# Odysseus: Ein Framework für maßgeschneiderte Datenstrommanagementsysteme

André Bolles, Marco Grawunder, Jonas Jacobi, Daniela Nicklas, H.-Jürgen Appelrath

[andre.bolles|marco.grawunder|jonas.jacobi|daniela.nicklas|appelrath]@uni-oldenburg.de

**Abstract:** In Anwendungssystemen, in denen kontinuierlich große Datenmengen auftreten, ist es häufig aus Performanz- und Speicherplatzgründen nicht möglich, diese Daten vor der Verarbeitung zu speichern. Daher müssen die Daten strombasiert verarbeitet werden. Im jungen Forschungsfeld des Datenstrommanagements sind bereits eine Reihe prototypischer Datenstrommanagementsysteme (DSMS) entstanden, in denen das Informationsbedürfnis, ähnlich wie in traditionellen Datenbankmanagementsystemen (DBMS), über deklarative Anfragen formuliert wird. Diese Prototypen sind in der Regel jedoch nur schwer an neue Anwendungsfelder anpassbar. In dieser Arbeit stellen wir mit Odysseus ein Framework vor, mit dem maßgeschneiderte DSMS für verschiedene Anwendungsfelder entwickelt werden können. Wir stellen die Architektur des Systems vor und beschreiben, wie einzelne Komponenten des Systems erweitert oder angepasst werden können, so dass bspw. die Verarbeitung neuer Datenmodelle oder Verarbeitungsansätze für Stromdaten evaluiert werden können.

## 1 Einleitung

Viele kontextbezogene Anwendungen arbeiten mit aktuellen Informationen aus der realen Welt, die häufig über Sensoren, z. B. bei der Überwachung oder Steuerung dynamsicher Systeme, hochaktuell gewonnen werden. Bisher werden solche Probleme in der Praxis meist mit Hard- oder Softwaresystemen gelöst, in welche die Verarbeitungslogik fest „einprogrammiert“ ist. Mit zunehmender Komplexität der Systeme und höherer Änderungsrate in den Anforderungen an die Systeme werden diese Lösungen jedoch zunehmend unbrauchbar und erfordern ein flexibleres Datenmanagement.

In dieser Domäne ist es häufig aus Performanz- und Speicherplatzgründen nicht möglich, die Daten vor der Verarbeitung z. B. in einem Datenbankmanagementsystem (DBMS) zu speichern. Stattdessen müssen die Daten strombasiert verarbeitet und ggfs. später archiviert werden. Im Bereich der Datenstrommanagementsysteme (DSMS) sind neben zahlreichen Forschungsarbeiten mittlerweile erste kommerzielle Systeme verfügbar. Analog zu einem DBMS ist die Kernidee dabei, das Informationsbedürfnis einer Anwendung nicht durch ein Programm zu kodieren, sondern in einer Anfrage zu deklarieren, die dann von dem DSMS übersetzt, optimiert und ausgeführt wird. Im Unterschied zu einem DBMS sind die Anfragen jedoch meist kontinuierlich. Das heißt, sie werden registriert und über einen längeren Zeitraum hinweg auf den Eingangsdatenströmen verarbeitet. Das Ergebnis einer Anfrage auf Datenströmen ist in der Regel auch wieder ein Datenstrom.

In diesem Beitrag präsentieren wir Odysseus, unser Framework zur Erstellung von DSMS. Motiviert durch verschiedene Anwendungsszenarien (Kapitel 2), die, wie es auch in [Sc05] beschrieben wird, aufgrund unterschiedlicher Anforderung maßgeschneiderte Lösungen benötigen, beleuchten wir zunächst verwandte Arbeiten in Kapitel 3. Daraufhin stellen wir die modulare Odysseus-Architektur vor (Kapitel 4). Sie erlaubt es, verschiedene Themen der Datenstromverarbeitung unabhängig voneinander zu erforschen, wie z.B. die Verarbeitung verschiedener Datenmodelle, die Optimierung der Ausführung oder die Integration domänenspezifischer Operatoren. Wir schließen den Beitrag mit einem Ausblick in Kapitel 5.

## 2 Anwendungsszenarien

Datenströme spielen insbesondere in Bereichen eine Rolle, in denen Daten kontinuierlich anfallen und in denen eine laufende Überwachung bestimmter Situationen erforderlich ist. Als klassisches Beispiel für eine Datenstromanwendung wird häufig die Überwachung von Börsentransaktionen genannt [ACc+03], wo es darum geht möglichst schnell und gezielt auf bestimmte Kursänderungen durch An- und Verkauf von Wertpapieren zu reagieren. Es gibt allerdings eine Vielzahl anderer Anwendungsgebiete für Datenstrommanagement. Hierzu zählen z. B. die Überwachung dezentraler Energieversorgungsanlagen in SCADA-Systemen und die Sensordatenfusion für Fahrerassistenzsysteme in Fahrzeugen. Da Odysseus in diesen beiden Szenarien eingesetzt wird, stellen wir sie im Folgenden kurz vor.

