in the environment model. Adapters transform

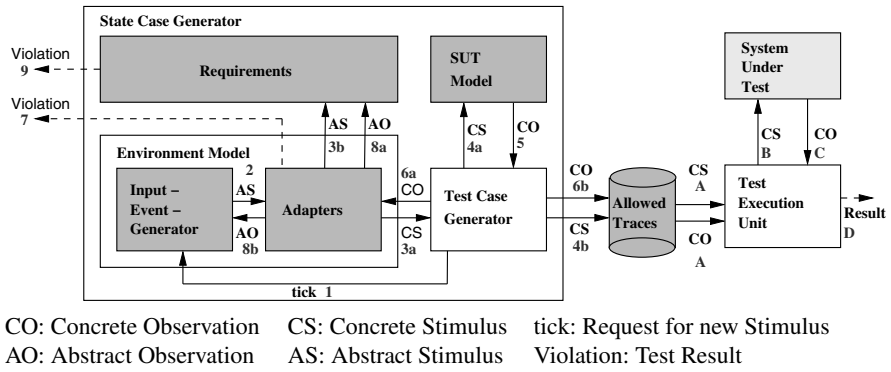


Fig. 1. Test architecture for batch testing.

the abstract stimuli into concrete stimuli for the Test Case Generator (3a) and send the abstract stimuli to the Requirements (3b). The Test Case Generator sends the concrete stimuli to the SUT Model, which determines suitable responses (4a, 5), and it stores the conrete stimuli and the determined concrete observations (4b, 6b). The Adapters transform the concrete observations (6a) into abstract observations that are checked by the Requirements (8a) and used to generate reasonable stimuli (8b, e.g., *isLowered* only after *LoweredSunBlind*). Violations can be detected by checking the requirements (9) and while transforming concrete stimuli into abstract stimuli (7). After the requirements are checked, a new tick (1) is generated. The generated Test Cases can be used to test the SUT with the Test Execution Unit. Concrete stimuli and observations in the allowed traces (A) are used to stimulate the SUT (B) and check the responses (C). Test results (D) are the output of the Test Execution Unit.

Alternatively, the environment model can be directly connected to the SUT, and within the simulated environment the requirements are checked at runtime. In this case no SUT model is necessary. This scenario is especially useful for acceptance tests. Here, regression tests can be performed by saving the generated event sequences. Then, the State Machine Executor recalculates the test events by simulating the environment and checks the observed behavior according to the requirements. The test system architecture – annotated with sample observations and stimuli for the sunblind example and with the execution order – for this "on the fly"-testing approach is shown in Figure 2.

5 Patterns for environment models

Setting up the state machines for the environment model is not a trivial task. However, we can identify different patterns for setting up environment models, especially for expressing requirements as state machines. The overall structure of the state machine consists of parallel regions. That is, the environment model is in all of the parallel machines R_i, *Input-Event-Generator* and *Adapter* at the same time, and the different sub-machines communicate with each other via common events. The left-hand side of Figure 4 shows an example of an input event generator. Note that assumption *A1* (namely, that heavy wind for less than 30 sec is not destructive) is modeled explicitly.

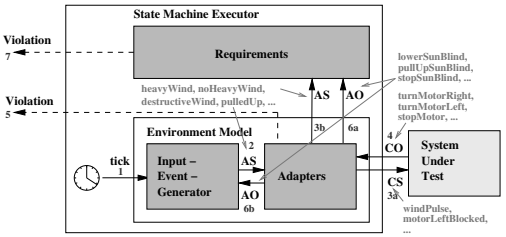


Fig. 2. Test architecture for on-the-fly testing.

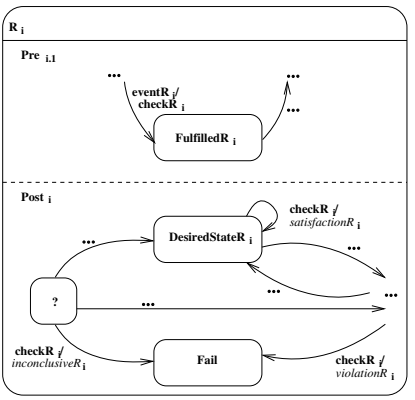


Fig. 3. Patterns for environment models

Moreover, probabilities for the different transitions are given. These can be processed by the TEAGER tool. As an example of an adapter, we present the motor adapter, which transforms concrete observations into abstract ones in the right-hand side of Fig. 4. It specifies how motor commands correspond to events that are visible in the environment.

For modeling requirements, we have developed different patterns, of which we can present only one for reasons of space. The pattern is usable when the requirement has the form "When [$eventR_i$] happens, [controlled domain] should be in [*desired-State*R_i]". Its representation as a state machine is shown in Fig. 3. When the event of interest happens, then the precondition of the requirement is fulfilled, and the event $checkR_i$ is generated. The state machine representing the postcondition contains the desired state and may also contain other states. Only if it is in the desired state, the test passes; otherwise, a violation is determined, or the test is inconclusive. The latter happens, for example, if the actual state of the system is not known. Then, the result of checking a requirement should neither be pass nor fail. In our example, we do not initially know the (physical) state of the sunblind. Hence, we introduced an "unknown state" (denoted by "?") expressing this situation. Checking requirement R_1 in this state yields an inconclusive result.

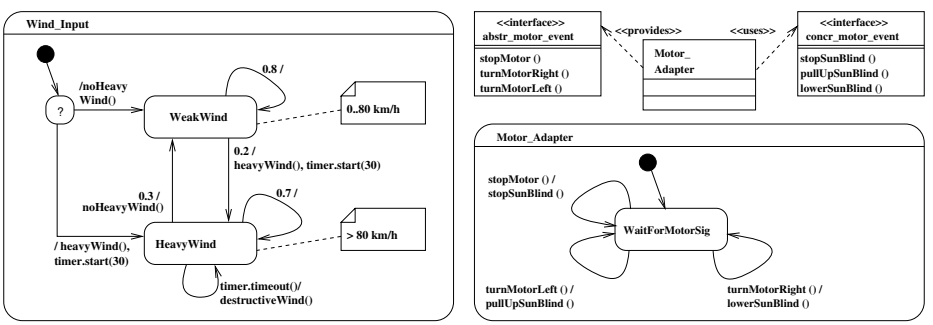


Fig. 4. Input generator and adapter for the sunblind

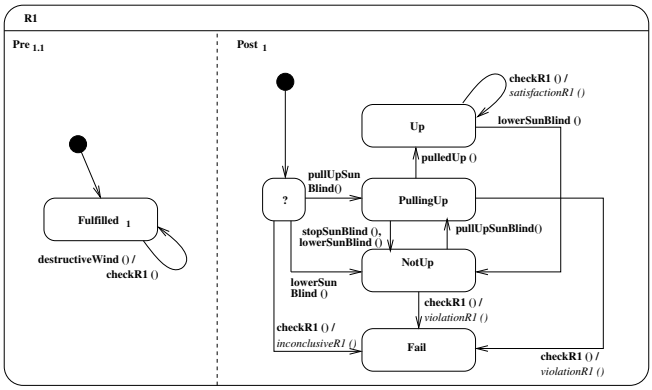


Fig. 5. State machine for requirement R_1

Requirement R_1 of Sect. 3 is an instance of this pattern: whenever there is destructive wind, the sunblind must be up. Figure 5 shows the instantiated pattern. Whenever the event *destructive Wind* occurs, the event *checkR1* is generated. If the sunblind is in state *up*, the requirement is satisfied. Otherwise, it is violated. The *Fail* state corresponds to a state where the sunblind would be destroyed.

All in all, to completely model the sunblind control problem, 4 input event generators, 4 adapters, and 7 requirement state machines have to be set up. All the requirements are instances of patterns.

6 Experimental results

To evaluate our approach, we used the tool suite TEAGER [SS06, Sei08]. The tool suite allows its users to generate and execute test cases or to directly stimulate the SUT. TEAGER logs the stimuli it sends to the SUT and the reactions of the SUT. During execution, these reactions are compared to the pre-calculated possible correct reactions to evaluate the test execution process. Table 1 shows some generation times for several experiments. Here, the number of triggers controls the length of test cases, while the search depth controls the exploration of the model's state space. The results show that no state explosion occurs, which makes the approach applicable in practice.

No. of Test Cases			No. of generated Triggers			Search Depth			Generation Time		
50	100	500	10	10	10	5	5	5	4 sec	5 sec	18 sec
50	100	500	100	100	100	5	5	5	15 sec	32 sec	150 sec
50	100	500	100	100	100	10	10	10	105 sec	180 sec	943 sec

Table 1. Test case generation times for several experiments.

7 Related work

To our knowledge, there neither exist approaches for testing requirements expressed as UML state machines, nor approaches for combining conformance testing on unit testing level with testing requirements on acceptance testing level.

A detailed overview of the fundamental literature for classical formal testing can be found in Brinksma's and Tretmans' annotated bibliography [BT01]. De Nicola and Hennessy [DH84] introduce a formal theory of testing on which Brinksma [Bri88] and Tretmans [Tre96] build approaches to derive test cases from a formal specification. In contrast to our work, these approaches assume that a testing process can communicate synchronously with the system under test. The developed tool TorX (`fmt.cs.utwente.nl/tools/torx`) also allows conformance testing of reactive systems.

Belli at al. (see [BH07] and the work cited there) base their testing methodology on a variant of state machines. In contrast to our approach, they do not test against requirements, but against a fault model that has to be set up explicitly. Moreover, they do not execute the state machines directly, but represent them as event sequence graphs.

Auguston et al. [AMS05] use environment models for test case generation. In contrast to our approach, they do not use state machines, but attributed event grammars.

While these works have their merits, we think that the combination of environment models and UML state machines for testing is a particularly attractive one.

Besides the work from academia, an increasing number of CASE Tool manufacturers offer components for model based testing. I-Logix Rhapsody, for example, offers two add-on products, Test Conductor and Testing and Validation, for testing state machines (`www.telelogic.com`). Simulink Verification and Validation generates test cases in Simulink and Stateflow, and measures test coverage for Statecharts (`www.mathworks.com`). Conformiq Software Ltd. offers a *Test Generator* which accepts "extended UML state charts" as the model of the system under test for dynamic testing (`www.conformiq.com`). *AsmL 2* by Microsoft provides an executable specification language based on the theory of Abstract State Machines (`research.microsoft.com/fse/asml`). The AsmL Test Tool supports parameter generation and test sequence generation based on interface automata.

8 Conclusion

We have developed a novel approach to testing reactive and embedded systems, based on environment models and using UML state machines. The approach is supported by the TEAGER tool. Using Jackson's terminology, we have defined uniform architectures and procedures for on-the-fly as well as batch testing that have the following characteristics:

- Requirements, facts, and assumptions are modeled explicitly.
- We have defined patterns for the different state machines: For requirements, a parallel state machine is set up for each precondition. When all preconditions are fulfilled, the postcondition is checked. Input generators and adapters also consist of parallel state machines, one for each item of the environment that generates stimuli or receives observations, respectively.

– Once these models have been set up manually (but systematically), the tests are performed automatically, using the tool TEAGER.

Our approach has the following advantages: When **requirements change**, in the test case generator only the state machine describing those requirements must be changed. On the other hand, changed requirements will lead to a new SUT model. The new SUT model can be validated while the test cases are generated. Modeling the facts and assumptions about the environment supports the **validation of requirements**. For example, it can be discussed if heavy wind can be destructive to the sunblind within the time that a sunblind needs to be pulled up. Although the model of the environment has nearly the same complexity as the model of the machine, a structured approach to develop the environment model helps to **identify subproblems** that can be treated separately. Sometimes, states like "sunblind destroyed" are not modeled in the machine, but must be modeled in the environment to verify that this state cannot be reached. On the other hand, states can be left out in the environment model if the machine implements features that are not part of the requirements. The same environment model can be **(re-)used** for a sunblind control that can stop at an arbitrary height and a sunblind control that can only open or close the sunblind completely. Modeling the environment adds **diversity** to the development process and thus helps to avoid that the same mistake occurs for test development and SUT development. This is because the test developers, who model the environment, must think in terms of the environment rather than the SUT behavior. In the environment model, a reasonable test case selection strategy can be defined, so that **no inadequate test cases** are generated. Atypical behavior can be identified and tested using a dedicated environment model.

References

[AMS05] M. Auguston, J. B. Michael, and M.-T. Shing. Environment behavior models for scenario generation and testing automation. In *Proc. First International Workshop on Advances in Model-Based Testing, ICSE 2005*, pages 1–6. ACM, 2005.

[BH07] Fevzi Belli and Axel Hollmann. Holistic testing with basic statecharts. In W.-G. Bleek, H. Schwentner, and H. Züllighoven, editors, *Software Engineering 2007 – Beiträge zu den Workshops*, LNI 106, pages 91–100. Ges. f. Informatik, 2007.

[Bri88] Ed Brinksma. A Theory for the Derivation of Tests. In *Protocol Specification, Testing and Verification*. North-Holland, 1988.

[BT01] Ed Brinksma and Jan Tretmans. Testing Transition Systems: An Annotated Bibliography. *Lecture Notes in Computer Science*, pages 187–195, 2001.

[DH84] Rocco De Nicola and M. C. B. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, pages 83–133, 1984.

[Jac01] Michael Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[Sei08] Dirk Seifert. The TEAGER Tool Suite. test execution and generation framework for reactive systems, 2008. `swt.cs.tu-berlin.de/~seifert/teager.html`.

[SS06] Thomas Santen and Dirk Seifert. Teager - Test Automation for UML State Machines. In B. Biel, M. Book, and V. Gruhn, editors, *Software Engineering 2006*, LNI P-79, pages 73–83. Gesellschaft für Informatik, 2006.

[Tre96] Jan Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software–Concepts and Tools*, 17(3):103–120, 1996.

Glassboxtest zur Testsuite-Optimierung

Rainer Schmidberger
rainer.schmidberger@informatik.uni-stuttgart.de
Universität Stuttgart, Institut für Softwaretechnologie
Universitätsstr. 38
70569 Stuttgart

Abstract: Der Glassboxtest kann mehr liefern als nur eine Prozentangabe der Programmcode-Überdeckung. Wenn das Glassboxtest-Werkzeug Informationen zum Ablauf der Testdurchführung erhält, ist eine wirksame Hilfestellung bei der Testsuite-Optimierung möglich. In diesem Artikel werden diese Hilfestellungen vorgestellt und am Beispiel des Glassboxtest-Werkzeugs CodeCover beschrieben.

1 Einführung

Die Literatur ist sich einig, dass der Glassboxtest einen wichtigen Beitrag zum Programmtest leisten kann. Zum einen kann Programmcodeüberdeckung als objektive Vollständigkeitsmetrik – z. B. als Testendekriterium – für einen Test verwendet werden. Zum anderen kann der Glassboxtest zur Qualitätsverbesserung einer Testsuite beitragen; der Glassboxtest soll auf fehlende oder redundante Testfälle hinweisen.

Betrachtet man den Glassboxtest im Vergleich zum Blackboxtest, so ist die Herleitung der Testfälle das unterscheidende Merkmal. Beim Blackboxtest wird die Spezifikation als Grundlage zur Herleitung der Testfälle verwendet, beim Glassboxtest werden die Testfälle aus dem Programmcode abgeleitet. Blackbox- und Glassboxtest unterscheiden sich aber zudem in ihrer prinzipiellen Zielsetzung: Während das Ziel beim Blackboxtest das Finden von Fehlern ist und somit die Qualitätsverbesserung des Produkts im Vordergrund steht, ist es beim Glassboxtest zunächst die Bewertung oder Verbesserung der Testsuite. Blackboxtest und Glassboxtest bilden keine Alternativen, sondern der Glassboxtest ergänzt den Blackboxtest.

Der Blackboxtest geht davon aus, dass die Spezifikation vollständig ist. Wir wissen, dass das in vielen Projekten nicht der Fall ist. Gerade bei Änderungen gegenüber einer ursprünglichen Spezifikation oder in der Wartung wird die Spezifikation oft nicht oder nur unzureichend gepflegt. Der Glassboxtest kann auf einen nicht ausgeführten Programmcode hinweisen, aus dem der Tester dann auf solche undokumentierten Anforderungen schließen kann. Welches Sollresultat für bestimmte Eingaben erwartet wird, muss aber außerhalb des Programmcodes und beispielsweise von den betreffenden Fachexperten ermittelt werden. Hier leistet der Glassboxtest keinerlei Hilfestellung.

Eine Werkzeugunterstützung ist beim Glassboxtest Voraussetzung. Am Markt sind viele Glassboxtest-Werkzeuge für viele der kommerziell wichtigen Programmiersprachen verfügbar [Ya06]. Die Produkte sind zum Teil kommerziell, zum Teil Open-Source, und als Resultat des Glassboxtests liefern die Werkzeuge typischerweise Angaben über Anweisungs- und Zweigüberdeckung. Auswertungsmöglichkeiten zur Programmcodeüberdeckung stehen mit diesen Werkzeugen zur Verfügung. Eine Hilfestellung für den Tester zur Qualitätsverbesserung der Testsuite wird von den Werkzeugen aber nur ansatzweise angeboten. In diesem Artikel wird das Open-Source-Glassboxtest-Werkzeug CodeCover vorgestellt, das speziell die Qualitätsverbesserung der Testsuite unterstützt.

1.1 Definitionen

In diesem Artikel wird Test im Sinne der Definition von Myers [My79] verstanden als „Programmausführung in der Absicht, Fehler zu finden“. Der Glassboxtest ist ein Test, in dessen Verlauf ausgeführte Programmelemente protokolliert werden. Als Programmelement sind prinzipiell alle Programmkonstrukte möglich, deren einzelne Ausführung erhoben werden kann und die Gesamtmenge bekannt ist. In der Praxis sind dies in der Regel Anweisungen (im Sinne der Grammatik), Zweige und Bedingungsterme. Testüberdeckung ist beim Glassboxtest das Verhältnis der ausgeführten Programmeinheiten zu allen im Programm enthaltenen Einheiten. Im Übrigen wird mit dieser Definition auch deutlich, dass eine Pfadüberdeckung nicht bestimmt werden kann. Die Ausführung einzelner Pfade kann zwar erhoben werden, die Gesamtmenge aller Pfade ist aber (in der Regel) unendlich.

Ein Testfall besteht aus Ausführungsbedingungen, einer Folge an Eingabedaten und einem Sollresultat. Während der Testdurchführung werden je Testfall die Eingabedaten in das zu testende System eingegeben und nach Abschluss der Eingabe wird das Sollresultat geprüft. Der Beginn der Eingabe wird im folgenden als Testfallbeginn, die Sollresultatsprüfung als Testfallende bezeichnet. Eine Testsuite umfasst mehrere Testfälle.

1.2 Verwandte Arbeiten

Lyu, Horgan und London beschreiben in [Ly93] das Glassboxtest-Werkzeug ATAC (Automatic Test Analysis for C). ATAC erhebt Datenflussüberdeckung und enthält Funktionen, um die Ausführung einzelner Testfälle zu vergleichen. Allerdings kann Testfallbeginn und -ende nicht während eines Programmlaufs bestimmt werden, sondern jeder Programmlauf – von Programmstart bis Programmende – entspricht der Ausführung für einen Testfall. Für den Test einzelner kleiner Funktionen ist diese Einschränkung akzeptabel, beim Test großer Systeme aber unpraktisch.

Eine selektive Verkleinerung einer Testsuite schlagen Jeffrey und Gupta [Je05] vor. Sie beschreiben ein Verfahren, bei dem die Testsuite zwar verkleinert, der Verlust an Testgüte aber möglichst gering gehalten werden soll. Sie führen ihre Untersuchungen an relativ kleinen Programmen durch und verwenden dabei das Werkzeug ATAC zur Datenerhebung. James und Harrold [Ja03] untersuchen die Testsuite-Verkleinerung auf der Basis der Bedingungsüberdeckung der „Modified Condition/Decision Coverage“. Sie betrachten speziell sicherheitskritische Software mit einer sehr hohen Zahl an Testfällen.

Größere Systeme untersuchen Piwowarski et al. [Pi94] und Kim [Ki03]. Sie berichten von praktischen Erfahrungen beim Einsatz des Glassboxtests bei großen Systemen, ohne dabei aber auf Testsuite-Verbesserung einzugehen. Sie stellen fest, dass die Tester selbst nach einem (soliden) Blackboxtest in vielen Fällen eine zu hohe Selbsteinschätzung der Programmüberdeckung haben.

Der Glassboxtest birgt auch Risiken. Zu dieser Erkenntnis kommen Lawrance et al. [La05] in ihrem Experiment zur Wirkung von Visualisierung im Glassboxtest. Sie untersuchen die Wirkung von grafischer, durch Markierung im Programmcode angezeigter Überdeckung. Sie stellen fest, dass Entwickler, denen die Überdeckung im Programmcode grafisch angezeigt wird, sich eher mit weniger Testfällen zufrieden geben als solche Entwickler, die entweder über keinerlei Glassboxtest-Werkzeug verfügen oder denen die grafische Anzeige nicht zur Verfügung steht. Das Glassboxtest-Werkzeug erzeugt hier offensichtlich eine „trügerische Sicherheit“ und signalisiert durch entsprechende Einfärbung, dass ein Programmcode „ausreichend“ getestet ist. Das Experiment zeigt, dass eine Teststrategie „nur-Glassboxtest“, also Glassboxtest ohne einen vorab durchgeführten Blackboxtest, i. A. nicht als sinnvolles Vorgehen angesehen werden kann. Auf eine „nur-Glassboxtest“-Strategie wird im folgenden auch nicht weiter eingegangen.