**Dezentrales Energiemanagement**: Die Energiebranche befindet sich unter anderem durch die zunehmende Verbreitung regenerativer Energieerzeugung mit geographisch und organisatorisch immer mehr und weiter verteilten Anlagen und Stromnetzen in einem Wandel. Für die Überwachung und Steuerung werden sogenannte Supervisory Control and Data Aquisition (SCADA) Systeme eingesetzt. Dies sind typischerweise hartkodierte Programme, die über Jahrzehnte hinweg entstanden sind. Aufgrund des Wandels in der Energieversorgung müssen diese SCADA Programme nun immer schneller neuen und wachsenden Anforderungen genügen - so wächst zum Beispiel die Anzahl von eingehenden Ereignissen allein aufgrund eines breiteren Sensorikeinsatzes in neueren Anlagen um ein bis zwei Größenordnungen. Die notwendigen Anpassungen an bestehender Software sind häufig mit sehr hohem Aufwand verbunden. Aus diesem Grund beschäftigen wir uns mit dem alternativen Einsatz von DSMS auf der Empfangsseite von SCADA-Systemen. DSMS besitzen gegenüber hartkodierten Programmen den Vorteil, dass sie sehr einfach zu erweitern sind, da neue Verarbeitungslogik einfach zur Laufzeit in Form von kontinuierlichen Anfragen hinzugefügt werden kann. Sie sind außerdem von vornherein für die Verarbeitung der zu erwartenden Datenmengen/-raten ausgelegt und können Optimierungen über viele gleichzeitig laufende Anfragen hinweg durchführen. Damit DSMS in einem SCADA System eingesetzt werden und ihre Stärken dort ausspielen können, müssen sie natürlich die gleichen Anforderungen (insbesondere bezüglich Hochverfügbarkeit, Performanz etc.) erfüllen wie die bisherige Software. Verfahren zur Erfüllung dieser Anforderungen in DSMS werden mit Odysseus evaluiert.

**Sensordatenfusion für Fahrerassistenzsysteme**: Zur Unterstützung des Fahrers bei der Aufgabe der Fahrzeugführung werden moderne Fahrzeuge mit einer Vielzahl neuer Assistenzfunktionen ausgestattet. Insbesondere aktive Fahrerassistenzsysteme, die in das Fahrverhalten des Fahrzeugs eingreifen, gewinnen hierbei an Bedeutung. Adaptive Cruise Control (ACC) Systeme regeln beispielsweise selbstständig die Geschwindigkeit in Abhängigkeit der Distanz zu vorausfahrenden Fahrzeugen. Im Vergleich zu einfachen Assistenzfunktionen, zu denen man auch einen Tachometer zählen kann, benötigen komplexere Assistenzfunktionen jedoch Informationen über das Fahrzeugumfeld. Hierzu sind insbesondere neuere Fahrzeuge mit einer Vielzahl von Sensoren ausgestattet, die das Fahrzeugumfeld abtasten. Dies können Radarsensoren, Laserscanner, Ultraschallsensoren oder aber auch Videokameras sein. Alleine ist jedoch keine dieser Sensortechnologien in der Lage, ein ausreichendes Modell der Umgebgung zu liefern, so dass ein Gesamtbild aus den Sensordaten verschiedener Sensoren gewonnen werden muss (Sensordatenfusion). So werden beispielsweise mehrere unterschiedlich ausgerichtete Radarsensoren eingesetzt, um den Abtastbereich um das Fahrzeug zu vergrößern. Zusätzlich können Laserscanner eingesetzt werden, um vorausfahrende Fahrzeuge zu vermessen, was mit Radarsensoren nicht ohne weiteres möglich ist. Bisherige Sensorfusionsysteme, nicht nur im Bereich der Fahrerassistenzsysteme, werden ähnlich zu SCADA-Systemen meist als C-Programme realisiert (vgl. [lCPRK08, Föl06, NLRR02] u. a.). Damit ist eine Anpassung dieser Systeme an neue Sensoren oder Sensorkonfigurationen sowie an neue Anforderungen von Fahrerassistenzsystemen sehr aufwendig. Deshalb wird in aktuellen Forschungsarbeiten Odysseus dazu eingesetzt, Sensordatenfusion durch die Verarbeitung deklarativer Anfragen in einem DSMS zu realisieren. Hierzu werden in Odysseus geeignete Verfarhren integriert, die eine Fusionierung von Sensordaten ermöglichen.

## 3 Verwandte Arbeiten

Ein zentraler Unterschied zwischen der Verarbeitung von Daten aus Datenbanken und Stromdaten liegt darin, dass ein Datenstrom potenziell unendlich sein kann. Aus diesem Grund muss inbesondere bei Operationen, welche mehrere Elemente aus einem oder verschiedenen Datenströmen miteinander vergleichen, auf ein nicht-blockierendes Verhalten geachtet werden. Um Blockierungen sowie Speicherplatzprobleme zu vermeiden, wird im Kontext von Datenströmen in der Regel anhand bestimmter Attributwerte der (zeitliche) Fortschritt eines Datenstroms beobachtet. Oft besitzen Sensordaten einen Zeitstempel, der anzeigt zu welchem Zeitpunkt ein Datum aufgezeichnet wurde. Dieser Zeitstempel kann dann dazu genutzt werden, Elemente eines Datenstroms in der richtigen Reihenfolge auszuwerten. Weiterhin wird zur Vermeidung von Speicherplatzproblemen die Annahme getroffen, dass in der Regel nur aktuelle Daten relevant sind und weniger aktuelle Daten daher nicht mehr für weitere Berechnungen betrachtet werden müssen. Die relevanten Ausschnitte eines Datenstroms werden durch sogenannte Fenster bestimmt. Die gängiste Methode sind Zeitfenster (vgl. u. a. [ACc$^+$03]), mit denen ausgedrückt werden kann, dass ein Datum in einem Datenstrom nur eine gewisse Zeit gültig sein soll. Andere Fensteransätze betrachten jeweils die letzten x Tupel als gültig (Tupelfenster) oder

bestimmen den Bereich der gültigen Daten in einem Datenstrom anhand der Daten selbst (Prädikatfenster) [GAE06]. Letztlich dienen Fenster dazu, die Gültigkeit von Elementen eines Datenstroms zu definieren und somit das entfernen ungültiger Elemente aus den Zustandsspeichern einzelner Operatoren, wie bspw. einem Join zu entfernen. Die Gültigkeit wird meist entweder durch ein Zeitintervall ([KS05] u. a.) oder durch einen positiven (Start der Gültigkeit) und negativen Marker (Ende der Gültigkeit), sog. Positiv/Negativ-Ansatz (PN-Ansatz) [AAB+05], repräsentiert.

Auch wenn es sich beim Datenstrommanagement um ein relativ junges Forschungsfeld handelt, so sind dort dennoch bereits einige Arbeiten zu DSMS entstanden. Zu den ersten Prototypen für DSMS zählen Aurora [ACc+03], Borealis [AAB+05], TelegraphCQ [CCD+03] und STREAM [ABB+03]. Neben diesen amerikanischen Projekten wurde jedoch auch sehr früh im europäischen Raum an DSMS geforscht. Die ETH Zürich erforscht bspw. mit die Verarbeitung von XML-Datenströmen [BFF+07]. Mit dem PIPES-Projekt der Universität Marburg wurde unter anderem eine vollständige semantische Definition der relationalen Algebra auf Datenströmen entworfen und umgesetzt [KS05]. Auch IBM entwickelt mit SystemS [GAW+08] ein DSMS.

Das in dieser Arbeit vorgestellte DSM-Framework Odysseus unterscheidet sich von den zuvor genannten System dadurch, dass es im Kern kein konkretes DSMS darstellt, sondern eine Architektur bereitstellt, mit der flexibel DSMS für verschiedene Anwendungsszenarien entwickelt werden können. Zwar kann bspw. auch PIPES erweitert werden, jedoch ist dieses System bspw. auf relationale Datenströme beschränkt. Mit Odysseus können unterschiedliche Verarbeitungsmechanismen, die auch auf unterschiedlichen Datenmodellen arbeiten, implementiert werden.

## 4 Architektur

Odysseus ist ein Framework für die Entwicklung von DSMS. Die komponentenbasierte Architektur von Odysseus (s. Abb. 1) erlaubt eine leichte Anpassbarkeit für Anwendungen verschiedener Domänen. Diese Anpassbarkeit wird durch eine Trennung von fixen und variablen Funktionen in den Komponenten erreicht (sog. Fixpunkte und Variationspunkte). Bei den Fixpunkten handelt es sich in der Regel um Odysseus-interne Verwaltungsstrukturen, die die Ausführung einzelner Verarbeitungsschritte übernehmen. Die Variationspunkte erlauben die Anpassung der Verarbeitung an anwendungsspezifische Anforderungen. Die einzelnen Komponenten von Odysseus decken den gesamten Workflow bei der Verwendung eines DSMS ab. Das bedeutet, dass in Odysseus sowohl Komponenten für die Eingabe von Anfragen als auch für deren Weiterverarbeitung im DSMS vorhanden sind.

Die Übersetzungskomponente (Translate) in Odysseus stellt Mechanismen für die Übersetzung von Anfragen in logische Anfragepläne bereit. Eine logische Algebra-Basis kapselt als Fixpunkt der Komponente die Verwaltungsstrukturen, die anwendungsunabhängig in gleicher Weise aufgebaut sind. Zur Unterstützung verschiedener Anfragesprachen kann die Algebra-Basis leicht um weitere Operatoren (Variationspunkt) erweitert werden.

Die Restrukturierungskomponente (Restruct) steuert die Restrukturierung und Optimie-

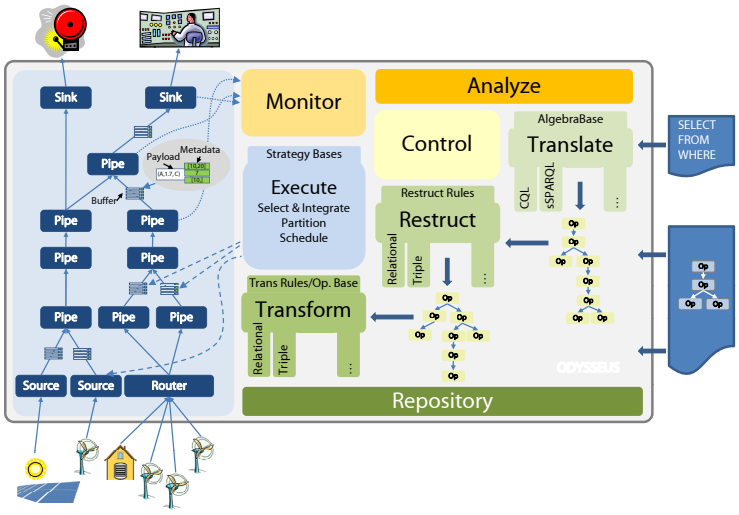


Abbildung 1: Architektur von Odysseus

rung logischer Anfragepläne. Diese regelbasierte Komponente stellt als Fixpunkte Mechanismen zur Verwaltung von Restrukturierungsregeln und deren Anwendung auf logische Anfragepläne bereit. Diese Komponente kann durch eigene Regeln (Variationspunkt) so erweitert werden, dass auch neue Anwendungsfelder oder Anfragesprachen von speziellen Optimierungsregeln profitieren können.

Die Transformationskomponente (Transform) sorgt für die Übersetzung eines (restrukturierten) logischen Anfrageplans in einen physischen Anfrageplan. Hierbei werden Algorithmen verschiedener Operatoren ausgewählt und verschiedene Anfragepläne durch ein Kostenmodell evaluiert. Auch diese Komponente ist regelbasiert und erlaubt somit die Erstellung eigener anwendungs- oder datenmodellspezifischer Regeln. Als Fixpunkt ist hier eine physische Algebra-Basis vorhanden, die den Aufbau physischer Anfragepläne und die Erweiterbarkeit von Odysseus um neue ausführbare Operatoren ermöglicht.