1.3 Werkzeuglandschaft

Yang et al. [Ya06] untersuchen und vergleichen 17 Glassboxtest-Werkzeuge für Java und C sowie C++ und geben einen guten Überblick über die aktuelle Marktsituation. Die Werkzeuge unterstützen weitestgehend die gleichen Überdeckungsmaße wie Anweisungs-, Zweig- und Methodenüberdeckung. Datenflussüberdeckung wird von keinem der Werkzeuge unterstützt, was die Autoren auf die Schwierigkeiten bei der Interpretation der Ergebnisse zurückführen. Bei der Anweisungsüberdeckung basieren die Werkzeuge teils auf Programmblöcken, auf Anweisungen im Sinne der Grammatik oder auf Programmzeilen. Einzelne Werkzeuge unterstützen auch Bedingungsüberdeckungen. Unterschiede zwischen den Werkzeugen bestehen im Wesentlichen in der Integrationsmöglichkeit in einen bestehenden Build-Prozess und bei den Reportfunktionen. Insgesamt müsste man aus der Zahl der Anbieter von Glassboxtest-Werkzeugen auf eine lebhafte Nachfrage schließen. In der Untersuchung von Müller et al. [Mü98] zeigt sich allerdings, dass kontrollflussorientierte Testverfahren keine hohe Verbreitung haben. Selbst Anweisungsüberdeckung praktizieren nur 12 % der befragten Firmen (wobei in dieser Untersuchung auch 12 % der Firmen Pfadüberdeckung einsetzen, was zeigt, dass besonders beim Test eine erhebliche Begriffsverwirrung bei den Praktikern herrscht).

1.4 Nutzen des Glassboxtests

Der Nutzen des Glassboxtests wird in der Literatur zusammenfassend wie folgt beschrieben:

- **Codeüberdeckungsmaße als Alternativ-Metrik zur Testgüte:** Der Glassboxtest liefert für die Durchführung einer Testsuite die ausgeführten Programmbereiche (z. B. Anweisungen oder Programmzweige), die ins Verhältnis zum gesamten Programm gesetzt werden. So ergibt sich für eine Testsuite ein objektives Vollständigkeitskriterium, das beispielsweise als Testendekriterium verwendet werden kann. Im Wesentlichen besteht Einvernehmen darüber, dass Codeüberdeckung und Testgüte korrelieren, wobei eine hohe Codeüberdeckung ein nötiges, aber kein hinreichendes Merkmal für eine hohe Testgüte darstellt [Lu07][Sp05].
- **Testsuite-Erweiterung:** Piwowarski et al. berichten in [Pi93], dass Tester beim Systemtest die Qualität einer Testsuite oftmals überschätzen und durch den Glassboxtest von der Menge an ungetestetem Programmcode überrascht werden. Der Glassboxtest zeigt diesen ungetesteten Programmcode. Allerdings ist es gerade bei Systemen mit viel Programmcode und großer Testsuite unter Verwendung heute verbreiteter Glassboxtest-Werkzeuge für einen Tester ausgesprochen schwierig, auf konkrete fehlenden Testfälle zu schließen.
- **Testsuite-Reduktion:** Das Ziel ist die Kostensenkung beim Test, speziell beim Regressionstest, durch eine Verkleinerung der Testsuite, ohne dabei aber (wesentlich) Testgüte einzubüßen. Die Grundüberlegung basiert darauf, dass Testfällen mit gleicher oder sehr ähnlicher Überdeckung eine geringere Chance auf das Auffinden von Fehlern eingeräumt wird als Testfällen mit deutlich unterschiedlicher Überdeckung. Testfälle, die auf diese Weise als redundant erkannt werden, können entweder ganz aus der Testsuite entnommen oder in der Priorität weiter nach hinten versetzt werden. Damit kommen sie abhängig von der angestrebten Testgüte zum Einsatz [Je05] [Jo03].
- **Grundlage für selektiven Regressionstest:** Zur Kostenreduktion im Regressionstest sollen anstelle der „rerun-all"-Strategie nur einzelne, ausgewählte Testfälle ausgeführt werden. Die Grundüberlegung ist, dass nach einer (kleinen und abgegrenzten) Programmänderung nur die Testfälle ausgeführt werden, die vor der Programmänderung die betreffenden Programmstellen auch ausgeführt hatten [Sn04].

Bei allen genanten Einsatzgebieten des Glassboxtests ist es erforderlich oder sinnvoll die Programmausführung testfallselektiv zu erheben. D.h. die vom Glassboxtest-Werkzeug gespeicherte Programmausführung wird in einzelne Testfälle, jeweils von Testfallbeginn bis –ende, untergliedert.

2 CodeCover

Das Werkzeug CodeCover wurde im Rahmen eines Studienprojekts am Institut für Softwaretechnologie der Universität Stuttgart entwickelt. CodeCover bietet zunächst die gemäß Yang et al. für Glassboxtest-Werkzeuge üblichen Funktionen: Integration in eine Build-Umgebung, Code-Instrumentierung, Laufzeit-Protokollierung und Auswertung. CodeCover unterstützt derzeit die Sprachen Java und COBOL und besitzt eine Schnittstelle, die eine Erweiterung um weitere Programmiersprachen ermöglicht. Im folgenden werden die Funktionen beschrieben, die direkt von der testfallselektiven Ausführungserhebung betroffen sind.

2.1 Testfallselektiver Glassboxtest

Ähnlich wie beim Glassboxtest-Werkzeug ATAC kennt CodeCover die einzelnen Testfälle der Testsuite und kann einzeln für jeden Testfall die Programmcodeüberdeckung angeben. Bei CodeCover wird der Testfall aber nicht durch Programmstart und –ende festgelegt, sondern der Tester gibt explizit Beginn und Ende eines Testfalls an. Die Anwendung muss hierzu nicht beendet werden. Die einzelnen Testfälle müssen auch nicht unmittelbar aufeinander folgen. Es ist denkbar, dass ein Testfall endet und der Tester eine Reihe an Funktionen des zu testenden Systems verwendet (z.B. Rücksprung auf einen Ausgangspunkt), bevor der dann folgende Testfall beginnt. Für die Praxistauglichkeit hat es sich als wichtig erwiesen, dass der Tester sehr einfach Beginn und Ende eines Testfalls bestimmen kann. Beim Test von Java-Anwendungen verwendet CodeCover die Java Management Extension (JMX) zur Übermittlung von Testbeginn und –ende. Dies hat große praktische Vorteile, weil die Anwendungen unabhängig von ihrer Architektur gleich behandelt werden können und auch eine Steuerung über das Netzwerk kein Problem darstellt. Über definierte JMX-Anweisungen werden dem zu testenden System Beginn und Ende eines Testfalls übermittelt. Diese JMX-Anweisungen können über eine beliebige JMX-Konsole oder auch über einen einfachen Eingabedialog, der Teil des eclipse-Plugin von CodeCover ist, ausgelöst werden. Die im zu testenden System erforderliche JMX-Server-Funktion wird bei der Instrumentierung automatisch eingerichtet.

Abbildung 1: Eingabe von Testfallbeginn und -ende

Bei COBOL-Anwendungen wurden bestimmte Kommentarstile festgelegt, die vom CodeCover-Instrumentierer in Prozeduraufrufe transformiert werden. Für Batch-Systeme ist ein Benutzereingriff ohnehin schlecht möglich.

2.2 Redundanzanalyse

Ausgangspunkt der Redundanzanalyse bildet die testfallselektive Erhebung der ausgeführten Programmeinheiten. Aus dem paarweisen Vergleich können solche Testfallpaare angezeigt werden, die eine gleiche oder nahezu gleiche Ausführung haben. CodeCover visualisiert für ausgewählte Testfälle die Resultate dieses paarweisen Vergleichs in einer Korrelationsmatrix. Der Grauton gibt die Übereinstimmung zweier Testfälle an.

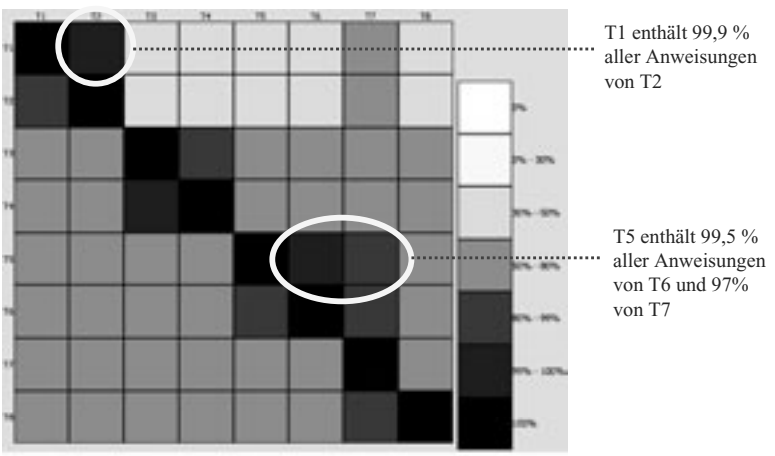


Abbildung 2: Beispiel einer Korrelationsmatrix

Im Beispiel von Abbildung 2 wurde der Testfall T1 wiederholt und als Testfall T2 erneut ausgeführt. Die Übereinstimmung von T1 und T2 ist damit nahe 100 %. Tatsächlich wurden bei T1 und T2 etwa 3000 Anweisungen gleich und 5 verschieden ausgeführt. Der auf den ersten Blick überraschende Unterschied liegt an unterschiedlich ausgeführten Singletons. Im Unterschied zu den Arbeiten von Jones und Herrold [J003] sowie Jeffrey und Gupta [Je05] sollen die Auswertungen lediglich Hinweise auf Redundanzen liefern und damit eine Inspektion der betroffenen Testfälle empfehlen. Eine automatische Testsuite-Reduktion ist nicht vorgesehen.

Da beim Einsatz mit großen Testsuites die Visualisierung als Matrix unübersichtlich wird, können die Datenwerte auch als csv-Datei exportiert und außerhalb weiter ausgewertet werden.

2.3 Testfallselektive Visualisierung der Programmausführung

Glassboxtest-Werkzeuge visualisieren ausgeführte Anweisungen in der Regel durch entsprechende Markierungen im Programmcode. Dort, wo Anweisungen ausgeführt wurden, wird der Programmcode grafisch hervorgehoben. Die Auswertung und Visualisierung bezieht sich auf die gesamte Testsuite. In CodeCover können wahlweise einzelne Testfälle in die Auswertung einbezogen oder davon ausgeschlossen werden. Im Beispiel von Abbildung 3 sind die Testfälle der Testsuiten 1 und 3 einbezogen, die Testfälle der Testsuite 2 nicht. Die Visualisierung wie auch die Überdeckungsberechnung erfolgt dann nur für die ausgewählten Testfälle. Umgekehrt werden zu einer im Programmcode selektierten Anweisung die Testfälle angezeigt, die zur jeweiligen Ausführung geführt haben. Im Beispiel von Abbildung 3 wird zur selektierten Anweisung der Testfall T6 angegeben.

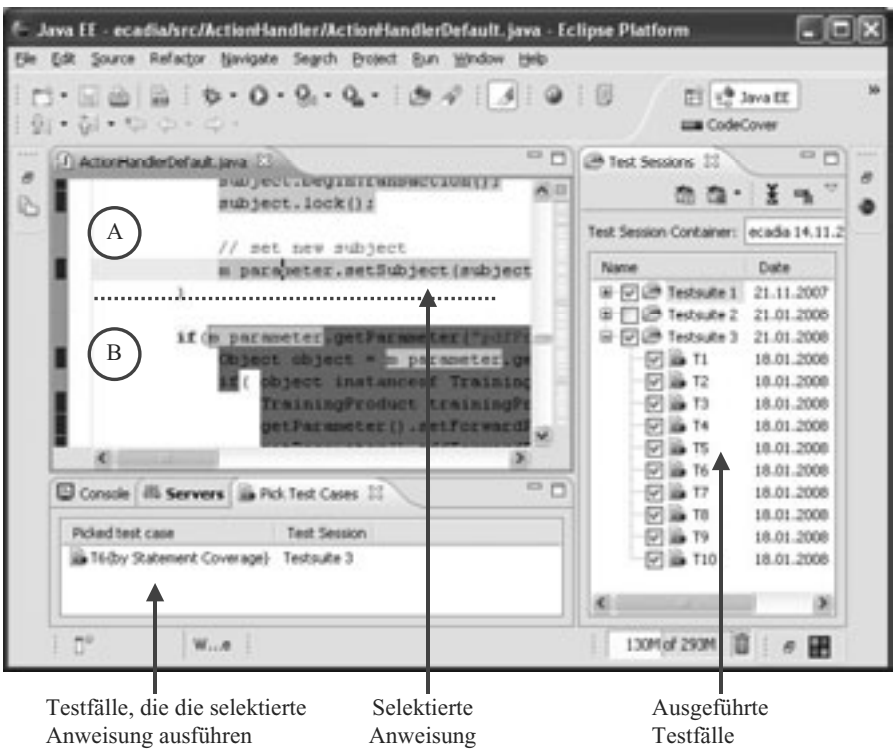


Abbildung 3: CodeCover-Benutzungsschnittstelle

Diese Testfallanzeige kann dazu dienen, fehlende Testfälle leichter zu finden. So besteht die Möglichkeit, einen fehlenden Testfall von einem bestehenden Testfall aus zu beschreiben. Im Beispiel wäre der bestehende Testfall T6, der den Programmcode im Bereich A ausführt. Der Programmcode im Bereich B wird von keinem Testfall ausgeführt. Es ist nun praktisch, den Testfall T6 als Ausgangspunkt zu verwenden und für den fehlenden Testfall das Prädikat der abweisenden Verzweigung einzubeziehen.

Die Testfallanzeige lässt sich auch als Grundlage für den selektiven Regressionstest verwenden. Im Beispiel würden Änderungen im Programmbereich A eine Testdurchführung nur von Testfall T6 unmittelbar erfordern. Bei Programmcode, der nur wenigen Änderungen unterliegt, ist auf diese Weise eine deutliche Kostenersparnis beim Regressionstest möglich.

3 Zusammenfassung

Der Glassboxtest kann einen wichtigen Beitrag zum Programmtest leisten. Das Werkzeug CodeCover stellt hierzu durch eine testfallselektive Erhebung der Programmausführung eine Reihe an Funktionen bereit. CodeCover steht unter EPL-Lizenz und ist unter www.codecover.org frei verfügbar.

Literatur

[Je05] D. Jeffrey and N. Gupta, “Test Suite Reduction with Selective Redundancy,” Proc. 21st IEEE Int'l Conf. Software Maintenance, pp.549-558, 2005.

[Jo03] Jones, J., Harrold, M.: „Test-Suite Reduction an Prioritization for Modified Condition/decistion Coverage”, IEEE Transactions on Software Engineering, Vol. 29, No. 3, March 2003

[Ki03] Kim, Y. W. “Efficient use of code coverage in large-scale software development”, Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, October 06 - 09, 2003). IBM Press, 145-155.

[La05] Lawrance, J., Clarke, S., Burnett, M., Rothermel, G., “How Well Do Professional Developers Test with Code Coverage Visualizations? An Empirical Study”, Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2005

[Lu07] Ludewig, J., Lichter, H.: “Software Engineering”. dpunkt Verlag 2007

[Ly93] Lyu, M.R., J.R. Horgan, S. London, "A Coverage Analysis Tool for the Effectiveness of Software Testing," Proceedings of ISSRE’93, Denver, CO, pp. 25-34, November 1993.

[Mü98] Uwe Müller, Thomas Wiegmann: “State of the practice“ der Prüf- und Testprozesse in der Softwareentwicklung — Ergebnisse einer empirischen Untersuchung bei Softwareunternehmen in Deutschland“ Technischer Bericht, Universität Köln, März 1998

[My79] Myers, G. J.: “Art of Software Testing”, John Wiley & Sons, Inc., New York, 1979

[Pi93] Piwowarski, P.; Ohba, M.; Caruso, J.: “Coverage Measurement Experience During Function Test”, Proceedings of the 15th International Conference on Software Engineering, ICSE'93, 1993

[Sn04] Sneed, H.M.: “Reverse engineering of test cases for selective regression testing”, Proceedings of CSMR 2004, pp 69-74, 24-26 March.

[Sp05] Spillner, A., Linz, T.: “Basiswissen Softwaretest”. 3. Auflage, dpunkt Verlag 2005

[Ya06] Yang, Q., Li, J. J., and Weiss, D. 2006. A survey of coverage based testing tools. In Proceedings of the 2006 International Workshop on Automation of Software Test (Shanghai, China, May 23 - 23, 2006). AST '06. ACM, New York, NY, 99-103.

Automatic Generation of Optimized Integration Test Data by Genetic Algorithms

Florin Pinte, Francesca Saglietti, Norbert Oster

Lehrstuhl für Software Engineering
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen
pinte@informatik.uni-erlangen.de
saglietti@informatik.uni-erlangen.de
oster@informatik.uni-erlangen.de

Abstract: The importance of software in nearly all of today's engineering disciplines demands for development and validation techniques ensuring high dependability of complex software systems. Component-based software development as the ultimate approach for dealing with complexity shifts the focus of verification from unit to integration tests addressing the correctness of component interactions. As the current state-of-the-art does not include a systematic and tool-supported approach to interface testing, this paper presents a procedure for the automatic generation of integration test data based on genetic algorithms.

1 Introduction

The clarity and economic attractiveness offered by systems consisting of reusable components frequently leads to the underestimation of a major source of failures, namely of those caused by the incorrect interaction of correct components [SJ04]. Integration testing is meant to expose defects at the interfaces and in the interactions between integrated components [IST07]. Due to the lack of tool support this testing phase is still carried out in a very unsystematic and cursory way. In particular, techniques to generate test data and to evaluate their interface coverage are missing in the industrial environment.

When compared to state-of-the-art integration testing techniques [MP05], which are mainly limited to the consideration of operational call sequences in UML diagrams, the novelty of our approach consists in including further details concerning component interaction, like message-based and state-based information, as well as in applying genetic algorithms for the purpose of achieving a high interaction coverage by a low number of test cases.

In view of the manual effort needed to verify the behaviour observed during testing it is crucial to avoid redundant testing activities. For this reason our research work addresses the problem of systematizing rigorously the integration testing process by providing

1. a hierarchy of interface coverage criteria,

2. for each interface coverage criterion automatic techniques resp. tools capable of generating input data aimed at

    2.1 maximizing the interaction coverage and

    2.2 minimizing the number of test cases.

In order to focus on component interaction, integration testing cannot be done in a purely black-box fashion; on the other hand, in general commercial off-the-shelf components have already been extensively tested, such that a white-box approach would imply unnecessary re-testing. For these reasons we decided to base our approach on a grey-box level as provided by UML diagrams.

Supported by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology, our ongoing project UnITeD (Unterstützung inkrementeller Testdaten) aims at fully automating the generation of optimized integration test cases on the basis of UML model descriptions of components enriched by interaction information.

Our methodology grounds on a similar approach [Os07] developed at our department, which successfully applies genetic algorithms to the automatic generation of optimized test data for white-box testing at code level. We also developed a further derivative of this technique which addresses grey-box testing of components at model level, thus providing automatic support for a number of model-based testing strategies [OSS07]. An appropriate hierarchy of interface coverage criteria was introduced in [SOP07] together with a worst case estimate of the number of test cases required to achieve a given coverage.

This article is structured as follows: section 2 introduces the grey-box model with respect to which we represent the behaviour and the interaction of components. Successively, genetic algorithms supporting the automatic test data generation are presented in section 3, together with a model simulator required to evaluate the interaction coverage achieved. Finally, section 4 describes experimental results obtained by applying this technique.

2 Modelling the Behaviour and the Interaction of Components

We chose to represent the behaviour and the interaction of components by UML state machines enriched by messages sent between the components as possible effects of state transitions. Whenever a state transition tA in an invoking component A results in calling a method of a component B leading to a corresponding transition tB, we call the pair (tA, tB) a mapping. Fig. 1 illustrates for example the mapping m1 = (tA6, tB4) resulting from the call of opB2 as effect of transition tA6 triggering transition tB4. A further mapping m2 = (tA6, tB3) arises from the call of opB2 as effect of transition tA6 triggering transition tB3.

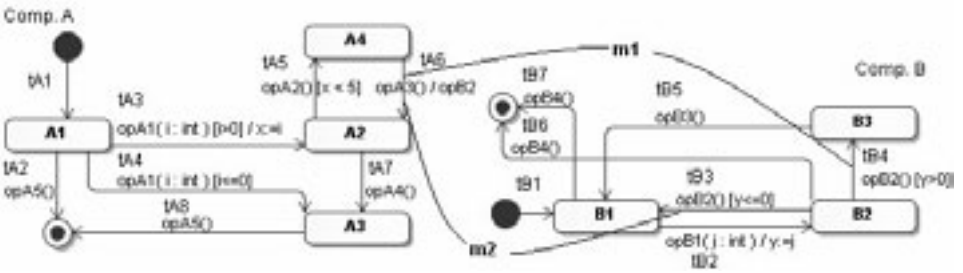


Fig. 1: State machines describing the behaviour of two communicating components A and B with mappings m1 = (tA6, tB4) and m2 = (tA6, tB3)

3 The Genetic Algorithm and the Model Simulator

This section addresses the question of applying genetic algorithms for the purpose of multi-objective optimization. Genetic algorithms work with populations of individuals, each individual being a possible solution to the optimization task.

In the following a test case is defined as a sequence of method calls together with corresponding data, while a test suite is defined as a set of test cases. In our approach each individual of a population corresponds to a test suite, each gene of an individual corresponding to a test case. For the purpose of evaluating the fitness of an individual the following properties are considered:

- the interaction coverage achieved,
- the size of the test suite, i.e. the number of test cases it contains.

Further properties which may be taken into account in addition to the above are:

- the maximum length of test cases in terms of the number of messages contained,
- the distance between the state reached after a test run and the final state of the machine (when test cases triggering complete paths from the initial node to the final node of the state machines are required).

Having determined the encoding and the fitness function a multi-objective genetic algorithm can be applied, starting with the generation of an initial population.

Typical genetic operators (e.g. mutation, crossover) are applied to each generation, thus resulting in the emergence of a new population. This process is repeated until a stopping rule is met. Examples of such stopping rules are the achievement of a given fitness or the generation of a given number of populations.