Die Ausführungskomponente (Execute) steuert die Ausführung physischer Anfragepläne. Genau wie in den übrigen Komponenten stellt Odysseus hier neben Fixpunkten auch Variationspunkte bereit. Der Fixpunkt in dieser Komponente ist ein Scheduler, welcher die Ausführungsumgebung bereitstellt. Die Variationspunkte dieser Komponente sind Strategien für den steuernden Zugriff auf physische Anfragepläne (Pufferplatzierungsstrategien, PPS) und für die eigentliche Ausführung der Anfragepläne (Schedulingstrategien, ScS).

Die Monitoring-Komponente (Monitor) sorgt für die Überwachung der Ausführung physischer Anfragepläne. In dieser Komponente wird ein Publish-Subscribe Mechanismus für die Verteilung und Verarbeitung von Metadaten über die Operatorausführung als Fixpunkt angeboten. Die Art der Metadaten und deren Verarbeitung ist auch hier erweiterbar.

Die einzelnen Komponenten werden in den folgenden Kapiteln näher beschrieben.

### 4.1 Übersetzungskomponente

Es gibt vielfältige Möglichkeiten zur Repräsentation von Daten in Datenströmen. Neben Arbeiten zu relationalen Datenströmen [KS05] existieren vor allem viele Arbeiten zu XML-Datenströmen [BFF+07]. Aber auch RDF-Daten können in Datenströmen auftauchen [BGJ08]. Unterschiedliche Datenmodelle erfordern in der Regel auch unterschiedliche Anfragesprachen, um auf die entsprechenden Daten zuzugreifen. So eignet sich SQL sehr gut für den Zugriff auf relationale Daten, während XQuery für den Zugriff auf XML-Daten und SPARQL für RDF-Daten optimiert ist. Um mit Odysseus die Möglichkeit zu bieten, auf Daten in unterschiedlichen Datenmodellen zuzugreifen, muss also eine Möglichkeit vorhanden sein, unterschiedliche Anfragesprachen an Odysseus anzubinden.

Odysseus bietet eine Übersetzungskomponente für unterschiedliche Anfragesprachen an. Durch eine abstrakte Parserschnittstelle ergibt sich die Möglichkeit, Parser für unterschiedliche Anfragesprachen an eine logische Algebra-Basis zu binden. Über diese Schnittstelle können Anfragen verschiedener Datenmodelle in logische Anfragepläne übersetzt werden, welche die Metadaten einer Anfrage in einer Baumstruktusr bereitstellen. Die Algebra-Basis stellt dabei mit abstrakten logischen Algebra-Operatoren Verwaltungsstrukturen für den Aufbau logischer Anfragepläne bereit. Diese abstraken logischen Operatoren bieten Mechanismen zur Operatorverknüpfung und zur Bereitstellung von Ein- und Ausgabeschemata (s. [AAB+05]. Diese logische Algebra-Basis ist so erweiterbar gehalten, dass auch datenmodellspezifische Operatoren durch Vererbungsmechanismen in Odysseus integriert werden können. Im Moment sind in Odysseus sowohl eine relationale Datenstromalgebra als auch eine Datenstromalgebra für RDF-Daten implementiert [BGJ08].

Die Abbildung 2 zeigt einen Ausschnitt der momentan in Odysseus verfügbaren Operatorhierarchie. Die abstrakten Operatoren stellen Verwaltungsstrukturen für den Aufbau von Anfrageplänen bereit. Allgemeine Operatoren können genutzt werden, um Operatoren zu beschreiben, die auf verschiedenen Datenmodellen verwendet werden können. Ein Beispiel hierfür wäre ein Join-Operator, der sowohl auf relationalen Daten als bspw. auch auf RDF-Daten angewendet werden kann. Da logische Operatoren noch keine Algorithmen implementieren, können solche allgemeinen Definitionen sehr gut wiederverwendet werden. Datenmodellspezifische Operatoren, wie bspw. ein SPARQL-Triple-Pattern-Matching-Operator, können dann entweder als Erweiterung der abstrakten Algebra-Operatoren oder als Erweiterung allgemeiner Operatoren implementiert werden (vgl [BGJ08]).

Die Metadaten, die von logischen Operatoren bereitgestellt werden, hängen natürlich vom verwendeten Datenmodell ab. Bei den abstrakten Operatoren sind dies bspw. Angaben über die Attribute eingehender Elemente und Attribute ausgehender Elemente. Bei den allgemeinen Operatoren können z. B. Prädikate, welche softwaretechnisch durch das Strategie-Muster [GHJV95] gekapselt werden, bereitgestellt werden. Informationen über Datenquellen werden bspw. durch den `AccessAO` bereitgestellt.