3.1 Coverage Criteria

So far, systematic integration testing criteria were essentially limited to the consideration of operation call sequences in UML diagrams ([WCO03] and [AO00]). In our opinion, thorough integration testing should not neglect further details on component interaction like message-based information, which takes into account the variety of data conditions underlying the calls, as well as of state-based information, which includes details on internal component transitions as done by the state-based coverage criteria shown in Fig. 2 and introduced in detail in [SOP07]. The most demanding among these criteria is denoted by "invoking/invoked-transitions" and requires to cover all mappings, i.e. all pairs of corresponding transitions, in particular taking into account all possible combinations of pre- and post-states of invoking and invoked components as well as all triggers and non-trivial effects of the invoking component.

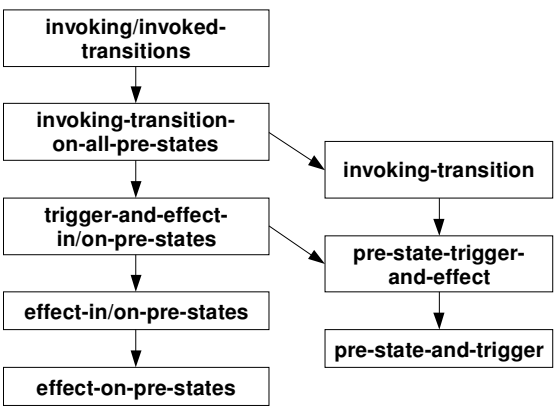


Fig. 2: Hierarchy of state-based interface coverage criteria

3.2 Initial Population

The automatic generation of test data by genetic algorithms starts with an initial population with each individual containing several test cases, i.e. sequences of messages to communicating components. Considering components A and B as shown in Fig. 1, in the following we will describe how such an initial message sequence can be built for the special case of two interacting components. For every component a possible message sequence is randomly generated, for example the sequence (opA1(4), opA2(), opA3()) for component A resp. (opB1(7), opB4()) for component B. Interleaving these two sequences results in a test case like (opA1(4), opB1(7), opA2(), opB4(), opA3()).

3.3 Coverage Measurement by Model Simulation

For the purpose of computing the fitness of a test suite, the coverage achieved by all test cases contained in an individual needs to be determined. This is done by means of a model simulator developed within our project. For every interacting component this model simulator is instantiated, processes the corresponding messages contained in the test case to be evaluated and returns the list of covered transitions.

In order to do so, the simulation of the mere component functionality was extended such as to include also component interactions by propagating internal messages (resulting as transition effects) to the components they are destined for.

For the simple example in Fig. 1 the coverage of the test case (opA1(4), opB1(1), opA2(), opA3(), opB4()) w.r.t. to the invoking-invoked criterion [SOP07] is evaluated by model simulation as described in the following. According to the invoking-invoked criterion all mappings between the two components must be covered, thus the target of the generation is to cover the mappings m1 = (tA6, tB4) and m2 = (tA6, tB3). The process of determining the coverage of this test case is explained step by step in the following.

Step 1: Instantiate and initialize the simulators: component A is in state A1 and component B is in state B1, such that transitions tA1 and tB1 are traversed immediately.

Step 2: Process the first message opA1(4) of the test case: this message is sent to the simulator of component A. The guard of transition tA3 is evaluated to true, the internal variable x is assigned the value 4 and transition tA3 is traversed, resulting in state A2.

Step 3: Process the second message opB1(1) of the test case: this message is sent to the simulator of component B. The internal variable y is assigned the value 1 and transition tB2 is traversed, resulting in state B2.

Step 4: Process the third message opA2() of the test case: this message is sent to the simulator of component A. The guard of the transition tA5 is evaluated to true (because $x = 4 < 5$) and transition tA5 is traversed, resulting in state A4.

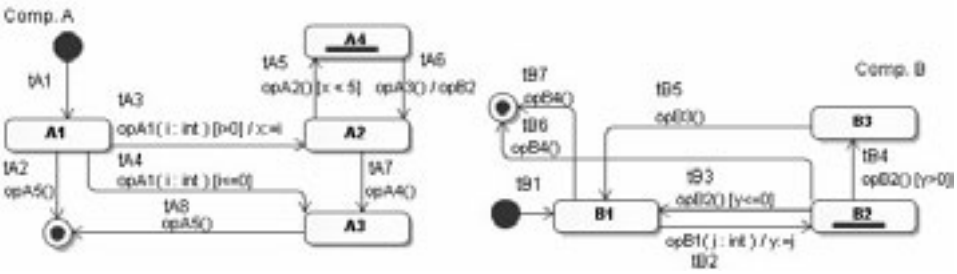


Fig. 3: The internal states A4 and B2 of the components as reached after the fourth step

Step 5: Process the fourth message opA3() of the test case: this message is sent to the simulator of component A. Transition tA6 is traversed, resulting in state A2; its effect consists of an internal message opB2 to component B, which is accordingly sent to the simulator of B. The guard of transition tB4 is evaluated to true (because y = 1 > 0) and transition tB4 is traversed, resulting in state B3.

Step 6: Process the fifth message opB4() of the test case: this message is sent to the simulator of component B. Because there is no transition leaving the actual state B3 with trigger opB4(), this message does not change the state of component B.

The transitions traversed by the test case considered are: tA1, tB1, tA3, tB2, tA5, tA6, tB4. Since the target is to cover both mappings m1 = (tA6, tB4) and m2 = (tA6, tB3), the test case has achieved a coverage of 50% - by covering only mapping m1. If the above test case is modified by swapping messages opA3() and opB4(), the coverage of the modified test case is 0. Full coverage may be achieved by an additional test case hitting also mapping m2, like (opA1(4), opA2(), opB1(-3), opA3()). But since our approach aims at minimizing the size of the test suite a better solution would consist of adapting the first test case such as to cover both mappings by one message sequence, e.g. (opA1(4), opB1(1), opA2(), opA3(), opB3(), opB1(-1), opA2(), opA3()).

On the whole we can note that the main difficulty in generating adequate test suites essentially lies in determining the order of messages between interacting objects. In fact, the message sequence must first bring the components into appropriate states enabling the subsequent coverage of the interaction entities envisaged. Obviously, our technique is also applicable to systems of more realistic complexity, in particular those with parameterized operational calls between components, as the one introduced in the following section.

4 Experimental Results

This section describes preliminary results gained by applying the technique described above. In cooperation with the industry a tool was developed which automatically generates test suites from EMF UML models. After having modelled the component-based software system under test the tester may choose among the following alternative interaction modes: interaction of two components, interaction of one component with the rest of the system and interaction of all components within the system.

The application of our technique was exemplified for the software-based elevator controller shown in Fig. 4 and consisting of 6 components. Some of the results obtained are presented in Table 1. The number of generations #G and the size #T of the test suite required to achieve a given coverage are of special interest.

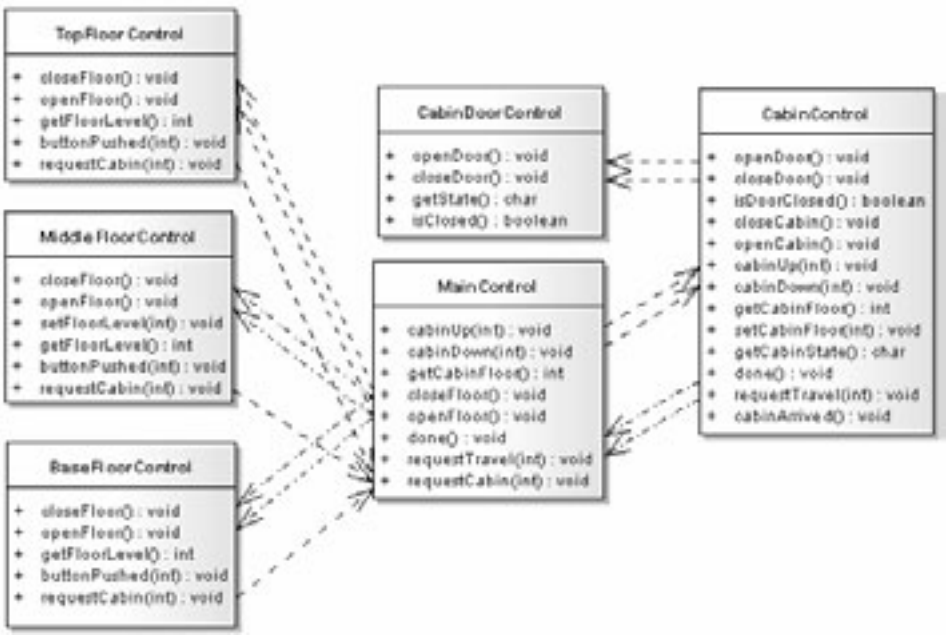


Fig. 4: The software components and their interactions for the elevator system

Interaction between:	#Est	#Cov	#G	#T
CabinControl and MainControl	16	100%	40	7
CabinControl and all other components	20	100%	50	9
All components	32	81% (26 mappings)	50	9

Table 1: Results related to the generation of test suites
for the *invoking/invoked-transitions* interface coverage criterion

#Est:	worst-case estimation of the number of test cases required
#Cov:	coverage achieved
#G:	number of generations required
#T:	number of test cases required

The number of test cases required to fulfil the criterion is less than half of the worst-case estimates determined according to [SOP07]. Even if the absolute decrease in test cases may not look too impressive at first sight, it allows a substantial effort reduction in view of the laborious manual verification required by each of the test runs involved.

5 Conclusions

This research paper introduced an innovative approach for the automatic generation and optimization of integration test suites fulfilling a variety of state-based coverage criteria as introduced in [SOP07]. The efficiency and economic attractiveness of this methodology was clearly demonstrated by means of convincing and promising preliminary results. The technique developed allows for filling a major gap in today's industrial software testing environment by offering a practical, fully automated support during the integration of component-based software systems.

Future work will include the extension of the present tool such as to include further UML behavioural diagrams like Activity and Sequence Diagrams. This will allow the generation of optimized test suites also for the mapping- and message-based testing criteria proposed in [SOP07].

References

[AO00] Abdurazik, A.; Offutt, J.: Using UML Collaboration Diagrams for Static Checking and Test Generation. Proc. 3rd Int. Conf. on The Unified Modeling Language, 2000

[IST07] Erik van Veenendaal (Ed.): Standard glossary of terms used in Software Testing, Version 1.3, ISTQB International Software Testing Qualification Board, 2007

[MP05] McQuillan, J. A.; Power, J. F.: A Survey of UML-Based Coverage Criteria for Software Testing. Technical Report NUIM-CS-TR-2005-08, Department of Computer Science, NUI Maynooth, Co. Kildare, Ireland, 2005

[Os07] Oster, N.: Automatische Generierung optimaler struktureller Testdaten für objekt-orientierte Software mittels multi-objektiver Metaheuristiken. Dissertation, in Arbeitsberichte des Instituts für Informatik, Vol. 40, Nr. 2, University Erlangen-Nuremberg, 2007

[OSS07] Oster, N.; Schieber, C.; Saglietti, F.; Pinte, F.: Automatische, modellbasierte Testdatengenerierung durch Einsatz evolutionärer Verfahren. In R. Koschke, O. Herzog, K.-H. Rödiger, M. Ronthaler (Eds.): Informatik 2007 - Informatik trifft Logistik, Vol. 110 of Lecture Notes in Informatics, Gesellschaft für Informatik, 2007

[SJ04] Saglietti, F.; Jung, M: Classification, Analysis and Detection of Interface Inconsistencies in Safety-Relevant Component-based Systems, in C. Spitzer, U. Schmocker and V. N. Dang (Eds.): Probabilistic Safety Assessment and Management, Springer-Verlag, 2004

[SOP07] Saglietti, F.; Oster, N.; Pinte, F.: Interface Coverage Criteria Supporting Model-Based Integration Testing. In M. Platzner, K.-E. Großpietsch, C. Hochberger, A. Koch (Eds.): ARCS '07 Workshop Proceedings, VDE Verlag, 2007

[WCO03] Wu, Y.; Chen, H.; Offutt, J.: UML-based Integration Testing for Component-based Software, Proc. Second International Conference on COTS-based Software Systems, Lecture Notes in Computer Science, Volume 2580, Springer-Verlag, 2003

Automatic Test Case Generation from UML Models and OCL Expressions

Stephan Weißleder[1], Dehla Sokenou[2]

[1]Humboldt-Universität zu Berlin, Institut für Informatik
Rudower Chaussee 25, 12489 Berlin, Germany
weissled@informatik.hu-berlin.de

[2]GEBIT Solutions
Koenigsallee 75 b, 14193 Berlin, Germany
dehla.sokenou@gebit.de

Abstract: In this paper, we discuss one approach of automated test case generation from UML models and OCL expressions. We show how to use UML and OCL to support several coverage criteria. We introduce our current prototype implementation, compare it to commercial tools, and sketch shortcomings and further development.

1 Introduction

Testing is the primary means to detect faults in a system under test (SUT), e.g. a software product. Modelling languages like the Unified Modeling Language (UML) [Gro05] are means to design tests. There is established tool support for model-based test generation based on UML. For instance, Rhapsody ATG [Tel] supports test generation for the target language C++. Leirios Test Designer [Lei] supports a subset of the Object Constraint Language (OCL). We argue that the tool support of model-based testing could be improved. The quality of test suites is often measured with coverage criteria (although there is no proof of their effectiveness). We argue that the test generation support for boundary-based coverage criteria [KLPU04] could also be improved. For instance, AETG [Tec] depends on user-defined partition boundaries instead of deriving them from the model.

Our approach is to automatically generate test cases from UML state machines, UML class diagrams, and OCL expressions to satisfy boundary-based coverage criteria. We correlate OCL pre-/postconditions of operations and guard conditions of state machines to derive test data input partitions automatically [WS07a]. This allows to generate test suites that satisfy, e.g., a combination of *Multi-Dimensional* [KLPU04] (each variable takes each boundary value of each partition at least once) and *MC/DC* [UL06] (every condition in a decision in the program has taken on each possible outcome at least once, and each condition has been shown to affect that decision outcome independently) (see [WS08]). To our knowledge, ParTeG is the first tool that generates equivalence classes for test input parameters from UML and OCL.

2 Test Case Generation from UML and OCL

UML is a widely accepted means to express models used in model-based test generation. Since OCL constraints can express conditions that can not be phrased in UML, OCL is often used together with UML models for test case generation. We shortly sketch previous approaches to use UML and OCL: One approach is to use OCL as an additional test oracle (compare e.g. [Sok06]). In this case, OCL expressions are simply transformed to the language of the SUT. Another approach is to relate pre-/postconditions of one operation and derive behavioural information from static models [BBH02]. A more advanced approach is presented by Leirios [Lei], that combines different OCL conditions by the control flow information of a state machine. Equations in postconditions are interpreted as assignments in order to allow a symbolic execution of the model. However, this approach supports only a subset of OCL. For instance, our approach additionally considers inequations and a wider set of logical expressions (e.g. "or", "not") in postconditions.

The basic idea of relating OCL conditions is that postconditions attached to a transition affect the satisfaction of subsequent guard conditions. Consequently, it is possible to identify dependencies between them in the control-flow of state machines. A classification of the elements comprised in postconditions allows to define the attributes of a postcondition, whose values are changing [WS07a]. This allows to correlate certain elements of guard conditions and preconditions to corresponding elements of postconditions. As a result of this approach, guard expressions can be transformed into conditions for input parameters. These resulting conditions are interpreted as partition boundaries.

2.1 Prototype Implementation

We implemented our approach in an Eclipse-based tool called ParTeG [Wei]. This tool checks all paths from the initial state of the UML state machine until a loop is reached. Every time the tool encounters, e.g., a guard condition, it searches the path backwards to find a postcondition that affects the outcome of this guard condition. The postconditions describe, e.g., the effect of input parameters on attribute values. This allows to create a connection between the values of test input parameters and the control flow within the state machine. The constraints on test input parameters are interpreted as equivalence classes and the concrete values are selected close to the corresponding boundaries (see Fig. 1).

2.2 Coverage Criteria

Coverage criteria are the primary means to measure the quality of a test suite. Nevertheless, there is no proof for a relationship between satisfied coverage criteria and the number of identified faults. This is fortified by the success of random testing compared to model-based testing, which brings up the question for cost efficiency of model-based testing [Pre06]. Consequently, some approaches combine random testing and the mere

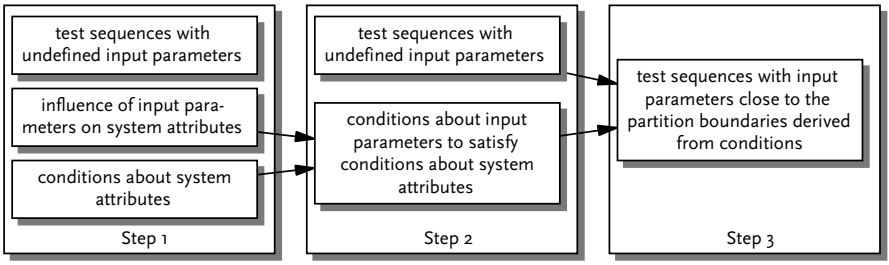


Figure 1: Basic idea of the test generation process.

observation of coverage criteria. For instance, Rhapsody ATG [Tel] follows the approach of generating test cases and measuring the satisfied coverage. Although our current approach is not random, we also observe the percentage of satisfied coverage criteria instead of enforcing it. However, we plan to change this strategy to enforce the satisfaction of each part of a coverage criterion (test goal).

In addition to satisfying just one coverage criterion, the satisfaction of a combination of different coverage criteria seems to be promising [WS08]. Since ParTeG is already focussed on boundary coverage, it allows to combine, e.g., boundary-based coverage criteria and control-flow-based coverage criteria.

2.3 Comparison to Commercial Tools

We compared our prototype implementation to already existing commercial tools. In this context, we used mutation analysis with a subset of the "sufficient mutation operators" defined in [OLR+96]. Since our approach is focused on the deduction of test input data partitions, the test suites generated by ParTeG reached a higher number of killed mutants than Rhapsody ATG and Leirios Test Designer [WS07b].

Most tools for automatic test generation are fixed on satisfying certain coverage criteria. Since each coverage criterion has its eligibility for a certain application field and a certain required safety level, it would desirable to let the user choose the coverage criterion to satisfy. ParTeG allows the user to choose from a set of coverage criteria, which makes it applicable for a broader set of possible application environments.

2.4 Shortcomings and Further Development

The results obtained from the comparisons to commercial tools only hold for examples with few loops and a few conditions based on the repetition of executing these loops. This is caused by the internal structure used for deriving test cases: a tree that describes possible test execution paths. In order to avoid infinite tree size, the length of the paths has to be

limited. This results in problems for covering conditions whose satisfaction depends on certain executed loops in a path (e.g., a cash card that is only retained after entering a false PIN three times). Since this leads to suboptimal coverage, we aim at a different approach: Instead of creating a tree, we stick to the graph structure of the state machine. Based on the coverage criteria to satisfy, we define certain test goals like a set of transitions for transition coverage or a set of assignments of a condition's attribut values for MC/DC. Following that, we iterate through all test goals and apply our approach of deducing test input values to a test case generation algorithm like presented in [OA99] or [Sok06].

Acknowledgements. This work was supported by grants from the DFG (German Research Foundation, research training group METRIK).

References

[BBH02] M. Benattou, J.-M. Bruel, and N. Hameurlain. Generating Test Data from OCL Specification. In *WITUML'02*, 2002.

[Gro05] Object Management Group. Unified Modeling Language (UML), version 2.0, 2005.

[KLPU04] N. Kosmatov, B. Legeard, F. Peureux, and M. Utting. Boundary Coverage Criteria for Test Generation from Formal Models. In *ISSRE '04*, pages 139–150. IEEE, 2004.

[Lei] Leirios. LTG/UML. http://www.leirios.com.

[OA99] J. Offutt and A. Abdurazik. Generating Tests from UML Specifications. In *UML'99*, pages 416–429, 1999.

[OLR+96] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, pages 99–118, 1996.

[Pre06] A. Pretschner. Zur Kosteneffektivität des modellbasierten Testens. In *MBEES'06: Modellbasierte Entwicklung eingebetteter Systeme*, pages 85–94, 2006.

[Sok06] D. Sokenou. Generating Test Sequences from UML Sequence Diagrams and State Diagrams. In *INFORMATIK 2006*, pages 236–240, 2006.

[Tec] Telcordia Technologies. AETG. http://aetgweb.argreenhouse.com.

[Tel] Telelogic. Rhapsody Automated Test Generation. http://www.telelogic.com.

[UL06] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., 2006.

[Wei] S. Weißleder. ParTeG (Partition Test Generator). http://parteg.sourceforge.net.

[WS07a] S. Weißleder and B.-H. Schlingloff. Automatic Test Generation from Coupled UML Models using Input Partitions. In *MoDeVVa*, 2007.

[WS07b] S. Weißleder and B.-H. Schlingloff. Deriving Input Partitions from UML Models for Automatic Test Generation. In *LNCS Volume on Models in Software Engineering*, 2007.

[WS08] S. Weißleder and B.-H. Schlingloff. Quality of Automatically Generated Test Cases based on OCL Expressions. In *ICST*, April 2008.

A Domain Specific Language for Uncovering Abstract Protocols and Testing Message Scenarios

Tim Reichert[1], Dominikus Herzberg[2]

[1]School of Computing, Engineering & Inf. Sciences, Northumbria University
Newcastle upon Tyne, NE2 1XE, United Kingdom
tim.reichert@unn.ac.uk

[2]Department of Software Engineering, Heilbronn University
74081 Heilbronn, Germany
herzberg@hs-heilbronn.de

Abstract: We present CFR, a language for the specification of protocols and communication scenarios and show how this language can be used for systematic testing of distributed systems. Our language is based on three basic concepts, namely channels, filters and rules. Using our current implementation, we can fully generate sophisticated analyzers from CFR specifications.

1 Introduction

Many of the errors in today's distributed systems occur in the context of complex communication scenarios involving multiple distributed components. Such errors are intricate and cannot be found by component testing alone. System testing complex scenarios is, however, a difficult task, mainly because it is hard to specify correct behavior at the right level of abstraction and in a way that can be used for automated testing.