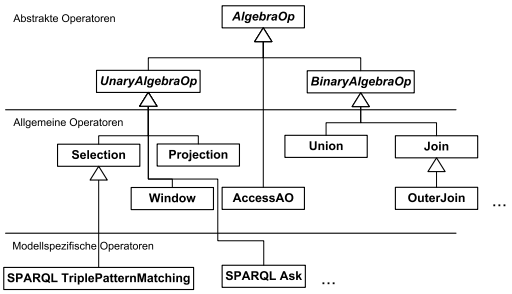


Abbildung 2: Hierarchie der logischen Operatoren

### 4.2 Restrukturierungskomponente

Bei der Übersetzung von Anfragen in logische Anfragepläne werden, je nach Datenmodell, logische Anfragepläne immer auf die gleiche Art und Weise aus einem abstrakten Syntaxbaum erzeugt. Solche Anfragepläne sind natürlich nicht zwangsläufig optimal im Sinne einer effizienten Verarbeitung von Stromdaten. Daher ist in vielen Fällen eine Restrukturierung des erzeugten Anfrageplans sinnvoll. Mit der Restrukturierungskomponente stellt Odysseus eine regelbasierte Umgebung bereit, die die Restrukturierung logischer Anfragepläne erlaubt. Zwar gibt es bereits viele Arbeiten, die sich mit der Restrukturierung von Anfragepläne beschäftigen. Um jedoch neue Forschungsergebnisse auch im Bereich der Restrukturierung von Anfragepläne auf Datenströmen in Odysseus integrieren zu können, ist auch die Restrukturierungskomponente flexibel gehalten.

Der Fixpunkt dieser Komponente ist durch eine Schnittstelle zu einer Regelengine gegeben. Dieser Engine kann als Eingabe ein logischer Anfrageplan sowie eine Menge von Restrukturierungsregeln übergeben werden. Anhand der übergebenen Regeln wird anschließend ein restrukturierter Anfrageplan erzeugt und zurückgeliefert. Aufgrund der Möglichkeit verschiedene Datenmodelle in Odysseus verarbeiten zu können, ist natürlich die Anwendung verschiedenster Regelgruppen zur Restrukturierung entsprechender logischer Anfragepläne notwendig. Daher können beliebige Regeln definiert sowie in Gruppen zusammengefasst werden und der Regelengine zur Anwendung übergeben werden. Als Regelengine wird in Odysseus zurzeit Drools [1] verwendet.

### 4.3 Transformationskomponente

Logische Anfragepläne in Odysseus enthalten nur Metadaten über die Anfrage. Die Operatoren beschreiben auf logischer Ebene, was mit den Daten in einem solchen Anfrageplan passieren soll. Da es für logische Operatoren verschiedene Implementierungen geben kann, ist die Transformation eines logischen Anfrageplans in einen äquivalenten physi-

[1] http://www.jboss.org/drools/

schen Anfrageplan notwendig. Die Transformationskomponente bietet Schnittstellen, um dies für Anfragepläne auf verschiedenen Datenmodellen zu tun. Die Transformationskomponente besteht, ähnlich wie die Restrukturierungskomponente, aus einer Schnittstelle zu einer Regelengine. Diese Schnittstelle ist einheitlich für alle Datenmodelle. Ihr können logische Anfragepläne mit entsprechenden Regeln übergeben werden.

Bei der Transformation eines logischen in einen physischen Anfrageplan werden die logischen Operatoren durch geeignete physische Operatoren ersetzt. Da für einige Operatoren mehrere Implementierungen zur Verfügung stehen können, ist die Erzeugung mehrerer physischer Anfragepläne möglich. Anhand eines Kostenmodells muss anschließend ein geeigneter Plan für die Ausführung gewählt werden.

Um auch die Transformationskomponente erweiterbar zu halten und sie insbesondere für Anfragepläne auf verschiedenen Datenmodellen wiederverwenden zu können, existieren auch hier einige Variationspunkte. Zum einen können beliebige Regeln definiert werden, anhand derer entschieden wird, wie ein logischer Anfrageplan in einen physischen Anfrageplan zu übersetzen ist. Zum anderen existiert eine einheitliche Schnittstelle für die Auswertung von Kostenmodellen. Solange diese Kostenmodelle die jeweiligen Anfragepläne auswerten können und sie außerdem die einheitliche Schnittstelle für die Auswertung implementieren, können sie beliebig definiert werden.

## 4.4 Ausführungskomponente

Die Ausführungskomponente (Execute) nutzt mehrere Teile, die im Folgenden vorgestellt werden. Ein physische Algebra kapselt die auszuführenden Operator-Algorithmen (4.4.1). Mit einem Metadatenframework (4.4.2) wird die Flexibilität für verschiedene Verarbeitungsansätze gewährleistet und ein Scheduler (4.4.3) bestimmt die Reihenfolge der auszuführenden Operatoren.

### 4.4.1 Physische Algebra

Die aus der Transformationskomponente erzeugten physischen Anfragepläne basieren auf einer physischen Algebra-Basis, die als ein Fixpunkt des Odysseus-Frameworks die Verwaltungsstrukturen für physische Anfragepläne bereitstellt. Ähnlich wie in der logischen Algebra-Basis, existieren auch in der physischen Algebra-Basis abstrakte Operatoren, die verschiedene Schnittstellen bereitstellen. Diese Schnittstellen werden im Folgenden genauer dargestellt.

Zwar bietet Odysseus die Möglichkeit Anfragepläne über verschiedenen Datenmodellen auszuführen, jedoch ist diese Ausführung in Odysseus einheitlich geregelt. So können sich Erweiterungen im Sprachumfang oder im Umfang der unterstützten Datenmodelle auf die eigentliche Verarbeitung der Daten beschränken. Die Ausführung oder der Aufbau von Anfragepläne selbst kann bei diesen Erweiterungen von Odysseus übernommen werden. Zu diesem Zweck bietet die physische Algebra-Basis in Odysseus insbesondere Verwaltungsstrukturen für den Aufbau physischer Anfragepläne an. Die Verknüpfung physi-

**Algorithm 1**: Join-Operator

**Input**: SweepAreas: $SA_1$, $SA_2$,
OutputQueue H, MergeFunction $m_d$, MetadataMergeFunction $m_m$,
TransferFunction t

```
1  foreach e := (val, meta) ↩ S_i, i ∈ {0,1} do
2      k ← (i mod 2) + 1;
3      SA_k.purgeElements(e,i);
4      SA_i.insert(e);
5      qualifies ← SA_k.query(e,i);
6      while qualifies.hasNext() do
7          ê := (v̂al, m̂eta) ← qualifies.next();
8          H.insert((m_d.merge(val,v̂al,i), m_m.merge(meta,m̂eta,i)));
9      end
10     t.transfer(Q);
11 end
```

scher Operatoren zu einem Anfrageplan wird über einen Publish-Subscribe-Mechanismus realisiert, der durch die physische Algebra-Basis gekapselt ist. Ein weiterer Publish-Subscribe-Mechanismus erlaubt die Registrierung für und das Auslesen von Ereignissen, die während der Verarbeitung von Stromdaten auftreten können. Auch dieser Publish-Subscribe-Mechanismus wird durch die physische Algebra-Basis gekapselt. Weiterhin implementiert die physische Algebra-Basis eine push-basierte Variante des Open-Next-Close-Protokolls (ONC) [Gra93]. Dies ermöglicht eine Kapselung der Vorgänge Open, Process_Next[2] und Close für datenmodellspezifische Algorithmen.

Um die Ausführung von Anfrageplänen auf verschiedenen Datenmodellen mit Odysseus zu ermöglichen, müssen physische Operatoren entsprechende Daten verarbeiten können. Daher ist die physische Algebra-Basis zunächst unabhängig von diesen Datenmodellen realisiert. Dies wurde im Java-basierten Odysseus durch Generics und das Strategie-Muster gelöst. Über Generics wird auch die Typsicherheit zwischen Operatoren gewährleistet. Da abstrakte physische Operatoren zunächst nur Verwaltungsstrukturen beschreiben jedoch nicht, wie bestimmte Daten verarbeitet werden, kann in der physischen Algebra-Basis auch von den Datenmodellen abstrahiert werden. Odysseus geht jedoch sogar einen Schritt weiter. Für Operatoren, in denen es möglich ist, datenmodellspezifische Eigenschaften zu kapseln, existieren in Odysseus entsprechende abstrakte Algorithmen. Dies lässt sich sehr gut an der Ripple-Join-Technik [HH99, DSTW02] darstellen, die in Algorithmus 1 zu sehen ist.

Im Prinzip funktioniert dieser Operator auf allen Datenmodellen, die eine elementbasierte Verarbeitung erlauben, gleich. Ein Element der linken Seite trifft im Datenstrom ein und wird mit allen Elementen der rechten Seite verglichen. Bei passenden Elementen wird das Join-Ergebnis berechnet und aus dem Operator geschrieben. Vergleiche und das

[2] Während ONC von oben nach unten durch Anfrageplan propagiert wird, geschieht dies bei unserem Protokoll von unten nach oben.

Zusammenführen von Elementen sind durch Strategien gekapselt (`MergeFunction`, `MetadataMergeFunction`, `TransferFunction`). Soll die Ripple-Join-Technik bspw. auf einem anderen Datenmodell angewendet werden, so müssen lediglich die entsprechenden Strategien implementiert werden. Der Algorithmus selbst und damit auch der physische Operator können unverändert wiederverwendet werden.

In Algorithmus 1 ist eine weitere Besonderheit der physischen Algebra zu erkennen. Zustandsbehaftete Operatoren, wie bspw. ein Join, benötigen zur Berechnung eines Ergebnisses Informationen über weitere Elemente im selben oder einem anderen Datenstrom. In der Regel wird auf Datenströmen unabhängig vom verwendeten Datenmodell eine bestimmte (meist zeitliche) Ordnung der Elemente angenommen. Entsprechend müssen diese Elemente also auch in den Zustandsspeichern vorgehalten werden. Da dies zunächst unabhängig vom verwendeten Datenmodell passieren kann, verwendet Odysseus für die Speicherung von Operatorzuständen den abstrakten Datentyp (ADT) SweepArea [DSTW02], der Methoden für das Speichern und den Zugriff auf Elemente in einem physischen Operator bereitstellt. Dieser ADT kann durch zwei Prädikate $p_{query}$ und $p_{remove}$ sowie eine Ordnungsrelation $\leq$ parametrisiert werden. In der Methode `query` bestimmt das Prädikat $p_{query}$, welche Elemente bzgl. $\leq$ sortiert zurückgeliefert werden sollen. Das Prädikate $p_{remove}$ wird in der Methode `purgeElements` genutzt, um alle nicht mehr gültigen Elemente aus der SweepArea zu entfernen. Auch an den SweepAreas ist wiederum der konsequente Einsatz des Strategie-Musters zu erkennen. Die Auswertung von Elementen ist vollständig in den Prädikaten $p_{query}$ und $p_{remove}$ gekapselt, so dass eine SweepArea wiederum datenmodellunabhängig ist und daher auch entsprechend wiederverwendet werden kann.

### 4.4.2 Verarbeitung von Metadaten

Orthogonal zu den verschiedenen Datenmodellen existieren auch Ansätze zur Verarbeitung von Stromdaten, wie in Kapitel 3 bereits mit dem PN-Ansatz sowie dem Intervall-Ansatz dargestellt wurde. Da Odysseus ein Framework ist, welches an verschiedensten Stellen der Stromdatenverarbeitung ansetzen können soll, müssen natürlich auch solche Verarbeitungsstrategien in Odysseus integrierbar sein. Insbesondere sollen natürlich auch neue Ansätze, die in Forschungsarbeiten entwickelt werden, in Odysseus eingebunden werden können, um eine Evaluation dieser Ansätze zu ermöglichen. Daher ist Odysseus auch in dieser Hinsicht erweiterbar gehalten.