Protocol analyzers can be used for monitoring and testing system communication on different layers of a protocol stack. However, the parts of the communication design that are not specified as part of a protocol are invisible during analysis. Instead, only the effect a design has on protocol layers can be traced. This has severe consequences, as test cases have to be specified at an inappropriate level of abstraction, which makes them overly complex and likely sources of errors. We provide an example of this in Section 3.

We developed a language called CFR (Channel Filter Rule) that allows comprehensive specification of communication behavior and the full generation of specialized protocol analyzers that can monitor and test this communication behavior. The main purpose of the CFR approach is to uncover and document hidden design intentions in protocol use and to enable specification of communication behavior at appropriate levels of abstraction.

2 The CFR Language

In this section, we introduce CFR, a domain specific language for the specification of both protocols and message scenarios. CFR models can be interpreted in two ways: As a specification of a protocol or as a specification of a protocol analyzer. In our current implementation, we use them to generate analyzers that can monitor and automatically test even complex scenarios.

A protocol layer can be described by relating its communication patterns to the communication patterns on the next lower layer. CFR models do this in a machine-processable form by combining the following basic language constructs:

- A *channel* is a medium that transports messages. The messages obey a protocol.
- A *filter* is a passive and stateless unit that redirects messages from an incoming channel to one of two outgoing channels. Depending on a given pattern, the filter determines which message is to be delivered to which outgoing channel. Since both the input and output of a filter are messages, the output of a filter can be used as the input of another filter. Filters might be put in parallel or in sequence. Filters are passive elements as they do not modify or create messages.
- A *rule* is an active and stateful message processing unit that consumes messages from one or more incoming channels and returns messages to one or more outgoing channels. In opposition to filters, a rule can modify incoming messages or create new messages. To do that, a rule contains defined logic in form of a state machine or a set of state machines with a selector function. Similar to filters, the output of one rule might be the output of another rule.

We represent the inputs and outputs of rules by pins with connectors between them. The message flow is from output to input pins. Channels and rules have an arbitrary number of input and output pins, while filters have exactly one input and two output pins. We use regular expressions to define the selection criteria in filters. State machines are used to define the memory of rules. We provide an example of a CFR model in Section 4.

3 Uncovering Abstract Protocols

A concrete protocol is a specification of messages and a set of message sequences. An abstract protocol constitutes its own communication architecture based on a concrete protocol. The communication architecture may be layered. At application level, however, strict layering is often impractical and broken by tunneling. An example is due. HTTP is a stateless and connectionless concrete protocol. It consists solely of request and reply messages and only a client can initiate communication via a request message. An application using HTTP may compensate for HTTP limitations and define its own interaction scheme. For instance, it might be necessary for a web browser based application to receive

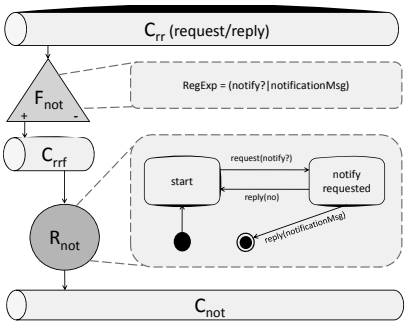


Figure 1: CFR model for the notification example in visual notation

notifications of important events from a web server. As the server cannot contact the client directly, an application programmer needs to design an abstract protocol to implement event notifications based on HTTP.

One way to realize the "illusion" of a server-based notification service is to let the client periodically send request messages to the server, asking whether there is a notification available. Upon receiving a request, the server replies by either sending a negative indication or it delivers the notification. This is exactly how the now popular AJAX web technology works. Other parts of the application might use the notification service via the abstract protocol – but they are not forced to do so. Application designers working with such a notification service certainly think in terms of notifications and probably do not even know about requests and replies. A conventional protocol analyzer on the other hand would only analyze the HTTP protocol, not knowing anything about the abstract protocol and its communication architecture. With CFR, we can specify abstract protocols and generate analyzers that understand these protocols.

Figure 1 shows a CFR model that defines the notification protocol just described. In our visual notation, Channels are tubes, filters are triangles and rules are circles. Channel C_{rr} transports all request/reply messages. The Filter F_{not} ensures that channel C_{rrf} transports only those requests and replies related to the notification service. The rule R_{not} uses a state machine to transform a request followed by a positive reply into a notification message and puts it on channel C_{not}.

4 Testing Scenarios with CFR Analyzers

Communication behavior in a distributed system can be described by a set of scenarios. A scenario for the aforementioned example might, for instance, be an update occurring on the server of which the client application needs to be notified. On the communication level, this involves at least two messages: An update message to the server followed by a notification message to the client. An effective method for testing the correctness of such a scenario is to stimulate the server by sending an update message and to check if the

expected notification message is sent. Using a conventional protocol analyzer, the stimulation and testing could in principle be done. However, the specification of the scenario would be at an inappropriate level of abstraction, namely at the request/reply layer.

If we view scenarios as sequences of messages, we can use CFR models to specify them as specialized high-level abstract protocols. By following a layered approach [HB05], we ensure that these specifications are always at the appropriate level of abstraction. We can specify even complex scenarios involving different protocols and subnetworks by combining all relevant messages onto a single abstract protocol layer and by then using this protocol layer for the definition of the scenario layer.

We require scenarios to be uniquely identifiable via their message sequences. Ambiguities have to be resolved at the specification level. This ensures that the step from merely monitoring a system to testing it is feasible. Based on a set of CFR models, we can use our analyzers to identify unspecified - and thus potentially faulty - communication behavior. In addition to that, we can use the analyzers to systematically stimulate components in order to trigger scenarios that are then verified. Under real conditions, a large number of different concurrent scenarios might be monitored simultaneously. We can refine our analyzers by assigning time intervals to the transitions of state charts within rules. This defines how long the monitor waits for the continuation of a scenario. If the expected future is not confirmed within a given time frame, tracing of the scenario is canceled and an error is reported.

5 Conclusions and Future Work

Our work is related to the large body of work done in forward engineering and analyzing protocols. Using a formalism based on channels, filters and rules, augmented with state charts and regular expressions, we are able to specify, monitor and test even very complex scenarios in distributed systems. Our layered approach ensures that specification can always be performed at the right level of abstraction. We have developed both a textual and visual notation for our language and extensively use code generation techniques to generate analyzers for different technology platforms. Currently, we are using our approach for testing AJAX based web applications and automotive systems [MH07] in order to gain more real-world experience. In future work, we will apply machine learning techniques for the automated extraction of CFR models from communication data and we are working on giving CFR a stronger formal basis by mapping CFR models to finite state processes.

References

[HB05] Dominikus Herzberg and Manfred Broy. Modeling layered distributed communication systems. *Formal Aspects of Computing, 28(4):751-763*, May 2005.

[MH07] Ansgar Meroth and Dominikus Herzberg. An Open Approach to Protocol Analysis and Simulation for Automotive Applications. In *Embedded World Conference*, 2007.

Path Constraint Construction with Lookahead

Ralf Gerlich

ralf.gerlich@bsse.biz*

Abstract: One of the basic tasks of constraint-based testdata generation is the selection of paths to be executed and the construction and solution of the associated path constraint. Here the problem of infeasible paths is not negligible. In this paper a new constraint-based method applicable to general control-flow-graphs is proposed for selecting feasible paths and at the same time constructing the associated path constraint. The method also allows the flexible application of methods for predicting control flow, thereby enhancing the identification of infeasible paths in advance.

1 Introduction

When automating software test the same tasks have to be carried out as for manual software test, among them the selection of test data. While randomized techniques allow unbiased coverage of the input domain, they in some cases fail in achieving required code coverage. This problem arises mostly from the fact that some specific paths through the system-under-test (SUT) are related to only a small subset of the input domain, and therefore are difficult to hit randomly.

Gotlieb[GBR98] originally introduced a constraint-based concept for generation of test data covering a given statement in the SUT, based on the semantics of control-flow constructs, such as `while` and `if`. The method can examine sequential constructs in parallel, thereby predicting parts of the control flow and allowing for elimination of some infeasible paths in advance. The concept is Turing-complete in theory, but in practice a considerable amount of effort is necessary to extend it to languages used in industrial practice. Sy and Deville[SD01] modified this method so that it applies to control-flow-graphs (CFGs), however without path prediction.

An alternative method incorporating both applicability to CFGs and path prediction, is introduced in this paper. The paper has three further parts. In Section 2 the method is described, followed by a short report on a first implementation in Section 3. Finally, conclusions are drawn in Section 4.

*This work is supported by a doctorate scholarship of the University of Ulm, Germany

1.1 Constraint Handling Rules

Constraint Handling Rules (CHR) is a declarative high-level language for specifying customised constraint solvers[Frü98]. A CHR program consists of a set of guarded rules which are used to transform a given constraint goal consisting of user-defined and builtin constraints until it is solved.

The rule forms available in CHR are simplification, written as $H \Leftrightarrow G|C$, and propagation, written as $H \Rightarrow G|C$, with the *head* H being a conjunction of user-defined constraints, the *guard* G a conjunction of built-in constraints and the *body* C a conjunction of built-in and user-defined constraints.

Simplification rules model replacement of user-defined constraints by logically equivalent constraints. Propagation introduces new, logically redundant constraints, which may lead to further simplification, e.g. in case of $A \leq B, B \leq C \Rightarrow A \leq C$. The language CHR$^{\vee}$ extends the theory of CHR by search capabilities by introducing disjunction in rule bodies.

2 The Method

In terms of constraint programming semantics, the problem at hand is best described as constructively proving the existence of a feasible path through the CFG from some node a to some node b. A path is *feasible* if and only if there is some input to a which can lead to execution of the path, otherwise it is infeasible. Consider, for example, the following program operating on two input arrays a and b, both with length l:

```
equal:=true;
for i=1 to l do
        if a(i)!=b(i) then equal:=false; break; end if
end for
if equal then print "Arrays are equal!"; end if
```

Given that a path is desired in which the `print`-statement is executed, any path via the statement `equal:=false` is infeasible. As a, b and l are to be found according to the desired path, there is a potentially infinite set of infeasible paths. Large subsets of infeasible paths can be excluded by a proper search strategy.

The input to the algorithm shall be given in the form of a CFG, consisting of a set of nodes and a set of edges. In the following $a \rightarrow b$ shall express the fact that there is an edge from node a to node b. Further $a \rightarrow^{+} b$ expresses that there is a non-empty sequence of edges from a to b. The nodes are labelled by their code text, and the edges are labelled by their associated conditions. An example for an implementation of the greatest-common-divisor-algorithm is shown in Figure 1.

The constraint $i\ B_n\ o$ shall be fulfilled if and only if the body of node n transforms the input i to the output o. Similarly, $o\ C_{a \rightarrow b}\ i$ is fulfilled if and only if the memory states o and i are equal and fulfill the condition for edge $a \rightarrow b$.

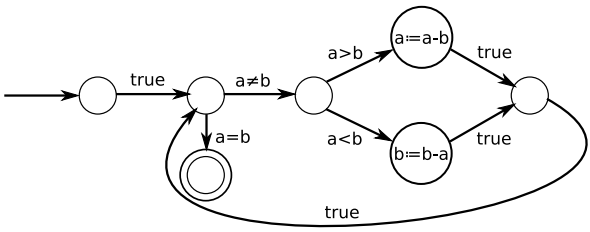


Figure 1: Labelled CFG of a gcd-Implementation

To express the feasibility constraint, a constraint $path\,(a, o_a, i_b, b)$ is introduced, which is fulfilled if and only if there is a path from a to b on which o_a, the output of a, is transformed to i_b, the input of b. By not using the input of a and the output of b directly, the calculations are simplified: o_a resp. o_b are easily found from i_a resp. i_b by the bodies of a resp. b.

The constructive proof is based on a divide-and-conquer strategy, where the path is either completed by a direct edge or split at some node n lying on some path from a to b. This strategy is described by the rules *Divide* and *Conquer* in the following CHR^v program. Note that these two rules actually have to be read as a disjunction for the program to be declaratively correct.

$$
\begin{array}{lcl}
\textbf{PredictVar} \mathbin{@} path\,(a, o_a, i_b, b) & \Rightarrow & v \in constvar\,(a, b) \mid o_a.v = i_b.v\ . \\
\textbf{PredictPath} \mathbin{@} path\,(a, o_a, i_b, b) & \Leftrightarrow & n \in allpaths\,(a, b) \\
 & \mid & path\,(a, o_a, i_n, n)\,, path\,(n, o_n, i_b, b)\,, \\
 & & i_n\ B_n\ o_n\ . \\
\textbf{Conquer} \mathbin{@} path\,(a, o_a, i_b, b) & \Leftrightarrow & a \rightarrow b \mid o_a\ C_{a \rightarrow b}\ i_b\ . \\
\textbf{Divide} \mathbin{@} path\,(a, o_a, i_b, b) & \Leftrightarrow & a \rightarrow^+ n, n \rightarrow^+ b \\
 & \mid & path\,(a, o_a, i_n, n)\,, path\,(n, o_n, i_b, b)\,, \\
 & & i_n\ B_n\ o_n\ .
\end{array}
$$

The rule *PredictVar* handles the case of variables which cannot be modified between on any path from a to b. This is the case when the definition nodes for v are either not reachable from a or cannot reach b, and is expressed by the constraint expression $v \in constvar\,(a, b)$.

The rule *PredictPath* predicts control flow by splitting the query at nodes which are traversed in any path from a to b, if any such node exists, using the builtin constraint $n \in allpaths\,(a, b)$. A special case of such nodes are the so-called dominators, which can be found efficiently from the CFG[LT79], and can be extended to the general case by using an appropriately chosen subset of the CFG with a as entry and b as exit node.

In the example, *PredictPath* can predict that the `if`-statement containing the `print`-statement must be executed. One of the predecessors of the `if` is infeasible due to the conflicting assignment, leaving only those paths exiting the loop normally, all of which are feasible.

3 Implementation

A prototype was implemented in Java for an imperative subset of Java. The builtin solver was implemented in JCHR[VWSD05] and specifically optimised for both solving and early inconsistency detection. To avoid recurring search of path subsets, *Divide* was refined to only select direct successors of a resp. predecessors of b.

Preliminary measurements on the generation rate of criteria-fulfilling test-inputs have been executed. They show that the new algorithm is about 10^5 times faster than random test-data generation for the hard-to-hit cases, while in simpler cases it is about 10^3 times slower.

4 Conclusions and Outlook

The algorithm presented is a generally applicable algorithm dealing directly with the CFG and performing control-flow prediction. However, to avoid bias introduced from the user-specified goals and to allow statistical evaluation of the results based on large test-input sets, it is recommended to use the algorithm only as a second complementary step to random test data generation.

Further work will focus on statistically controlling path length and finding mutually exclusive choices of path split-points, as well as on systematic performance measurement and extension of the supported language set in the prototype.

References

[Frü98] Thom Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.

[GBR98] Arnaud Gotlieb, Bernard Botella, and Michel Rueher. Automatic test data generation using constraint solving techniques. In *ISSTA '98: Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis*, 1998.

[LT79] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

[SD01] Tran Nguyen Sy and Yves Deville. Automatic Test Data Generation for Programs with Integer and Float Variables. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE)*, 2001.

[VWSD05] Peter Van Weert, Tom Schrijvers, and Bart Demoen. K.U.Leuven JCHR: a user-friendly, flexible and efficient CHR system for Java. In Tom Schrijvers and Thom Frühwirth, editors, *Proceedings of the Second Workshop on Constraint Handling Rules (CHR 2005)*, pages 47–62, Oct 2005.

Simplifying the Description and Application of Tests

Daniel Brenner and Colin Atkinson

Software Engineering Group
University of Mannheim, 68131 Mannheim
{dbrenner, atkinson}@uni-mannheim.de

Abstract: This paper provides a brief introduction to test sheets, a new approach to test specification and execution that combines the expressiveness of programmatic test definition approaches such as JUnit with the readability of tabular approaches such as Fit using a spreadsheet metaphor.

1 Introduction

Testing has always been one of the key techniques used in software engineering to ensure that software components do what they are expected to do, but its importance has increased in recent years with the rise of test-driven development methods such as extreme programming [Be02]. As well as providing a prescriptive description of how to evaluate whether a component is fit for purpose, in principle tests are also an excellent way of specifying what service a component offers or what requirements a component should fulfill. However, they are rarely used for this purpose today, because the languages and notations used to describe tests are highly implementation-oriented and, thus, are meaningful only to developers. In fact, tests are usually written in the same programming language used to implement the software.

In the Java community, for example, tests are usually written in Java using JUnit. They are very detailed and, without the necessary knowledge about Java and JUnit, are unintelligible to people who are not experts in Java. To make tests useful as a specification technique that is accessible to customers a much more concise and high-level language for defining them is needed. One such approach is Fit [Mu05]. Fit was introduced to give customers the possibility to create tests. The tests are specified in a very high-level and abstract form and do not require any programming skills or any knowledge about the actual details of the software.

Fit has two major disadvantages, however. First, in contrast to JUnit, the user defined Fit test are not immediately executable. Developers need to translate the high-level form into an executable test. Second, Fit is rather limited in its range of functionality, and in particular is not well-suited to defining tests on software abstractions with state (i.e objects). To address this problem we have developed a new test definition approach which we refer to as test sheets. It combines the advantages of high-level approaches like Fit and programmatic approaches like JUnit. In this short paper we give a brief overview of the main ideas behind test sheets.

2 Simple Test Sheets

Figure 1 shows an example of a simple test sheet defining a test case for a simple component representing a bank account. This component exports three operations, deposit(), withdraw(), and getBalance(). The first two take a floating point value as input parameter but return no result, while the latter has no input parameter but returns a floating point value.

	BankAccountTest		A	B
1	BankAccount	create		
2	B1	deposit	32.33	
3	B1	withdraw	20.20	
4	B1	getBalance		12.13

Figure 1: Simple Test Sheet

A test sheet essentially consists of a sequence of method invocations: each row in the table representing a particular operation invocation. It also contains a so-called invocation line which divides the test sheet into two parts. The left-hand side (LHS) holds all the input values and the right-hand side (RHS) shows the expected output values. The number of labeled columns on the LHS is determined by the operation with the highest number of parameters. The number of labeled columns on the RHS is determined by the number of output values supported by the language used to implement the class. Since Java, for example, allows only one return value per method, there is only one column on the RHS. The first row (row 1) in Figure 1 shows the creation of a BankAccount. The first cell in that row identifies which entity is invoked (i.e. the BankAccount) and the next cell identifies which operation of the entity is executed (i.e. the constructor). Since this operation has no parameters the other cells before the invocation line are grayed out.

Test sheets look very much like spreadsheets in popular office suites such a Microsoft Office (Microsoft Excel) and OpenOffice (OpenOffice Calc). A spreadsheet is a matrix of labeled rows and columns. In a test sheet each labeled cell acts like a variable that holds a value. As in programming languages this value can be of a primitive data type or can point to an object. The advantage of the spreadsheet style is that it is possible to refer to each cell by its row and column identifier. This, in turn, makes it possible to reference the values stored in other cells. Thus, the literal value 12.13 in B4 could be replaced by "A2-A3" (see Figure 3), since this is the intended relationship between the values (and is how the value 12.13 is calculated). For example, in row 1 the output from the invocation of the BankAccount's create() operation is stored in the output column (B1) and this value (the pointer to the created instance) can be referenced in subsequent invocations using the label of this cell. Thus the label B1 appearing in the first column of all subsequent row (after row 1) refers to the instance of BankAccount generated by the execution of the first row.

The result columns of the RHS contain the value that the operation is expected to return based on the specified input parameters and the operations executed previously. If an operation delivers no result this column is simply left empty. The default comparison operator is "==". Thus, cell B4 in Figure 1 specifies 12.13 as the expected result [Br08]. This means that the actual result which the operation getBalance() delivers, which is stored in B4, should equal ("==") 12.13. Besides the default ("==") comparison operators "<=", "<", ">", ">=", and "<>" are also allowed. With these operators it is possible to specify almost all required comparisons. If the compared values are of primitive data types, the regular comparison operators will be used. If they are complex data types, the appropriate operations of the associated class will be used. In the case of Java this will be the equals() method. Each comparison can thus conceptually be mapped to an equivalent JUnit assertXYZ() method.

When a fully specified test sheet is applied to the software under test, a new related test sheet is created which displays the result of the test. The columns on the RHS of such "result" test sheets play a special role – they indicate whether the expected value was actually returned using the color coding popularized in JUnit. When the actual returned value matches the expected value the cell is colored green, and when it does not it is colored red and the actual returned value is shown alongside the expected value. **Fehler! Verweisquelle konnte nicht gefunden werden.** show an example of a result test sheet corresponding to the data test sheet in Figure 1.

	BankAccountTest		A	B
1	BankAccount	create		
2	B1	deposit	32.33	
3	B1	withdraw	20.20	
4	B1	getBalance		12.10 | 12.13

Figure 2 Result Test Sheet

3 Higher-Order Test Sheets

The test sheets we have considered so far only contained literal values or, in the case of references, only "internal" references that refer to values inside the same test sheet. This allows abstract relationships to be specified between operation input parameters and result values, but only for a given set of input values. With simple tests of the form just described, the overall test logic cannot be applied to more than one set of input values – that is, it cannot used as the basis for data-driven testing. In data-driven testing, sets of tests (i.e. input data as well as expected output data) are defined and each is executed using the same test logic. This has the effect of injecting the data in each into the each execution of the test logic. Such tests are also called parameterized tests [Ti05].

This can be realized using enhanced forms of test sheets – parameterized test sheets and higher-order test sheets. Parameterized test sheets are like simple data test sheets except that they contain at least one value that is injected into the test sheet when it is executed. Figure 3 is such a parameterized version of the simple bank account test sheet where two values, labeled ?A and ?B, are parameters inject from outside.