Ansätze wie der PN-Ansatz oder der Intervall-Ansatz basieren insbesondere auf einer geeigneten Repräsentation des (zeitlichen) Fortschritts in einem Datenstrom. Anhand dieser Ansätze wird auf unterschiedliche Art und Weise entschieden, welche Elemente in einem Datenstrom aktuell gültig sind. Es geht bei diesen Ansätzen also nicht um die Daten selbst, sondern um Metadaten wie positive oder negative Marker oder auch Gültigkeitsintervalle. Um solche und beliebige weitere Metadaten in Odysseus verarbeiten zu können, bedarf es neben einer flexiblen Möglichkeit zur Repräsentation der entsprechenden Metadaten auch einer gekapselten Verarbeitung dieser Metadaten.

Metadaten, wie die zuvor dargestellten, ergänzen jedes Element eines Datenstroms. Daher bietet Odysseus eine parametrisierte Schnittstelle für Datenstromelemente an, wie in

Abbildung 3 anhand der Klasse `AbstractStreamElement` dargestellt ist. Um ein Element in einem bestimmten Datenmodell darzustellen, muss eine Klasse für das entsprechende Datenmodell diese Schnittstelle implementieren. Dies ist in Abbildung 3 beispielhaft durch die Klassen `RelationalTuple` und `RDFNodeList` dargestellt. Der Parameter der abstrakten Klasse `AbstractStreamElement` repräsentiert die Metadaten eines Elements im Datenstrom. Dieser Parameter muss dabei lediglich die Schnittstelle `IMetadataItem` implementieren. Von dieser Schnittstelle existieren zahlreiche spezialisierte Schnittellen für Gültigkeitsintervalle (`ITimeInterval`), Prioritäten (`IPriority`), Latenzen (`ILatency`) usw. Um einem Datenstromelement also mehrere Metadaten gleichzeitig, z. B. eine Latenz und ein Gültigkeitsintervall, anzuhängen, bedarf es lediglich der Implementierung beider Schnittstellen in einer Klasse. Natürlich hängt die Verarbeitung der Metadaten entscheidend vom Typ der Metadaten ab. Wie jedoch in Algorithmus 1 anhand des Join-Algorithmus zu erkennen ist, geschieht die Verarbeitung der Metadaten gekapselt in entsprechenden Strategien (hier `MetadataMergeFunction`), so dass eine leichte Erweiterbarkeit der physischen Algebra-Basis möglich ist. Neben der Entwicklung von Strategien für die Verarbeitung von Metadaten in den klassichen Operatoren können aufbauend auf der physischen Algebra-Basis natürlich auch spezielle Operatoren zum Umgang mit Metadaten entwickelt werden. So existiert in Odysseus bspw. ein Operator zur Vergabe von Prioritäten an Elemente in Abhängigkeit eines Prädikats (vgl. [JBG+09]).

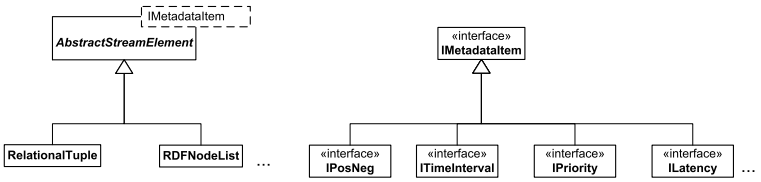


Abbildung 3: Repräsentation von Metadaten in Odysseus

Durch die Kapselung der Metadaten kann Odysseus sehr leicht um verschiedene Verarbeitungsmodi erweitert werden. So unterstützt Odysseus sowohl den PN-Ansatz als auch den Intervallansatz, sowie in Erweiterung zu letzterem die priorisierte Verarbeitung von Datenstromelementen [JBG+09].

### 4.4.3 Scheduler

Nach der Erzeugung eines physischen Anfrageplans wird dieser von Odysseus für die Ausführung vorbereitet. Anfragen auf Datenströmen werden in Odysseus grundsätzlich push-basiert verarbeitet. Das bedeutet, dass bei Ankunft neuer Elemente in einem Datenstrom, diese auch sofort verarbeitet oder explizit gepuffert werden. Für die Ausführung von physischen Anfrageplänen stellt Odysseus eine eigene Ausführungsumgebung bereit, die eine monolithische oder verteilte Ausführung von physischen Anfrageplänen erlaubt. Der Kern dieser Ausführungsumgebung ist ein Scheduler, der eine einheitliche Schnittstelle für die Ausführung von Anfrageplänen bereitstellt. Diese Schnittstelle stellt dabei einen