BankAccountTest			A	B
1	BankAccount	create		
2	B1	deposit	?A	
3	B1	withdraw	?B	
4	B1	getBalance		A2 - A3

Figure 3: Parameterized Test Sheet

A parameterized test sheet cannot be tested directly but only as a subject of a higher order test sheet. To do this, the parameterized test sheet is regarded as a component (e.g. BankAccountTest in this case) with a single operation, test(), which has an input parameters corresponding to each parameter of the test sheet (in this case ?A and ?B), defined in alphabetical order. In addition the test() operation returns a string which contains a comma separated list of the output cells which would be colored red in a particular execution of the test sheet on a particular set of input data.

BankAccountHOSheet			A	B	C
1	BankAccountTest	test	32.33	20.20	null
2	BankAccountTest	test	...	...	...

Figure 4: Higher Order Test Sheet

Figure 4 shows such a higher order test sheet holding the values. They are basically like ordinary test sheets except they are applied to the conceptual test() operation of the conceptual component derived from a parameterized test sheet. As before the first column contains the row number, followed by the software under test, the invoked operation, the operation's input parameters, and the expected output. There are two specialties here. The first one is that, instead of the name of a component, the name of the lower order test sheets is specified. Each test sheet only has the one operation, test(), which is called in each row with a different set of input value. The second is that the required value for the return string value is stated to be null (i.e. empty) since one usually desires each test (i.e. invocation of the parameterized test sheet) to have no discrepancies. This means that the error string returned from each invocation should be empty (i.e. should identify no red cells). This capability can now easily be combined with the automatic generation of test data to support data-driven testing.

4 Conclusions

Unlike other tabular test definition approaches like Fit, test sheets are intended to be directly executable. We implemented a first test sheet execution environment and we are currently evaluating it. The evaluation results are used to further improve the approach.

References

[Be02] Beck, K.: Test Driven Development: By Example. 2002.

[Mu05] Mugridge, R.; Cunningham, W.: FIT for Developing Software. Framework for Integrated Tests, Robert C. Martin, 2005.

[Br08] Brenner, D.; Atkinson, C.: Software Testing using Test Sheets, ISSTA. 2008 (submitted)

[Ti05] Tillmann, N.; Schulte, W.; Grieskamp, W.: Parameterized Unit Tests. 2005.

Statistical Testing within the Scope of the System Validation of a RIS/PACS System

S. Holpp[1], H. Goetz[2], W. Dulz[3]

[1]IKM ES SV, Siemens Healthcare
Hartmannstr. 16, 91052 Erlangen
Stefan.Holpp.ext@siemens.com

[2]IKM IQ SQ, Siemens Healthcare
Hartmannstr. 16, 91052 Erlangen
Helmut.H.Goetz@siemens.com

[3]Department of Computer Science 7
Martensstr. 3, 91058 Erlangen
Dulz@informatik.uni-erlangen.de

Abstract: The division IKM of Siemens Healthcare develops and tests the software platform *syngo* Suite, an integrated RIS/PACS solution for radiology departments. Within the scope of a diploma thesis, the practicality of the Statistical Testing approach was proven in comparison to present deterministic testing.

1 Introduction

The core objective of statistical testing is *not* to improve the systems quality by finding more software defects, but to *estimate* the achieved quality. Statistical testing is based on *Markov chain usage models* which represent all possible uses of a software system at some level of abstraction. From these models, test cases can be derived which represent a statistical sample of possible uses (see figure 1). Based on this sample, statistics like a reliability estimator can be obtained ([BD03]). A usage model describes the expected use of a software system in terms of a Markov chain by adaption of the transition probabilities. This provides the possibility to create customer profiles which reflect the expected system usage of specific customers or customer types, based on a common software system.

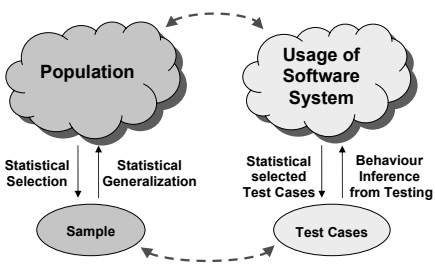
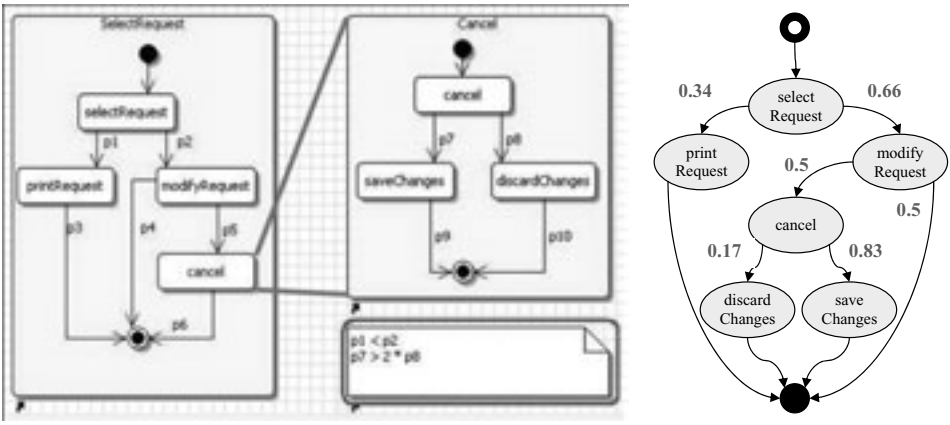


Figure 1: Statistical Testing Approach

2 Generation of Markov Chain Usage Profiles

In the department of *System Validation*, UML activity diagrams are used to model use cases from which test cases are derived. To handle growing model complexity, these activity diagrams follow a hierarchical[1] approach. One main objective was to find a feasible way to map those models to Markov chains. We have shown that it is possible to define a mapping between the hierarchical structure of present UML activity diagrams and a semantical equivalent (flat) Markov chain. This mapping has further been implemented so that the conversion can be performed automatically.



(a) Hierarchical UML

(b) Resulting Markov Chain

Figure 2: Generation of Customer Usage Profiles from hierarchical UML Diagrams

In order to define customer usage profiles, appropriate transition probabilities had to be obtained. In [JHPW00], the authors propose a constraint based approach to define customer usage behavior. This approach is based on statements like *"I do not know the probability of p_1, but I know that it is at least twice as big as p_2"*, and therefore provides an intuitive way to describe customer behavior.

These statements can be expressed by (in)equations. In this case the resulting inequation would be $p_1 > 2 \cdot p_2$. A new mathematical approach has been developed [Hol08] to obtain a suitable probability distribution from a set of such constraints (see figure 3).

For the creation of customer profiles, the long years of customer experience within IKM has been utilized. Customer experts and application trainers have been interviewed and it turned out that the constraint based approach is a very suitable way to convert informal customer knowledge into mathematical constraint definition.

[1] Actions within an activity diagram may reference other activities. This ends up in a tree-like composition of multiple activity diagrams.

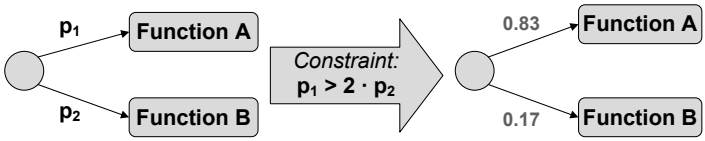


Figure 3: Constraint Based Approach

3 Metrics

Markov chains provide the possibility to derive a multitude of metrics out of the created customer profiles. These metrics can be obtained analytically so that no further effort is necessary. For example, two customer profiles can be compared in order to obtain a quantitative measure of behavior deviation. Of course, this information may have a huge influence regarding test planing. Imagining two customers with strongly deviating system usage. Intuitively, both customers are likely to demand for different testing in order to assure high quality results.

Further, the expected fraction of times a user performs an specific action, observed over a long period of time, can be obtained. While the specific use case is continuously repeated (for example several days or weeks), the user will take different paths through the underlying model. The average of this set of paths can be taken to calculate the long run probability of being in each state while executing this use case. This information can be used to analyze the usage of specific features of the system. Highly used features then may ask for improved usability or performance to satisfy customer needs.

4 Test cases and Reliability

Statistical testing demands the choosing of test cases in a statistical correct manner. To achieve an unbiased set of test cases, sampling must be done randomly but based on the underlying usage profile. Consequently, paths with high probability are more likely to occur in a test set than low probable paths.

In comparison to present deterministic test case selection based on path prioritization or path coverage algorithms, the statistical approach provides the possibility of a customer-based quality assessment by performing a reliability estimation. Therefore, the Miller Reliability Model ([PP04]) has been chosen since it offers many advantages like the possibility to consider a-priori knowledge about the system. After a set of test cases has been executed, test cases, test results and a usage profile are taken to calculate the Miller reliability estimator. This can be also done *during* test execution so that the reliability progress can be observed over the whole testing period. Further, usage profiles can be exchanged in order to obtain customer specific reliability estimators.

From this, several possibilities to define a test end criterion emerge. For example, testing can be stopped if the reliability for a specific customer type or an average over available usage profiles has reached a certain stability level which can be verified by the reliability variance. Another possibility is to stop testing if the tested paths do correspond closely to a chosen usage profile. The amount of correspondence between tested paths and expected usage can thereby be obtained analytically.

Suggestions on how to apply the statistical testing approach over a full release cycle of the *syngo* Suite product are presented in [Hol08] as well as results based on present test data. In addition to that, a future pilot project is expected to verify the described approaches for their applicability in the domain of IKM *System Validation*.

5 Conclusions

The statistical testing approach is a promising extension to present deterministic testing. We have shown that the creation of Markov chains based on present UML models can be done with a minimum of manual effort. The constraint based approach represents a suitable technique to transform present customer knowledge into Markov chain usage profiles. From this profiles, a multitude of metrics can be obtained analytically to improve test planning and support management decisions. A possibility to achieve a customer based reliability estimator at every stage of a test execution phase has been shown as well as the definition of possible test stopping criteria. By being able to obtain quantifiable customer based statements during the complete test cycle, decision processes for quality assurance can be significantly supported.

References

[BD03] Matthias Beyer and Winfried Dulz. Scenario-Based Statistical Testing of Quality of Service Requirements. In Stefan Leue and Tarja Systä, editors, *Scenarios: Models, Transformations and Tools*, volume 3466 of *Lecture Notes in Computer Science*, pages 152–173. Springer, 2003.

[Hol08] S. Holpp. Proving the Practicality of the Scenario-based Statistical Testing Approach within the Scope of the System Validation of a RIS/PACS System., 2008.

[JHPW00] Walton G.H. J. H. Poore and J.A. Whittaker. A constraint-based approach to the representation of software usage models. *Information and Software Technology*, 42(12):825–833, 2000.

[PP04] S. J. Prowell and J. H. Poore. Computing system reliability using Markov chain usage models. *J. Syst. Softw.*, 73(2):219–225, 2004.

Static Bug Detection Through Analysis of Inconsistent Clones

Elmar Juergens, Benjamin Hummel, Florian Deissenboeck, Martin Feilkas

Institut für Informatik
Technische Universität München
{juergens,hummelb,deissenb,feilkas}@in.tum.de

Abstract. Existing software systems contain a significant amount of duplicated code. Such redundancy can negatively impact program correctness, since inconsistent updates to duplicated code fragments are prone to introduce subtle bugs. This paper outlines our work-in-progress to statically detect inconsistencies in duplicated code fragments in order to find clone-related bugs. We illustrate the problem of clone related bugs with the example of such a bug in Eclipse, outline our algorithm for detecting inconsistencies in clones and report initial experiences from an industrial case study conducted with the Munich Re Group.

1 Introduction

Existing research in clone detection indicates that software systems typically contain a significant amount of duplicated code. Of all lines of code in the respective system, 8,7% of GCC [DRD99], 19% of X Windows [Ba95], 22,7% of Linux and 29% of JDK [Li06] have been found to be part of at least one duplication. Code duplication can be observed independent of programming language, target platform or application domain [Ko07].

Cloning renders maintenance difficult, since changes to one instance of a duplication (clone) potentially affect the other clones as well. Or, if a bug is contained in a duplicated code fragment, it needs to be removed from all its clones. If one or more clones are overlooked by the developer while he performs the change operation, subtle bugs can be introduced, or known bugs can remain in a system. In response to this, numerous clone detection approaches and tools that help to identify regions of duplicated code have been proposed by the research community [Ko07]. Such tools are valuable in reducing the risk of introducing inconsistencies during maintenance of duplicated code, since they help programmers in identifying all potentially affected code fragments during a change operation. Their shortcoming is that they cannot detect clone-related bugs once they have entered a system. To mitigate this, we outline an analytic approach that helps to detect faults causing such bugs contained in a system.

Contribution: We illustrate the risk cloning poses to program correctness and propose an approach to statically detect clone-related bugs. Our approach is applicable to both modern and legacy programming languages, for which few bug detectors exist. We show preliminary results from an industrial case study that indicate that we can detect a significant number of defects in practice.

2 Detecting Inconsistently Evolved Clones

A clone is a sequence of statements that can be found more than once in a software system[1]. An inconsistent clone is a sequence of statements that would become a clone, if we inserted or deleted a small number of statements. An example of this is illustrated in Figure 2 that displays a code fragment from Eclipse 3.2.2, in which the statement `target.appendChild(export)` is missing on the left side[2]. As can be seen, the fragments of an inconsistent clone that are separated by gaps (insertions or deletions), are shorter clones themselves.

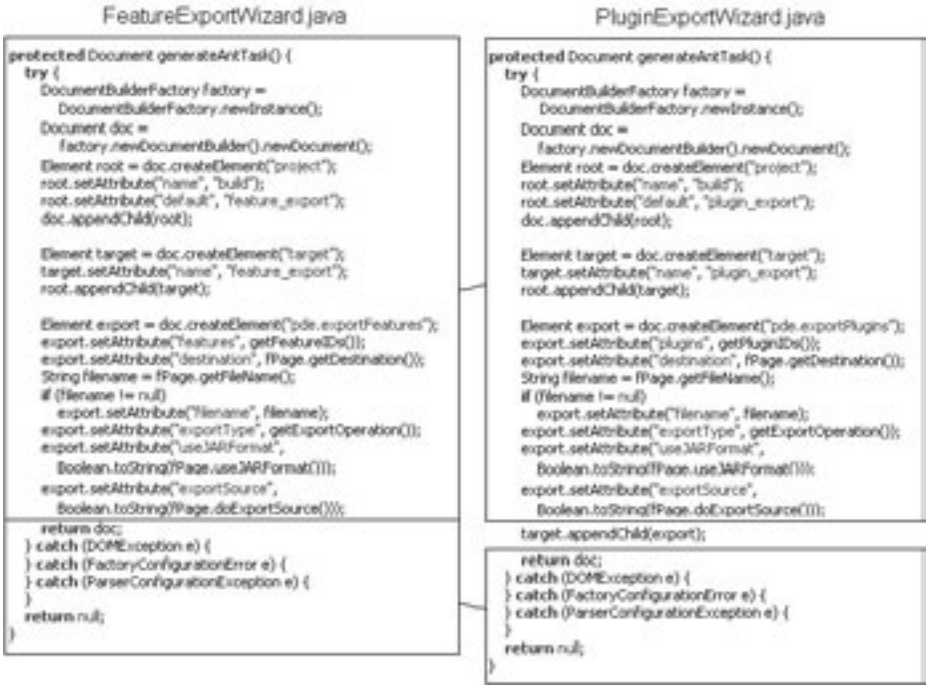


Fig. 1. Bug 155070 in Eclipse 3.2.2

This observation leads to a first simple algorithm for finding inconsistent clones: Firstly, it determines all consistent clones using one of the existing clone detection algorithms. Then it looks for pairs of clones which, together with their siblings, are sufficiently close to each other. These are then assembled into a single inconsistent clone. We implemented this detection algorithm as part of our open source software *Continuous Quality Assessment Toolkit ConQAT* [DPS05] that implements an extensible clone detection tool. Due to the language independence of the inconsistency detection, our implementation can be used with all languages that ConQAT supports, namely C/C++, Java, VB, C#, PL/I and Cobol.

[1] Typically minor differences in identifier naming or literal values are ignored.

[2] The missing statement results in loss of information and leads to incomplete generated XML. It is the cause of Eclipse Bug 155070.

3 Case Study

We have performed a case study of clone-related bugs due to inconsistent clones on an industrial business information system from the financial domain that is developed and maintained at the Munich Re Group. The system comprises 400 kLoC of C#, of which 215 kLoC are handwritten[3].

Setup: We performed the analysis in three steps: Firstly, we applied our tool to detect inconsistent clones. Secondly, we manually inspected the detected inconsistent clones to exclude inconsistencies that obviously do not indicate bugs and categorized the remaining inconsistencies according to their defect probability. Thirdly, the remaining inconsistencies are currently being inspected by the developers of the system under investigation.

Results: The detection produced 106 inconsistent clones (comprising 272 code fragments) and took approx. 30s and 40MB on a 1.7 GHz notebook. Manual investigation identified 67 inconsistent clones to be false positives, typically due to purely syntactical differences (e.g. inconsistent use of braces surrounding single statement conditionals and loops, order of commutative statements or use of local or fully qualified names). Of the remaining 39 inconsistencies (comprising 89 code fragments), 11 could be classified as bugs for technical reasons due to missing or incomplete null checks or inconsistent use of break statements that cause different behavior of the same algorithm in different places of the system. 8 inconsistencies due to unused or commented code were classified as style issues that, although they do not present defects at present, should be mended in order to simplify future maintenance. The remaining 20 inconsistencies were mainly caused by differences in conditions or missing method calls and cause variations in the system's behavior. More detailed knowledge about the system than available to us is required to decide whether they are desired or erroneous. They are still under investigation by the system's developers, but preliminary results are very promising.

Discussion: Although the evaluation result of 20 inconsistent clones by the system developers is still pending, the 11 bugs and 8 style issues identified by our analysis already indicate a substantial value of our analysis in practice. The resulting precision of 18%-38% (depending on the 20 pending assessments) can, while acceptable, still be improved by ignoring purely syntactic inconsistencies such as those mentioned above.

The current analysis approach still has drawbacks, since it only detects inconsistencies with a single gap that are encased by sufficiently large clones. Hence, we might miss relevant inconsistencies that are encased by clones that are too small to be discovered by the initial consistent clone detection phase. We plan to improve on this by applying an algorithm that directly detects gapped clones in our future work. The parameters for minimal clone and maximal gap size heavily influence precision and recall of detection results. Our choice of 5 as threshold for both for our case study is based on our experience gained in early experiments and demands further research.

[3] The remaining code is generated and has been excluded from the analysis, since it is not maintained by hand and thus cannot suffer from inconsistent manual changes.

4 Related Work

A variety of approaches and tools that search for bugs by performing different types of static program analyses—such as syntactic pattern matching or data flow analysis—has been proposed, see e.g. [RAF04,Zh06] for a survey. Our approach is complementary to them in that it detects bugs that typically cannot be detected with existing tools, since it uses duplication inconsistencies as additional information source unavailable to them. A large body of research exists that deals with various aspects, i. e. origin, evolution, detection and removal of cloning. Due to space constraints, please refer to Koschke's comprehensive survey [Ko07]. Approaches to detect gapped clones exist [Ko07], but are not targeted at bug detection. To our knowledge, only two approaches exist that employ clone detection to detect bugs. In [Li06], Li et. al. search for inconsistently renamed variables to detect clone-related bugs. In [JSC07], Jiang et. al. extend that approach by searching for different contexts of duplicated code fragments that potentially indicate bugs. Both approaches focus on detecting bugs that have been introduced during clone creation. In contrast, our approach focuses on detection of bugs introduced by inconsistent evolution of clones, which we consider the bigger threat to program correctness in long lived software systems.

5 Future Work

Empirical Analysis: We plan to conduct a comprehensive industrial case study to better understand the significance of clone-related bugs in practice.

Algorithm: We are currently developing an extension of suffix-tree based clone detection that directly detects inconsistencies in clones. We expect this approach to yield better completeness than our current one.

Increasing precision: We plan to increase the precision of our approach to detect clone-related bugs by ignoring purely syntactic inconsistencies and by exploiting evolution information from the version control system.

References

[Ba95] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proceedings of WCRE 1995*.

[DPS05] F. Deißenböck, M. Pizka, T. Seifert. Tool Support for Continuous Quality Assessment. In *STEP 2005*.

[DRD99] S. Ducasse, M. Rieger, S. Demeyer. A Language Independent Approach for Detecting Duplicated Code. In *Proceedings of ICSM 1999*.

[JSC07] L. Jiang, Z. Su, E. Chiu. Context-Based Detection of Clone-Related Bugs. In *FSE 2007*.

[Ko07] R. Koschke. Survey of Research on Software Clones. In *Duplication, Redundancy, and Similarity in Software*, Dagstuhl Proceedings, 2007.

[Li06] Z. Li, S. Lu, S. Myagmar, Y. Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale SW. *IEEE TSE*, 2006.

[RAF04] N. Rutar, C. B. Almazan, J. S. Foster. A Comparison of Bug Finding Tools for Java. In *Proceedings of ISSRE 2004*.

[Zh06] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J.P. Hudepohl, M.A. Vouk. On the value of static analysis for fault detection in SW. *IEEE TSE*, 2006.

Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung

Inhaltsverzeichnis

2. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung 449
Joerg Doerr, Peter Liggesmeyer

Assessing Relations between Non-Functional Requirements 451
Michael C. Jaeger, Anne Hoffmann

Qualitätsrisiken und deren Abhängigkeiten 457
Andrea Herrmann

Erfassung, Strukturierung und Überprüfung von Qualitätsanforderungen durch aktivitätenbasierte Qualitätsmodelle 464
Stefan Wagner, Florian Deissenboeck, Sebastian Winter

Über die Auswirkungen nicht-funktionaler Anforderungen beim Entwurf eines Rahmenwerkes zur Anbindung von Smart Items an ERP-Systeme 470
Rosemaria Giesecke, Florin Spanachi

2. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung

Joerg Doerr, Prof. Dr. Peter Liggesmeyer

Fraunhofer Institut Experimentelles Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern
joerg.doerr@iese.fraunhofer.de
peter.liggesmeyer@iese.fraunhofer.de

Abstract: Die Beachtung von nichtfunktionalen Anforderungen (auch häufig als Qualitätsanforderungen bezeichnet) ist essentiell für erfolgreiche Projekte und Produkte. Das Vernachlässigen dieser Qualitätsanforderungen führt häufig zu gescheiterten Projekten, geringer Produktqualität, verlängerter Time To Market und hohem Rework-Aufwand. In der Praxis werden nichtfunktionale Anforderungen oftmals kaum oder nur ad hoc behandelt; eine systematische Erhebung, Spezifikation und Analyse konkreter und messbarer nichtfunktionaler Anforderungen ist oftmals nicht zu beobachten. Dieser Workshops ist eine Forsetzung des 1. Workshops welcher auf der SE 2007 in Hamburg erfolgreich durchgeführt wude.

1 Einleitung

Sowohl für Eingebettete Systeme, als auch für klassische interaktive Informationssysteme ist die Beachtung von nichtfunktionalen Anforderungen (auch häufig als Qualitätsanforderungen bezeichnet) essentiell für erfolgreiche Projekte und Produkte. Neben der reinen Funktionalität finden Qualitätsattribute wie Effizienz, Benutzungsfreundlichkeit, Sicherheit und Zuverlässigkeit in den verschiedensten Domänen mehr Beachtung. Das Vernachlässigen dieser Qualitäten führt häufig zu gescheiterten Projekten, geringer Produktqualität, verlängerter Time To Market und hohem Rework-Aufwand. Demgegenüber stellen erfolgreich umgesetzte Qualitätsanforderungen Differenzierungsmerkmale der Produkte gegenüber Wettbewerbern dar. Qualität entsteht aber in der Systementwicklung nicht durch Zufall, vielmehr muss sie von Beginn an mitbedacht werden, in Design und Implementierung durchgesetzt und schließlich durch entsprechende Tests sichergestellt werden. Die Bruchstelle, die man in der Praxis häufig in diesem Ablauf findet, ist bereits die initiale Definition der Qualität in der Anforderungsphase. Ohne eine hinreichend detaillierte Definition der Qualitätsanforderungen ist es nur schwer möglich sicherzustellen, dass die notwendige Qualität in den späteren Phasen erreicht wird.

Die Spezifikation von Qualitätsanforderungen wird jedoch von vielen Ansätzen der Anforderungserfassung nicht oder nur unzureichend unterstützt. Dadurch werden zu abstrakte oder mitunter auch unangemessene Qualitätsanforderungen definiert. In der Praxis werden nichtfunktionale Anforderungen oftmals kaum oder nur ad hoc behandelt; eine systematische Erhebung konkreter und messbarer nichtfunktionaler Anforderungen ist oftmals nicht zu beobachten.

2 Ziele des Workshops

Nach einem erfolgreichen 1. Workshop auf der SE2007 bringt dieser Workshops erneut Praktiker und Akademiker, welche auf dem Gebiet der nichtfunktionalen Anforderungen arbeiten zu einem gemeinsamen Erfahrungsaustausch zusammen. Die Praktiker stellen im Workshop ihre aktuelle Verfahrensweise bzgl. nichtfunktionaler Anforderungen und aktuelle Herausforderungen dar. Aus akademischer Sicht werden aktuelle Lösungsansätze, Methodenbeschreibungen und aktuelle Forschungsthemen vorgestellt. Der Workshop fokussiert bewusst nicht auf spezielle Qualitätsattribute oder Domänen, um synergetische Diskussionen zu fördern: Was können Ansätze einer Domäne von anderen Domänen lernen? Wie können spezielle Ansätze für ein Qualitätsattribut (Usability, Performanz) ebenfalls für andere Qualitäten genutzt werden?

Die Ansätze und Methoden können verschiedene Fragestellungen bzgl. nichtfunktionaler Anforderungen beleuchten, beispielsweise:

- Wie stelle ich sicher, dass meine nichtfunktionalen Anforderungen messbar und testbar spezifiziert sind?
- Wie stelle ich fest, welche Qualitäten für meine Systementwicklung relevant sind (Priorisierung der Qualitätsattribute)?
- Wie kann ich Vollständigkeit gewährleisten?
- Wer sind die Stakeholder, die befragt werden müssen?
- Welche Rolle spielen Standards?
- Wie kann ich die Verfolgbarkeit meiner nichtfunktionalen Anforderungen zu meinen funktionalen Anforderungen und zur Architektur gewährleisten?
- Wie stelle ich sicher, dass eine möglichst minimale Menge von nichtfunktionalen Anforderungen spezifiziert werden, die somit den Entscheidungsspielraum für das Design nicht unnötig einschränken.
- Wie kann ich Erfahrung mit nichtfunktionalen Anforderungen aus alten Projekten wiederverwenden?
- Wie hängen interne Produktqualitätseigenschaften (z.B. interne/externe Qualitäten aus ISO 9126) mit vom Nutzer wahrnehmbaren Qualitäten (z.B. Quality in Use aus ISO 9126) zusammen?

Praktiker werden in diesem Workshop Einblicke in neue Ansätze zur Erhebung, Spezifikation und Analyse von nichtfunktionalen Anforderungen erhalten. Neue Forschungskooperationen zwischen Akademikern, aber auch zwischen Industrie und Forschung sollen durch diesen Workshop gefördert werden.

Assessing Relations between Non-Functional Requirements

Michael C. Jaeger and Anne Hoffmann

Siemens Corprate Research and Technology, Software & Engineering (CT SE)
Otto-Hahn-Ring 6, D-82000 München, Germany
michael.c.jaeger@siemens.com
anne.hoffmann@siemens.com

Abstract: The development of requirements involves also necessary checks for consistency, coherency and identification of dependencies between them. While approaches for requirements development exist that cover the functional characteristics of a system, the support for non-functional requirements is not well elaborated today. This work proposes to assess non-functional requirements (NFRs) by their relations between them in order to provide a reference for a systematic analysis of NFRs.

1 Introduction

Generally, a non-functional characteristic describes – contrary to a functional characteristic – the behaviour of system during its use. Accordingly, during the engineering process of a system, the requirements are separated into two major parts: functional requirements and non-functional requirements. Functional requirements constitute the nature of the desired system. In addition to that, non-functional requirements (NFRs) express requirements in order to enable the use of such system. Thus, requirements engineering must consider NFRs as important as the function of a system.

This work is driven by the fact that an NFR implies different actions and decisions during the subsequent specification and design of a system. Consider the example that a system must provide extraordinary reliability. In this case, a possible implication on the design is the redundant set up of critical components of that system. However, this also implies a higher operational cost, because the footprint of the system increases leveraging a redundant design. Thus, in this simple example, the NFR referring to the reliability interferes with a possible NFR on operational cost. The important point for the requirements engineering is: *What are the relations between different NFRs and what are the resulting implications for the requirement engineering phase?* In order to answer to this question, this work aims to identify the dependencies among NFRs.

To answer this question, we must determine the relevant NFRs for the engineering of software-intensive systems. The discussion on relations among NFR cannot be general or on an abstract level, but must consider the nature of individual requirements in a specific domain. For example, implications on capacity when constructing buildings are different as when considering software systems. Thus, this work entirely focuses on the requirements engineering of software-intensive systems. It lists relevant non-functional characteristics of these and it explains their interpretation today.

Finally, we can analyse how these NFRs interrelate with each other. Based on the interpretation of the NFRs, the implications are explained. Then, this discussion puts these implications in a common context in order to identify conflicts or – what is also considered possible – favouring effects. The rest of this work is organised in the following way: the next section gives a brief overview on requirements engineering. Section 3 discusses relevant NFR for software-intensive systems and leads to a rough overview of dependencies among these. Finally, Section 4 presents our conclusions so far and our next steps in this field.

2 Brief Overview on Requirements Engineering

In general, dealing with requirements can be separated into the development and the management of requirements. In requirements development, the main activities are elicitation, analysis, and specification. Beginning with the elicitation, an appropriate scope of the desired system is defined that limit the extent of the requirements. Then requirements are collected and a common understanding among the stakeholders of the development process needs to be ensured. In multi-party software development projects, the elicitation also involves the negotiation of requirements, for example between client and developers. As the next step, the analysis of the requirements takes place: Requirements are classified and checked for consistency and completeness. Also, dependencies among requirements would be identified in order to cover potential conflicts. Subsequently, the third step is about the specification of requirements, which includes their presentation, their structuring and the setting of criteria for determining under what conditions a product meets the requirements.
Besides the development of requirements, the management of requirements is also necessary, because in a software development process, subsequent steps, such as the design or the implementation take place beyond the developed requirements. Thus, a change management of the requirements is necessary that allows to apply and track modifications. Our work proposes to enhance the development of requirements: Our aim is to bring the focus of requirements engineering to checking consistency and dependencies with in particular with NFR. Our anticipated result is a comprehensive reference that provides requirements engineers with a guide when structuring NFRs. This reference enables the requirements engineer to:

- **Define implications of particular NFRs:** Using this framework, the requirements engineer can pick up relevant implications for the entire set of particular NFRs.
- **Discover possible conflicts between NFRs:** Knowing about implications of particular NFRs enables the requirements engineer to get a more comprehensive image of the desired system. Within this image, conflicting NFRs become explicit.
- **React to conflicts in the requirements phase:** With the identification of conflicts, the requirements engineer in able to apply appropriate corrections in an early phase of the system engineering process. This improves the control on the entire engineering process and reduces the overall efforts.

3 The Non-Functional Requirements and their Dependencies

The term non-functional requirement refers to the non-functional characteristics as requirements on the desired system. The main question that rises with non-functional characteristics is what their difference to functional characteristics are. Cases exist where a characteristic is functional in one application and is non-functional in another. For example, ensuring the provision of a communications link with a particular bandwidth for voice transmissions, the provided bandwidth is actually the provided function. However, with a Web server application, the bandwidth is clearly a non-functional requirement since the application would work with almost any bandwidth characteristics.
The research consensus in order to answer this question is to define the functional core of a provided function as discussed by Werner [Wer07]. Then, a characteristic that is not covered by the functional core is named non-functional. Therefore, for a precise definition, it depends on the application to unambiguously define which characteristics are considered non-functional. However, in most cases, the non-functional characteristics can be easily identified. In this area, several characteristics exist as a group. A large group is comprised by the quality-of-service (QoS), which are considered a subset of non-functional characteristics. Different works state that QoS characteristics are non-functional characteristics that are quantifiable (e.g. [OMG04]). In other words, the QoS refers to the measureable characteristics when a system provides its functionality.
In addition, to the cluster of QoS characteristics, other groupings exist such as the dependability or the RAMSS characteristics (Reliability, Availability, Maintainability, Security and Safety). Both are umbrella terms as well that comprise different individual characteristics. For this work, different work in the domain of distributed software systems are considered: the ISO QoS reference model as a supplemental proposal for the ISO Reference Model for Open Distributed Processing (RM-ODP) [ISO98] and the OMG UML QoS Profile [OMG04]. In addition, we consider the works of Colouris [CDK02], Tanenbaum and Goodman [TG01], Ran [Ran03] and Buschmann et al. [BMR+96] as references for non-functional characteristics in the area of distributed software systems in an informal way. Our selection includes the following characteristics:

- Organisational characteristics:
  - Location of provision: This characteristic refers to the location where the system is provided. This could involve for example different end user devices.
  - Cost (-): The cost for either realising or using the provided system.
  - Affiliation: The affiliation denotes the organisation that provides the system or its parts. For example, legal issues might be relevant with specific affiliations.
- Characteristics from the dependability area:
  - Reliability (+): The reliability of a system denotes the ratio of successful systems execution according to the specification.
  - Availability (+): The availability denotes a ratio on how many times the system's functionality can be accessed.
  - Safety (+): The safety is meant to cover the operational safety.

  - Security (+): This umbrella term refers to different particular characteristics, such as confidentiality. In some works, the safety is also included.

- Performance characteristics:
  - Bandwidth (+): The bandwidth denotes the amount of data that a system can process within a defined time interval.
  - Response Time (-): The amount of time to process a particular request.
  - Resource Consumption (-): This characteristic denotes the consumption of technical resources, for example, CPU time or main memory.

- Technical characteristics:
  - Maintainability: During its lifetime, the software must be updated, corrected or data must be maintained. In order to preserve the operation of the software, efficient mechanisms for maintenance are desired.
  - Extensibility: This characteristic refers to the suitability of the software for applying extension for the support of new functionality.
  - Deployability: Describes how easy or difficult a software is to deploy.
  - Scalability: Denotes how well a system performs with a rising amount of use.

In this list, a plus or minus sign indicates that the particular characteristic is quantifiable and thus has a direction. A minus indicates that lower values are considered better; a plus vice versa. A systematic selection could be based on assessing actual occurrences in the requirement engineering of industrial projects. However, this would exceed the available space for this work and is, thus, postponed as further work.
These characteristics result in particular implications on the requirements engineering when involved as non-functional requirements. Such implications result in new requirements or complement existing ones. For example, requirements on the scalability supplement requirements on the response time. A requirement on a high level of extensibility implies modular system architecture. With maintenance, the implications are more complicated as this is understood as a characteristic in order to improve the general availability, or reliability. However, modularity can result in a performance penalty, as this, for example, is an issue with the design of operation system kernels. But, to show the difficulty of this interpretation, modularity can also lead to the solution on the response time: For example, throughput-intensive application servers are modularised in order to balance the load in an optimal way.
In order to establish a starting point with the goals of in this work, an empirical assessment serves as a first proposal. This proposal is summarised in Figure 1. In this figure, a relation denotes that improving a particular characteristic:

1. worsens the other, depicted by an umbrella,
2. has no general impact on the other, shown in the table as grey box,
3. implies the improvement of another characteristic,
4. results in improving another characteristic, indicated by green boxed smiley.

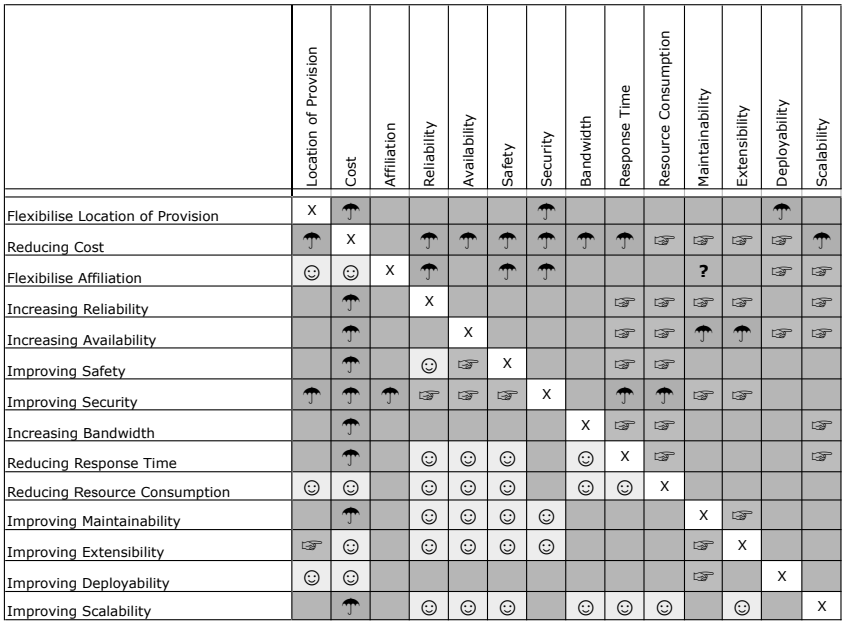


Figure 1: Summary on Relations

This figure represents a first result on our personal assessment. Some of the mentioned authors (e.g. [BMR+96]) already discuss implications of particular NFRs. However, to our best knowledge, we are not aware of work that provides a comprehensive presentation. Certainly, the particular relations as given by Figure 1 could be discussed in further detail. However, due to space limitations, we can give only a brief summary on our assumptions:

- Location of provision implies a modularity to some certain extend, because specification of special end user devices or particular location of provision improve with a separation of the business logic from presentation concerns.
- A cost improvement generally reduces other QoS characteristics. However, some non-functional characteristics result in improvement of the cost, such as low resource consumption or extensibility.
- To bring the affiliation into relation to other non-functional characteristics is generally difficult. It is listed here to demonstrate an unusual candidate in the discussion.
- The reliability implies some characteristics such as good response time or good safety. The availability is related to the reliability w.r.t. the other characteristics.
- The safety implies the proper working of the system and also benefits from a good maintainability.

- The security forms a trade off with performance characteristics and thus has also a negative on resource consumption.
- The bandwidth and response time are performance characteristics and thus result in good availability and reliability but worsen resource consumption and cost. They both benefit from a good scalability. The scalability is a hidden characteristic that results in other observable characteristics, such as reliability or response time.
- The maintainability has effects on most of the other characteristics. It results in good reliability but can also result in more cost. The same applies to the extensibility.
- The deployability has an effect on the cost and on the location of provision. However regarding the rest of the characteristics it develops neutral.

4 Conclusions and Future Work

This work has presented an overview on non-functional characteristics relevant for the requirements engineering. In addition, it has presented the role and handling of non-functional requirements in the requirements engineering field. Based on this, an outlined overview on how NFRs could depend on each other was given. Our future aim is to elaborate on this work in iterations in order to provide a mature reference about dependencies between NFRs. As the next steps in our future work we consider relevant an assessment of the importance and popularity of different non-functional characteristics for requirements engineering. This could involve *1)* the analysis of projects where the results of requirements engineering is accessible, *2)* the conduction of questionnaires and interviews, and *3)* further research in books and articles where particular implications are already discussed for individual NFRs.

References

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture – Volume 1*. Wiley &Sons, 1996.

[CDK02] George Colouris, Jean Dollimore, and Tim Kindberg. *Verteilte Systeme*. Pearson Education Deutschland, München, Deustchland, 2002.

[ISO98] ISO/IEC. CD 15935 Information Technology: Open Distributed Processing - Reference Model - Quality of Service. (CD Ballot), October 1998.

[OMG04] Object Management Group OMG. UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms. ptc/2004-06-01, June 2004.

[Ran03] Shuping Ran. A Model for Web Services Discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[TG01] Andrew S. Tanenbaum and James Goodman. *Computerarchitektur*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2001.

[Wer07] Matthias Werner. Eigenschaften Verlässlicher Systeme. Habilitationsschrift der Technischen Universität Berlin, 2007.

Qualitätsrisiken und deren Abhängigkeiten

Andrea Herrmann

Software Engineering Group, Institut für Informatik
Universität Heidelberg
Im Neuenheimer Feld 326
69120 Heidelberg
andrea.herrmann@informatik.uni-heidelberg.de

Abstract: Es hat sich bewährt, Softwarequalität anhand von Qualitätsrisiken zu beschreiben, d.h. anhand derjeniger Ereignisse oder Zustände, die man durch die Qualitätseigenschaft vermeiden möchte. In dieser Arbeit werden Qualitätsattributen jeweils ein unerwarteter Zustand zugeordnet sowie deren kausale Abhängigkeiten modelliert. Diese Abhängigkeiten können u.a. die Qualitätsvorhersage unterstützen.

1 Einleitung: Qualität versus Qualitätsrisiken

Softwarequalität kann man definieren als das Nicht-Vorhandensein oder die Minimierung von Risiken. Insbesondere wird dies in sicherheitsrelevanten Bereichen so gehandhabt [SO00], [SO01], [Fi03]. An der Universität Heidelberg wenden wir dieses Prinzip erfolgreich auf alle Arten nicht-funktionaler Anforderungen (NFA) an. Interessanterweise beschreiben die Definitionen aller Qualitätsattribute (QA) viel öfter einen unerwünschten Zustand des Systems[1] als den erwünschten Zustand [He07b]. Die gängigen NFA-Metriken messen meist die Auftretenshäufigkeit eines unerwünschten Zustands, manchmal den daraus entstehenden Schaden[2] [He07b].

Wir haben ausgehend vom Standard ISO 9126 [IS91] eine QA-Taxonomie entwickelt, in der jedes QA genau einem unerwünschten Zustand entspricht[3] [He07b]. Im Folgenden wird modelliert, wie diese unerwünschten Zustände kausal zusammenhängen und damit auch die QA und die Erfüllung von NFA. Dies soll uns helfen, Risiken zu quantifizieren, um darauf aufbauend NFA risikobasiert zu priorisieren wie in [He07a] dargestellt.

---

[1] Wobei hier zu dem „System“ nicht nur Hardware und Software zählen, sondern auch Benutzer und sonstiges Personal (Entwickler, Administrator, usw.), Entwicklungs- und Testwerkzeuge, die umgebende physische Welt (z.B. Gebäude) sowie Schnittstellen zu anderen Systemen.

[2] Oft besteht der Schaden in einer erhöhten Wahrscheinlichkeit eines nachfolgenden unerwünschten Zustands.

[3] Im Vergleich zur zweiten Ebene des ISO 9126 Standards wird in unserer Taxonomie das Attribut „Sicherheit“ untergliedert (in Betriebssicherheit, Vertraulichkeit und Überlebensfähigkeit), „Usability“ wird in andere Faktoren zerlegt (Produktivität, Bedienbarkeit und Zufriedenheit), „Portierbarkeit“ wird nicht näher unterteilt (was dann sinnvoll wäre, wenn man die einzelnen Schritte der Portierung betrachten möchte, was bisher jedoch nicht unser Schwerpunkt war).

Es werden zunächst in Abschnitt 2 die vier Typen unerwünschter Zustände dargestellt und in Abschnitt 3 die Zusammenhänge zwischen den unerwünschten Zuständen. Abschnitt 4 gibt einen Ausblick darauf, wo und wie wir diese Abhängigkeiten nutzen möchten und Abschnitt 5 fasst diesen Beitrag zusammen.

2 Unerwünschte Zustände

Softwarequalität wird beschrieben durch ein Spektrum an Softwareeigenschaften, die QA. NFA sind Instanzen von QA und beschreiben eine konkrete Anforderung an eine bestimmte Software. Welchem QA wir welchen unerwünschten Zustand zuordnen, ist in den Abbildungen 1-4 dargestellt. Wir unterscheiden vier Typen unerwünschter Zustände, die sich jeweils im Hinblick auf die sie quantifizierenden Metriken ähneln [He07b]:

- Typ D: Defekte (englisch: fault, defect oder bug) unterscheiden wir danach, von welcher erwünschten Eigenschaft das System abweicht (Abbildung 1).

- Typ F: Eine Fehlfunktion (englisch: failure) bedeutet, dass das Verhalten des Systems vom erwarteten Verhalten abweicht. Eine Fehlfunktion tritt auf, wenn ein Teil des Systems ausgeführt wird, der einen Defekt enthält [Oh84]. In Abbildung 2 sind die Fehlfunktionen dargestellt. Diese werden nach ihrer Ursache unterschieden.

- Typ A: Es entsteht unnötig oder inakzeptabel hoher Aufwand, Kosten oder Ausfallzeit bei Benutzung (Typ Ab) oder Wartung (Typ Aw). Die unerwünschten Zustände des Typs Aw werden nach ihrer Ursache (der durchgeführten Tätigkeit) unterschieden (Abbildung 3), des Typs Ab nach den Folgen (Abbildung 2).

- Typ S: sonstiges, wie Sicherheitsverletzungen (Abbildung 4), Benutzeraktion wird nicht erfolgreich durchgeführt oder Benutzer ist unzufrieden (Abbildung 2).

3 Kausale Abhängigkeiten der unerwünschten Zustände

Die unerwünschten Zustände verschiedener QA hängen kausal miteinander zusammen. Hierbei kann ein unerwünschter Zustand mehrere verschiedene nach sich ziehen oder ein erwünschter Zustand mehrere mögliche Vorgängerzustände haben. Aktivitäten führen von einem unerwünschten Zustand zum nächsten. Um die Abhängigkeiten der unerwünschten Zustände untereinander sowie die Aktivitäten darzustellen, ist ein UML-Aktivitätendiagramm geeignet, aber auch jede andere Notation, die Aktivitäten und Zustände unterscheidet und gerichtet verknüpft.

Wegen seiner Größe kann das eine UML-Aktivitätendiagramm aller QA hier nicht als Ganzes abgebildet werden. Darum zeigen die Abbildungen 1 bis 4 jeweils einzelne Swimlanes dieses Diagramms, einschließlich Querverbindungen zu unerwünschten Zuständen in anderen Swimlanes. Nicht dargestellt sind positive Alternativverläufe, z.B. wenn ein Defekt nicht zu einer Fehlfunktion führt, wenn eine Schwachstelle nicht für einen Angriff ausgenutzt wird. Diese werden bei einer quantitativen Betrachtung berücksichtigt, indem man von einem Zustand zum nächsten nur mit einer gewissen Übergangswahrscheinlichkeit von <100% gelangt.

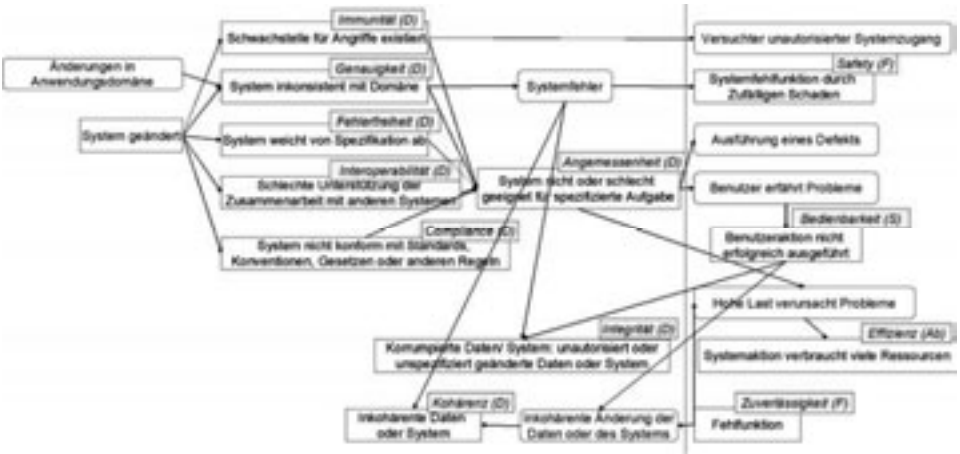


Abbildung 1: Szenario „intended use: software & hardware“; Typ D (Defekte)

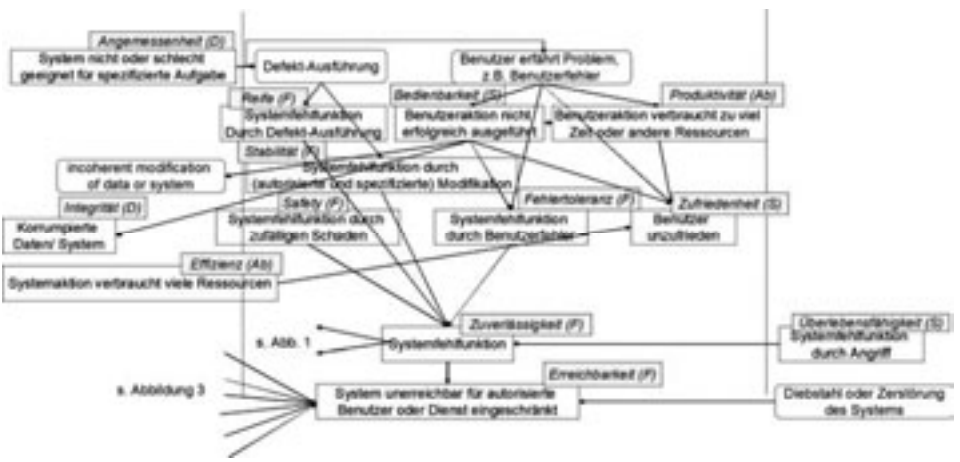


Abbildung 2: Szenario „intended use: user“; Typ F (Fehlfunktionen) und Typ Ab (zu hoher Aufwand oder Kosten während der Benutzung)

Das Aktivitätendiagramm hat vier Swimlanes von links nach rechts: „fault correction, maintenance and enhancement“, „intended use: software & hardware“, „intended use: user“ und „security-relevant misuse“. Diese beschreiben die drei Hauptszenarien, wobei das Szenario „intended use“ der Übersicht halber in die Perspektiven der Software und Hardware einerseits und des Benutzers andererseits aufgeteilt ist. Diese drei Szenarien lassen sich wie folgt beschreiben:

1. **Intended use** (siehe Abbildung 1 und Abbildung 2): Während das System wie vorgesehen benutzt wird (Swimlane “intended use: user”), führen die Defekte (Swimlane “intended use: software and hardware”) zu Fehlfunktionen und Benutzerproblemen (Benutzer im weitesten Sinne, einschließlich Wartungspersonal, Entwicklern, usw.), was zu Defekten in Bezug auf Integrität, Effizienz, Kohärenz oder Zuverlässigkeit führt.
2. **Fault correction, maintenance and enhancement** (siehe Abbildung 3): Diese Aktivitäten führen u.a. dazu, dass das System für autorisierte Benutzer zumindest zeitweise nicht erreichbar ist oder der Dienst eingeschränkt. Außerdem sind dadurch Änderungen am System entstanden, die eventuell neue Defekte enthalten.
3. **Security-relevant misuse** (Abbildung 4): Das Sicherheitsszenario kennt Benutzer, die von der vorgesehenen Benutzung abweichen, während im Szenario “intended use” das System vom vorgesehenen Verhalten abweicht.

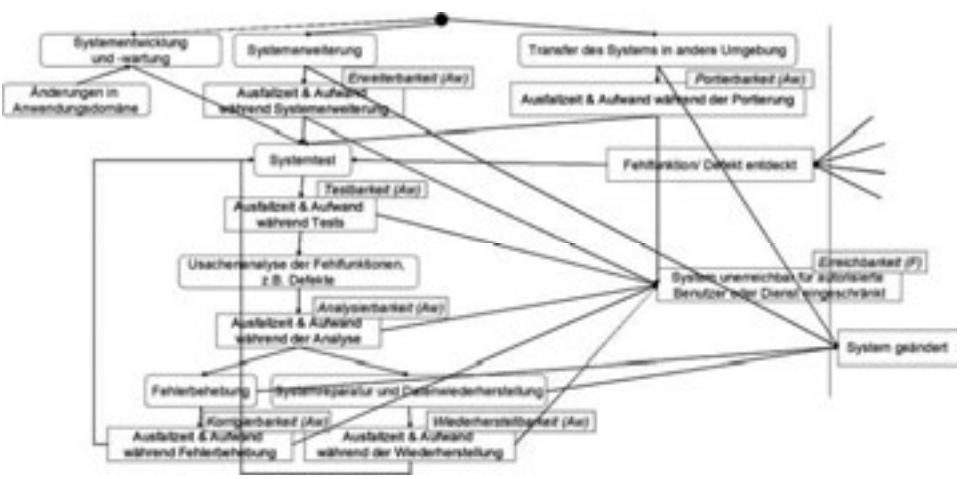


Abbildung 3: Szenario „Fault correction, maintenance and enhancement“, Typ Aw: zu hoher Aufwand oder Kosten während der Wartung

So selbstverständlich die hier dargestellten Abhängigkeiten zwischen unerwünschten Zuständen erscheinen mögen, gab es jedoch bisher keine alle QA umfassende Überlegungen hierzu. Einzelne Arbeiten existieren wie diejenige von Madan et al. [Ma02], welche Übergänge zwischen erwünschten Zuständen in Bezug auf Sicherheit in Zustandsübergangsmodellen darstellt. Singh et al. [SCC01] beschreiben erwünschte Zustände in Sequenzdiagrammen, um Zuverlässigkeitsabweichungen zu analysieren.

Die Granularität der QA und der damit zusammen hängenden unerwünschten Zustände muss jeweils entsprechend dem gewünschten Zweck gewählt werden. Beispielsweise kann man sich vorstellen, Portabilität noch weiter aufzuschlüsseln nach den Aktivitäten, die zur Portierung gehören.

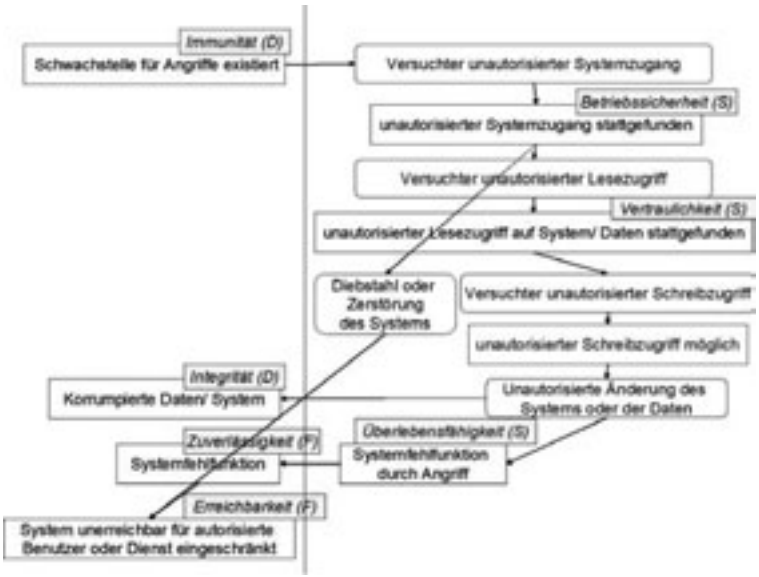


Abbildung 4: Szenario „security-relevant misuse“; Typ S (Sicherheitsverletzungen)

4 Nutzung der Abhängigkeiten

Unsere Motivation für die Herleitung dieser QA-Taxonomie und der Darstellung der Abhängigkeiten zwischen unerwünschten Zuständen bestand darin, dass wir Risiken (d.h. Wahrscheinlichkeit und Schaden) unerwünschter Zustände während des Requirements Engineerings (d.h. vor der Implementierung) quantifizieren wollten. Diese dienen uns als Grundlage für die Priorisierung von Software-Anforderungen, welche Qualitätsrisiken verringern [He07a]. Diese Vorhersage hatte sich in unserer Erfahrung als schwierig erwiesen. Existierende Vorhersagemethoden von Qualitätseigenschaften bzw. Qualitätsrisiken setzen zumeist eine Messung an einem existierenden System voraus, das zumindest ähnlich demjenigen ist, das man betrachtet [He07b].

Die Kenntnis der Abhängigkeiten zwischen unerwünschten Zuständen ersetzt zwar keine Messung, stellt aber ein Rahmenwerk dar, das es erlaubt, von der Erfüllung einer NFA auf eine andere zu schließen. Singh et al. [SCC01] beispielsweise ergänzen UML Use Case und Sequenzdiagramme mit Auftretenswahrscheinlichkeiten und Übergangswahrscheinlichkeiten und verwandeln so diese Diagramme in Bayes-Netzwerke. Dasselbe möchten wir auch mit unseren Aktivitätsdiagrammen tun, um ein System zu analysieren. Wenn unerwünschte Zustände aufeinander folgen, hängen auch deren Wahrscheinlichkeiten voneinander ab. So berechnet sich beispielsweise in einem vereinfachten Modell, in dem ein System entweder erreichbar ist oder aufgrund einer Fehlfunktion in Reparatur, die Wahrscheinlichkeit der Erreichbarkeit dieses Systems näherungsweise als $\rho/(\rho + \lambda)$ aus der Wahrscheinlichkeit λ einer Fehlfunktion und der Reparaturrate ρ, d.h. der Anzahl an Fehlfunktionen, die pro Zeiteinheit repariert werden [Vo00]. Ob analoge Abhängigkeiten für die Risiko- und Qualitätsvorhersage für alle QA verwendet werden können oder ob dies wegen der vielfältigen Abhängigkeiten zu kompliziert wird, wäre noch zu untersuchen. Gesucht werden dann: die Häufigkeiten der unerwünschten Zustände, die Übergangswahrscheinlichkeiten zwischen diesen sowie der jeweils zu erwartende Schaden.

Solche Werte stammen von Messungen ähnlichen Systemen oder an Teilsystemen. Die Leichtigkeit, mit der man diese Größen messen, schätzen oder Erfahrungswerte wieder verwenden kann, unterscheidet sich vermutlich für verschiedene QA. Daher ist es nützlich, wenn man von bekannten Werten auf unbekannte schließen kann, so dass sich die Menge der nötigen Werte verringert. Übergangswahrscheinlichkeiten zwischen Qualitätseigenschaften werden z.B. von [DK06] quantifiziert, die empirisch messen, wie in einem konkreten System die Produktivität, Safety und Zufriedenheit u.a. von der Bedienbarkeit, Angemessenheit und Genauigkeit abhängen. Bei diesem Experiment wurden Paare von QA identifiziert, die am stärksten miteinander korrelieren. Diese entsprechen in unserem Modell unerwünschten Zuständen, die direkt aufeinander folgen. Diese Übereinstimmung unabhängig gefundener empirischer Ergebnisse mit den Vorhersagen unseres durch Literaturrecherchen entwickelten Modells ist ermutigend.

Vorstellbar wäre auch, dass die Kenntnis der Abhängigkeiten zwischen den unerwünschten Zuständen noch auf folgende Weisen beim Requirements Engineering nützlich ist: Besitzt man Schätzwerte über die Wahrscheinlichkeit einiger unerwünschter Zustände, kann man auch realistische Vorgaben für die NFA davon abhängiger QA machen; die Spezifikation nur einer minimalen Menge an NFA wird unterstützt, wenn von direkt voneinander abhängigen QA nur eines betrachtet wird; Impakt-Analyse, d.h. Analyse der Auswirkungen, wenn eine bestimmte Qualitätseigenschaft geändert wird.

5 Zusammenfassung

In diesem Beitrag wird ein Ansatz aufgezeigt, der die Vielfalt der QA strukturiert und Abhängigkeiten zwischen ihnen darstellt, indem man jedem QA einen unerwünschten Zustand zuordnet und deren Abhängigkeiten darstellt.

Wie gut die quantitative Vorhersage von Qualitätsrisiken auf der Grundlage der in Abschnitt 3 dargestellten Abhängigkeiten funktioniert, muss noch untersucht werden. Eine Sammlung von Erfahrungswerten über Häufigkeiten und Schaden unerwünschter Zustände sowie von Übergangswahrscheinlichkeiten an einem bestimmten System wäre ein erster Schritt.

Literaturverzeichnis

[DK06] Dörr, J.; Kerkow, D.: Total control of User Experience in Software Development – a Software Engineering dream? In: Proc. 2nd COST294-MAUSE Int. Open Workshop, 2006; S. 94-99.

[Fi03] Firesmith, D.G.: Analyzing and Specifying Reusable Security Requirements. In: Proc. Requirements for High Assurance Systems Workshop RHAS, Monterey, 2003.

[He07a] Herrmann, A.: Priorisierung von Qualitätsanforderungen auf der Basis von Risikoabschätzungen. Software & Systems Quality Conf. Int., Düsseldorf, 2007.

[He07b] Herrmann, A.: Von Risiken zu Nutzen: Vorhersage von Risiken durch Schätzmethoden. Metrikon, Kaiserslautern, Germany, 2007, S. 179-199.

[IS91] International Standards Organization ISO: International Standard ISO/IEC 9126, Information technology - Software product evaluation - Quality characteristics and guidelines for their use. ISO, International Electrotechnical Commission, Geneva, 1991.

[Ma02] Madan, B.B.; Goševa-Popstojanova, K.; Vaidyanathan, K.; Trivedi, K.S.: Modeling and Quantification of Security Attributes of Software Systems. In: Proc. Int. Conf. On Dependable Systems and Networks DSN, June 2002.

[Oh84] Ohba, M.: Software Reliability Analysis Models. In: IBM J. of Res. and Development 28 (4) 1984; S. 428-443.

[SCC01] Singh, H.; Cortellessa, V.; Cukic, B.; Gunel, E.; Bharadwaj V.: A bayesian approach to reliability prediction and assessment of component based systems. In: Proc. 12th Int. Symposium on Software Reliability Engineering (ISSRE'01), 2001.

[SO00] Sindre, G.; Opdahl, A.L.: Eliciting Security Requirements by Misuse Cases. In: Proc. TOOLS Pacific 2000; S. 120-131.

[SO01] Sindre, G.; Opdahl, A.L.: Templates for Misuse Case Description. In: Proc. REFSQ - Workshop on Requirements Engineering for Software Quality, Foundations for Software Quality, Essener Informatik Beiträge Bd.6, Essen/ Germany, 2001; S. 125-136.

[Vo00] Vouk, M.A.: Software Reliability Engineering: In: 2000 Annual Reliability and Maintainability Symposium.

Erfassung, Strukturierung und Überprüfung von Qualitätsanforderungen durch aktivitätenbasierte Qualitätsmodelle

Stefan Wagner, Florian Deissenboeck, Sebastian Winter

Institut für Informatik
Technische Universität München
wagnerst@in.tum.de
deissenb@in.tum.de
winterse@in.tum.de

Abstract: Die Behandlung von Qualitätsanforderungen ist ein zentrales Thema bei der Entwicklung von Software-Systemen. Aufgrund der Komplexität und Vielschichtigkeit der Thematik „Software-Qualität", stellt bereits die geeignete Formulierung dieser Anforderungen eine Herausforderung dar. Qualitätsmodelle haben sich hierbei als ein vielsprechender Ansatz zur Beherrschung der Komplexität erwiesen. Eine neue Art von Qualitätsmodellen verwendet die Aktivitäten, die auf und mit dem System ausgeführt werden, als eine explizite Dimension zur Strukturierung von Qualitätsmodellen. Dieses Papier beschreibt die Verwendung von Qualitätsmodellen dieser Art zur Erfassung, Verfeinerung und Überprüfung von Qualitätsanforderungen.

1 Qualitätsanforderungen

Qualitätsanforderungen werden üblicherweise als Teil der *nichtfunktionalen* Anforderungen an ein System gesehen. Diese nichtfunktionalen Anforderungen beschreiben Eigenschaften eines Systems, die nicht zur primären Funktionalität gehören. Obwohl die Begrifflichkeit „nichtfunktionale Anforderungen" teilweise umstritten ist, existieren immer Anforderungen, die zu spezifischen Qualitäten eines Systems Bezug nehmen [Gli05]. Diese Art von Anforderungen werden *Qualitätsanforderungen* genannt.

Qualitätsanforderungen werden in der Anforderungsanalyse von Softwaresystemen oft nur unbefriedigend behandelt. Ein Hauptgrund dafür ist, dass diese Anforderungen grundsätzlich schwieriger in einer messbaren Art und Weise ausdrückbar sind, was sie wiederum schwieriger analysierbar macht [NE00]. Zugrunde liegt das Problem, dass Qualität selbst ein komplexes und vielschichtiges Konzept ist [Gar84] und deren Ausmaß oder Existenz schwierig zu analysieren ist. Dadurch wird auch die Definition von Qualitätsanforderungen eine anspruchsvolle Aufgabe. Besonders die vollständige Integration der verschiedenen Aspekte aller Stakeholder ist aufwändig und fehleranfällig. Das Problem besteht also darin, wie Qualitätsanforderungen in einer strukturierten und vollständigen Art und Weise erfasst und überprüft werden können.

Es wird ein fünfstufiges Vorgehen zur Behandlung von Qualitätsanforderungen unter Nutzung eines aktivitätenbasierten Qualitätsmodells [DWP+07b] vorgeschlagen. Das Qualitätsmodell verwendet Aktivitäten als eine explizite Dimension und beschreibt die Einflüsse der Systementitäten und ihrer Attribute auf diese Aktivitäten. Diese beiden Dimensionen können zweckdienlich als Strukturierungsmittel für Qualitätsanforderungen verwandt werden, da sie eine abstrakte Sicht auf Qualitätsanforderungen bieten. Ihre Verfeinerung kann durch eine Analyse der Systementitäten und durch welche Aktivitäten diese beeinflusst sind erfolgen. Schließlich ist dadurch eine direkte Nachverfolgbarkeit von der Qualitätssicherung zurück zu den Qualitätsanforderungen durch das Qualitätsmodell gegeben, da das Modell wiederum als Basis für Qualitätssicherungsmaßnahmen, wie Reviews, verwendet werden kann.

2 Qualitätsmodellierung

Um Qualitätsanforderungen effizient handhaben zu können, braucht man eine Möglichkeit, sie in einer präzisen und konsistenten Art und Weise ausdrücken zu können. Seit den 1970ern wurden eine Vielzahl von Qualitätsmodellen entwickelt, die das Ziel hatten genau dies zu erreichen [BBK+78, ISO03]. Jedoch wird in [DWP+07b] argumentiert, dass diese Ansätze eine Reihe von Schwachpunkten besitzen. Am wichtigsten ist hierbei, dass sie es versäumen, die Zusammenhänge zwischen Systemeigenschaften und den auf und mit dem System durch die Stakeholder ausgeführten Aktivitäten mit in Betracht zu ziehen. Diese Auslassung der Aktivitäten ist ein ernstes Problem, da sie den größten Teil der Gesamtkosten von Software bestimmen. Zusätzlich erlauben gerade die Aktivitäten ein natürliches Kriterium zur Dekomposition des komplexen Konzepts *Qualität*, das anderen Ansätzen fehlt.

Um diesen Problemen zu begegnen, wurde eine konsequente Unterscheidung von Aktivitäten und Systementitäten vorgeschlagen [DWP+07b]. Diese Unterscheidung unterstützt die Identifizierung von fundierten Qualitätskriterien und erlaubt über deren Zusammenhänge Schlussfolgerungen zu treffen. Zur Illustration verwenden wir im Folgenden das Qualitätsattribut *Wartbarkeit*, das bekannterweise einen starken Einfluss auf die Gesamtkosten im Lebenszyklus eines Softwaresystems hat und darüber hinaus oft als eines der vielfältigsten, am schwierigsten zu beherrschenden, Qualitätsattribute wahrgenommen wird.

Die erste Dimension des Qualitätsmodells besteht aus den von den Stakeholdern auf und mit dem System ausgeführten Aktivitäten. Im Falle von Wartbarkeit hängt die Menge der relevanten Aktivitäten vom jeweiligen Entwicklungs- und Wartungsprozess der Organisation ab. Da Aktivitäten einfach in Teilaktivitäten strukturiert werden können, formt diese erste Dimension einen Baum: den *Aktivitätenbaum*.

Die zweite Dimension des Modells, der *Entitätenbaum*, beschreibt die Dekomposition der *Entitäten*, vor allem des Softwaresystems. Um Wartbarkeit in einem gegebenen Projekt messen zu können, müssen Verbindungen zwischen den Entitäten und Aktivitäten etabliert werden. Dieser Zusammenhang kann durch eine Matrix wie in Abb. 1 dargestellt werden.

Wie die Abbildung zeigt, muss man die Entitäten mit grundlegenden *Attributen* wie *Kon-*

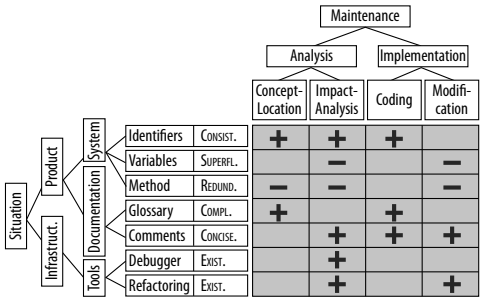


Abbildung 1: Wartbarkeitsmodell

sistenz, Vollständigkeit, Prägnanz oder *Redundanz* ausstatten, um die Zusammenhänge ausdrücken zu können. Diese Zusammenhänge werden nun durch die Identifizierung von *Einflüssen* dargestellt. Ein Einfluss ist definiert als die Relation zwischen einem Tupel bestehend aus Entität und Aktivität und einer Aktivität, wobei das Zeichen „+“ einen positiven und „−“ einen negativen Einfluss darstellt.

Ein Beispiel in obiger Abbildung ist die Prägnanz von Bezeichnern im Programmtext, die einen positiven Einfluss auf das Auffinden von Konzepten im Code hat oder der negative Einfluss von überflüssigen Variablen auf die Änderung existierenden Codes. Das Beispiel zeigt, dass sogar das offensichtlich einfache Attribute *Existenz* sehr mächtig sein kann, wenn gezeigt werden soll, dass eine passende Infrastruktur, wie zum Beispiel ein Debugger, einen Einfluss auf bestimmte Aktivitäten hat.

Solch ein Modell ist gut geeignet, um Qualitätsanforderungen handhabbar zu machen, da die Aktivitäten eine direkte Verbindung zwischen den Stakeholdern, die schließlich Qualitätsanforderungen definieren, und dem Software-System liefert. Im Beispiel der Wartbarkeit ist der betroffene Stakeholder ein Entwickler. Seine Hauptaktivität *Wartung* kann in handhabbarere Teilaktivitäten heruntergebrochen werden. Diese Teilaktivitäten können dann durch einfache Attribute den Systementitäten zugeordnet werden. In [WWD08] wurde gezeigt, wie solch ein Modell zur Beschreibung der *Gebrauchstauglichkeit* angewandt werden kann. Dort ist der zentrale Stakeholder der Nutzer der Software und seine Hauptaktivität *Nutzung* kann wiederum leicht in spezifischere Subaktivitäten aufgeteilt werden. Diese können dann zu konkreten Entitäten, wie den Schriftarten in der Nutzerschnittstelle, zugeordnet werden.

3 Erfassung und Verfeinerung

Die Hauptansätze zur Erfassung von Qualitätsanforderungen bestehen entweder aus dem Abarbeitung verschiedener Anforderungstypen und der abgeleiteten Erstellungen von Prototypen [RR99] oder der Verwendung von positiven und negativen Szenarios [Ale03]. Beide Ansätze haben jeweils ihre Vorteile und sollten am besten in Kombination eingesetzt werden. Jedoch kann die zusätzliche Einbindung von Qualitätsmodellen, wie sie im Abschnitt 2 dargestellt wurden, die Ergebnisse durch eine verfeinerte Struktur verbessern.

Ebert klassifiziert in [Ebe97] Anforderungen in *benutzerorientert* und *entwicklungsorientiert*, was bereits in Richtung einer Strukturierung nach Aktivitäten geht. Diese Unterscheidung ist aber noch zu grob. Doerr et al. verwenden in [DKK+05] bereits Qualitätsmodelle in für nichtfunktionale Anforderungen. Die verwendeten Modelle liefern aber keine direkte Verbindung zu Aktivitäten und damit auch keine direkte Verbindung zu Stakeholdern.

Die Struktur, die durch das aktivitätenbasierte Qualitätsmodell induziert wird, bietet die Grundlage, um Qualitätsanforderungen zu erfassen und zu verfeinern. Der Erfassungs- und Verfeinerungsprozess besteht aus fünf Hauptschritten, die von den etablierten Erfassungstechniken unterstützt werden sollten. Einen Überblick verschafft Abb. 2. Die Schritte sind stark an der Verwendung der beiden Bäume im Qualitätsmodell orientiert und haben das Ziel – soweit möglich – strukturiert bis zu quantitativen Werten für die Anforderungen zu gelangen. Natürlich hängt der Ansatz von der Verfügbarkeit eines geeigneten Qualitätsmodells ab. Idealerweise exisitiert bereits ein solches, aber es kann auch parallel zu den Anforderungen entwickelt werden. Die oberen Ebenen der Bäume lassen sich erfahrungsgemäß aus anderen Qualitätsmodellen der Literatur [DWP+07b] wiederverwenden.

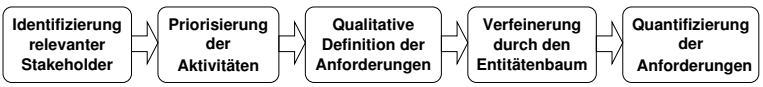


Abbildung 2: Die Prozessschritte zur Erfassung und Verfeinerung von Qualitätsanforderungen

Identifizierung relevanter Stakeholder Der erste Schritt ist, in Anlehnung an andere Ansätze, die Identifikation aller Stakeholder des Software-Systems. Für Qualitätsanforderungen sind dies üblicherweise die Nutzer, Entwickler und Warter, Operateure und Nutzertrainer. Weitere Stakeholder können natürlich, abhängig vom konkreten Projekt, zusätzlich relevant sein. Nach dieser Identifzierung kann das Qualitätsmodell zur Ableitung ihrer jeweiligen Aktivitäten verwendet werden. Beispielsweise gehören *Konzeptlokalisierung*, *Auswirkungsanalyse*, *Kodierung* oder *Änderung* zu den Aktivitäten des Wartungsingenieurs [DWP+07b]. Insbesondere die spezifischen Aktivitäten des Nutzers können durch zusätzliche Nutzungsszenarien detailiert werden.

Priorisierung der Aktivitäten Im nächsten Schritt werden die Aktivitäten der relevanten Stakeholder im Bezug auf ihre Wichtigkeit priorisiert. Falls nicht alle notwendigen Aktivitäten im Modell definiert sind, wird es entsprechend erweitert. Dies resultiert in einer Liste aller Aktivitäten der relevanten Stakeholder, sortiert in der Reihenfolge der Wichtigkeit. „Wichtigkeit“ meint hierbei die Häufigkeit der Ausführung und die Aufwändigkeit der Aktivitäten. Die Rechtfertigung der Wichtigkeit kann Expertenmeinung oder Erfahrungen aus ähnlichen Projekten sein. Diese Liste wird in den folgenden Schritten zur Fokusierung der Definition und Verfeinerung der Anforderungen verwendet.

Qualitative Definition der Anforderungen Im nächsten Schritt muss entschieden werden, wie stark die unterschiedlichen Aktivitäten unterstützt werden sollen. Die Antworten stellen schließlich qualitative Anforderungen and das Software-System dar. Zum Beispiel

kann es gewünscht sein, dass die Aktivität *Konzeptlokalisierung* einfach durchzuführen ist, da die Problemdomäne der Software viele komplexe Konzepte enthält. Abhängig von der Zahl der Aktivitäten, die berücksichtigt werden müssen, können die Aktivitäten mit der geringsten Priorität ignoriert werden.

Verfeinerung durch den Entitätenbaum Wie in Abschnitt 2 enthält der Entitätenbaum die Entitäten der Software, der Umgebung und des Entwicklungsprozesses, die in irgendeiner Form relevant für die Software-Qualität sind. Der Baum organisiert sie in einer definierten, hierarchischen Weise, die ideal zur Verfeinerung der auf Basis der Aktivitäten erfassten Anforderungen geeignet ist. Das Qualitätsmodell selbst hilft hierbei, da es in den Einflüssen den Zusammenhang zwischen Entitäten und Aktivitäten dokumentiert. Daher muss zur Identifzierung der Entitäten, die die identifizierten Aktivitäten beeinflussen nur den dokumentierten Einflüssen gefolgt werden. Falls die Einflüsse im aktuellen Modell noch unvollständig sind, kann dieser Schritt wiederum zur Erweiterung des Modells verwendet werden. Dadurch ist es auch bedeutend einfacher Konsistenz mit der späteren Qualitätssicherung herzustellen, da diese auch auf dem Modell basiert. Zur Definition der verfeinerten Qualitätsanforderungen können die definierten Attribute der Entitäten herangezogen werden. Beispielsweise kann eine detailierte Anforderung sein, dass jedes Objekt einen von aussen zugänglichen Zustand besitzen muss, damit dieser im Test überprüft werden kann, was direkt aus einem Modellelement abgeleitet werden kann. Eine Verfeinerung ist vor allem für die Anforderungen erforderlich, die in den vorigen Schritten als wichtig eingestuft wurden.

Quantifizierung der Anforderungen Das endgültige Ziel ist es, die Anforderungen in einer quantitativen und damit überprüfbaren Form vorliegen zu haben. Durch die direkte Zuordnung zu einer Aktivität ist es auch möglich Anforderungen, wie beispielsweise *geeignetet Modularisierung des Systems*, spezifisch zu quantifizieren. Die Quantifizierung kann sowohl auf der Aktivitätenebene als auch auf der Entitätenebene passieren. Auf der Aktivitätenebene wäre das zum Beispiel, dass eine durchschnittliche *Änderungsaktivität* nicht mehr als vier Personenstunden an Aufwand brauchen darf. Anforderungen auf der Entitätenebene können abhängig vom gewählten Attribut quantitativ überprüfbar sein (vgl. [DWP+07b]). Zum Beispiel können *überflüssige Code-Variablen* gezählt und damit ein oberes Limit geprüft werden. Möchte man eine Anforderung, wie beispielsweise die *Angemessenheit der Datenstrukturen*, bewusst nicht quantifizieren, besteht die Möglichkeit dies direkt im Qualitätsmodell zu vermerken. In jedem Fall bietet das Qualitätsmodell dann die Möglichkeit der Überprüfung der Qualitätsanforderungen, da es eine direkte Verbindung zu qualitätssichernden Massnahmen darstellt [DWP+07b].

4 Zusammenfassung

Wir haben bereits in [DWP07a] vorgeschlagen, Qualitätsanforderungen anhand der Kosten und damit der Stakeholder zu klassifizieren. In diesem Papier haben wir diesen Weg konse-

quent fortgesetzt und eine konkrete Methode zur Erhebung und Strukturierung solcher Anforderungen mit Hilfe von aktivitätenbasierte Qualitätsmodellen angegeben. Die verwendeten Qualitätsmodelle stellen einen wiederverwendbaren „Speicher“ an Qualitätswissen dar, der die Erhebung erleichtert und später auch zur Überprüfung verwendet werden kann, da Qualitätssicherungsmaßnahmen direkt auf dem Qualtätsmodell aufbauen können. Wir planen die einzelnen Schritte der Methode noch genauer durch konkrete Vorlagen auszuarbeiten. Diese werden in Fallstudien erprobt und auch die Verbindung zu funktionalen Anforderungen wird hergestellt werden.

Literatur

[Ale03] Ian Alexander. Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, Seiten 58–66, 2003.

[BBK+78] Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. Macleod und Michael J. Merrit. *Characteristics of Software Quality*. North-Holland, 1978.

[DKK+05] Joerg Doerr, Daniel Kerkow, Tom Koenig, Thomas Olsson und Takeshi Suzuki. Non-Functional Requirements in Industry – Three Case Studies Adopting an Experience-based NFR Method. In *Proc. 13th International Conference on Requirements Engineering (RE'05)*, Seiten 373–382. IEEE Computer Society, 2005.

[DWP07a] Florian Deissenboeck, Stefan Wagner und Markus Pizka. Kosten-basierte Klassifikation von Qualitätsanforderungen. In *Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung. Halbtägiger Workshop in Zusammenhang mit der SE Konferenz 2007*, 2007.

[DWP+07b] Florian Deissenboeck, Stefan Wagner, Markus Pizka, Stefan Teuchert und Jean-François Girard. An Activity-Based Quality Model for Maintainability. In *Proc. 23rd International Conference on Software Maintenance (ICSM '07)*. IEEE Computer Society, 2007.

[Ebe97] Christof Ebert. Dealing with Nonfunctional Requirements in Large Software Systems. *Ann. Softw. Eng.*, 3:367–395, 1997.

[Gar84] David A. Garvin. What Does »Product Quality« Really Mean? *MIT Sloan Management Review*, 26(1):25–43, 1984.

[Gli05] Martin Glinz. Rethinking the Notion of Non-Functional Requirements. In *Proc. Third World Congress for Software Quality (3WCSQ)*, Seiten 55–64, 2005.

[ISO03] ISO 9126: Product Quality – Part 1: Quality Model, 2003.

[NE00] Bashar Nuseibeh und Steve Easterbrook. Requirements Engineering: A Roadmap. In *Proc. Conference on the Future of Software Engineering (ICSE '00)*, Seiten 35–46. ACM Press, 2000.

[RR99] Suzanne Robertson und James Robertson. *Mastering the Requirements Process*. ACM Press, Addison-Wesley, 1999.

[WWD08] Sebastian Winter, Stefan Wagner und Florian Deissenboeck. A Comprehensive Model of Usability. In *Proc. Engineering Interactive Systems 2007 (EIS '07)*. Springer, 2008. Zur Veröffentlichung angenommen.

Über die Auswirkungen nicht-funktionaler Anforderungen beim Entwurf eines Rahmenwerkes zur Anbindung von Smart Items an ERP-Systeme

Rosemaria Giesecke, Florin Spanachi

SAP Research
SAP AG
Vincenz-Prießnitz-Str. 1
76131 Karlsruhe
rosemaria.giesecke@sap.com
florin.spanachi@sap.com

Werden Smart Items wie etwa RFID-Tags, Sensornetzwerke oder eingebettete Systeme an ERP-Systeme angebunden, so steht man vor der Aufgabe, mit Datenmengen in bislang ungekannten Größenordnungen umgehen zu müssen. An solch ein Rahmenwerk zur Anbindung von Smart Items werden eine Reihe nicht-funktionaler Anforderungen gestellt wie etwa

- **Änderbarkeit und Übertragbarkeit**
  Die Architektur soll zukunftssicher sein in dem Sinne, dass Meldungen von unterschiedlichsten Quellen und Tag-Arten bearbeitet werden sollen, auch von Geräten, die erst in den nächsten Jahren entwickelt werden.

- **Zuverlässigkeit → Robustheit**
  Das System muss mit sich dynamisch ändernden Mengen unterschiedlichster Geräte (Smart Tags) umgehen können.

- **Übertragbarkeit → Anpassbarkeit**
  Das System soll während der Laufzeit änderbar sein bezüglich der Anzahl und Art der Geräte und Meldungen sowie der Funktionalität zur Vorverarbeitung eintreffender Meldungen (Änderbarkeit → Modifizierbarkeit).

- **Zuverlässigkeit → Konformität**
  Es muss auf Wunsch sichergestellt werden, dass Meldungen nicht verlorengehen.

- **Zuverlässigkeit → Fehlertoleranz**
  Für bestimmte Meldungen muss sichergestellt werden, dass die Zeit zum Bearbeiten und Weiterleiten ein vorgegebenes Maximum nicht überschreitet.

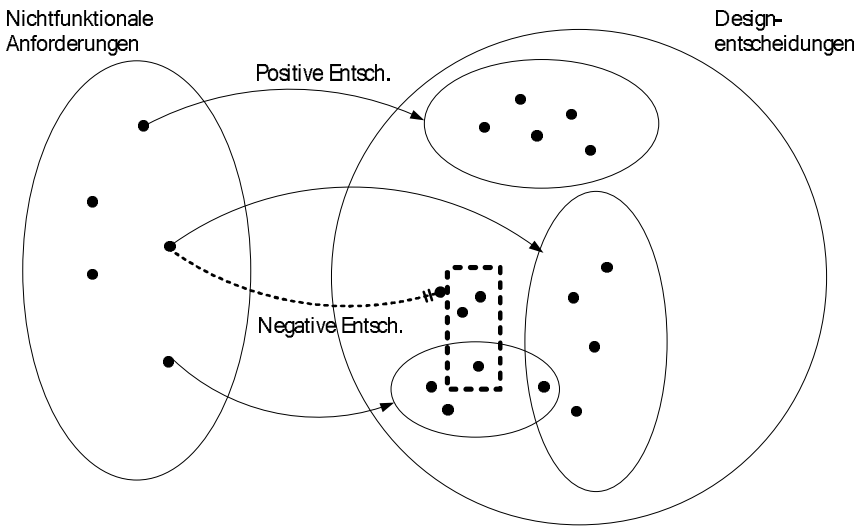
- **Übertragbarkeit → Anpassbarkeit**
  Das System soll dynamisch hoch skalierbar sein. Ein System, das wenige Geräte anbindet, soll einen kleinen Footprint haben, während Systeme für viele Geräte durchaus auf Clustern mit mehreren Servern laufen können.

- **Übertragbarkeit → Anpassbarkeit**
  Das System soll für die unterschiedlichsten Anwendungen von Smart Items hochgradig konfigurierbar sein.

- **Übertragbarkeit → Anpassbarkeit**
  Das System soll für Kunden offen sein in dem Sinne, dass Benutzerschnittstellen für unterschiedliche Anwendungen vom Kunden oder von Drittanbietern realisiert und genutzt werden können.

In diesem Beitrag wird über die Auswirkungen dieser nicht-funktionalen Anforderungen auf den Entwurf der Funktion und Architektur eines Rahmenwerkes berichtet. Es soll diskutiert werden, ob sich möglicherweise daraus Muster erkennen lassen zur Umsetzung einiger nicht-funktionaler Anforderungen.

Bei der Umsetzung von Anforderungen in einen Systementwurf gibt es für viele nicht-funktionale Anforderungen jeweils Design-Entscheidungen die zu deren Erfüllung beitragen und solche, die deren Erfüllung entgegenstehen. Die Idee ist nun, für möglichst viele Kategorien nicht-funktionaler Anforderungen solche positiven und negativen Designetscheidungen zu identifizieren, zu dokumentieren und anhand möglichst vieler Anwendungsbeispiele zu verifizieren.



Das Ergebnis eines solchen Prozesses gibt dem Designer ein Hilfsmittel an die Hand, sicherer und schneller im Entwurf von Systemen zu werden.

Nach wie vor ist u.a. die Frage offen, wie man mit Abhängigkeiten nicht-funktionaler Anforderungen untereinander umgehen kann und ob es dafür stets Lösungen gibt. Falls nicht, kann man solche Fälle dann mit dem oben erstellten Hilfsmittel einfacher aufdecken?

Es wäre zudem interessant die ISO-Kategorisierung nicht-funktionaler Anwendungen zu untersuchen ob sie für das oben beschriebene Verfahren in der existierenden Konstellation geeignet wäre oder nicht.

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-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

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