Fixpunkt im Odysseus-Framework dar. welcher jedoch durch Pufferplatzierungsstrategien (PPS) und Schedulingstrategien (ScS) gesteuert werden kann. Eine PPS bestimmt, wo im Anfrageplan Steuerungspunkte, Puffer, integriert werden (vgl. [CHK+07]). Puffer sind Elementspeicher, welche zwischen physischen Operatoren platziert werden können. Zur Auführung eines physischen Anfrageplans steuert der Scheduler nur Quellzugriffsoperatoren und die Puffer zwischen den physischen Operatoren. Die übrigen Operatoren werden dadurch angestoßen, dass ein Puffer ein Element direkt den nachfolgenden Operatoren übergibt. Die Abbildung 4 zeigt, wie ein physischer Anfrageplan aussehen kann, nachdem zwischen einigen der Operatoren Puffer platziert wurden. Wird beispielsweise im Puffer 4 ein Element zum nächsten Operator transferiert, so wird anschließend direkt der folgende Filter-Operator angestoßen. Alle physischen Operatoren, die direkt, also nicht über Puffer, miteinander verbunden sind, werden immer in einem Scheduling-Schritt ausgeführt. Es ist leicht ersichtlich, dass die Wahl der Pufferplatzierung entscheidenden Einfluss auf das Scheduling hat. Um also verschiedene Platzierungsstrategien leicht in Odysseus integrieren zu können, lässt sich die Odysseus-Ausführungsumgebung durch solche Strategien parametrisieren. Aktuell sind in Odysseus PPS implementiert, die Puffer zwischen allen Operatoren oder keine Puffer im Anfrageplan platzieren. Außerdem ist die Hybrid Multi-Thread-Scheduling (HMTS) [CHK+07] PPS in Odysseus implementiert.

Da in einem Anfrageplan in der Regel mehrere Puffer und mehrere Quelloperatoren vorhanden sind, lässt sich natürlich auch die Reihenfolge des Schedulings dieser Operatoren unterschiedlich wählen. Es gibt viele Arbeiten zu solchen ScS. Um mit Odysseus die Möglichkeit zur Evaluation verschiedener ScS zu bieten, muss eine leichte Integration dieser ScS in Odysseus gewährleistet sein. Daher kann der Odysseus-Scheduler neben der PPS auch durch eine ScS parametrisiert werden. In Odysseus sind momentan unter anderem folgende ScS implementiert: (1) Aurora MinLatency, (2) Aurora MinCost, (3) Aurora MinMemory [CcR+03], (4) Hybrid Multi-Threaded Scheduling [CHK+07], (5) Chain [BBMD03] sowie naive Ansätze wie Round Robin oder Biggest Queue. Durch einen zwei-Ebenen-Ansatz im Scheduler können auch verschiedene Scheduling-Strategien kombiniert werden. So können verschiedene Teilpläne einer Anfrage durch eine ScS gesteuert werden, während innerhalb der Teilpläne die Operatoren durch eine andere ScS gesteuert werden (vgl. [ACc+03, CHK+07]).

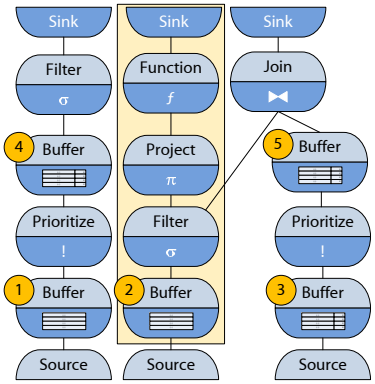


Abbildung 4: Puffer im Anfrageplan

Zusammenfassend wird der Odysseus-Ausführungsumgebung also zunächst ein physischer Anfrageplan, eine PPS sowie eine ScS übergeben. Anhand der PPS werden die Steuerungspunkte für den Scheduler im physischen Anfrageplan gesetzt. Anschließend entscheidet der Scheduler anhand der ihm übergebenen ScS über die Ausführungsreihenfolge der Operatoren.

### 4.5 Monitoringkomponente

Neben der Ausführung von Anfrageplänen auf Datenströmen spielt natürlich auch die Überwachung einzelner oder mehrerer Anfragen in einem DSMS eine Rolle. Odysseus bietet mit der Monitoringkomponente eine Schnittstelle zur Überwachung von Anfrageplänen. Da in verschiedenen Domänen verschiedenste Überwachungsszenarien auftreten können, stellt Odysseus auch hier zunächst eine einheitliche Schnittstelle bereit, welche genutzt werden kann, um Informationen über einen physischen Anfrageplan zur Laufzeit auszulesen und diese Informationen entsprechend auszuwerten. Diese Schnittstelle ist über einen Publish-Subscribe-Mechanismus realisiert, der es erlaubt Events, die während der Verarbeitung von Stromdaten in einem physischen Operator auftreten können, auszulesen und in enstprechenden Verarbeitungseinheiten zu behandeln.

Als Fixpunkt des Frameworks stellt Odysseus abstrakte Events (Initialisierung, Lesen und Schreiben eines Elements) bereit, die zum einen die Art des Events und zum anderen den Ursprung des Events beinhalten. Weiterhin gibt es eine abstrakte Schnittstelle `EventListener`, welche genutzt werden kann, um Events entgegen zu nehmen. Die eigentliche Verarbeitung spezieller Eventtypen wird hier noch nicht betrachtet. Um dies zu erreichen, müssen spezialisierte EventListener implementiert werden, die dann bestimmte Events verarbeiten können. Die Verarbeitung von Events kann periodisch oder auf Anfrage passieren.

Die Registrierung von EventListenern an den Operatoren eines physischen Anfrageplans ist an beliebigen Stellen möglich. Selbst zur Laufzeit können über den von Odysseus bereitgestellten Publish-Subscribe-Mechanismus beliebige EventListener an beliebigen Operatoren registriert werden. Odysseus erlaubt die Erweiterung der Event-Bibliothek um weitere Events. Sollen jedoch spezialisierte Events an bestimmten Stellen während der Verarbeitung von Stromdaten in den Operatoren gefeuert werden, so müssen die Operatoren angepasst werden. Am einfachsten lässt sich dies über spezialisierte Varianten der anzupassenden Operatoren erreichen. So muss in solchen Operatoren lediglich die Verarbeitung der Stromelemente angepasst werden. Alle übrigen Verwaltungsstrukturen verbleiben jedoch unverändert und werden von Odysseus gesteuert.

## 5 Zusammenfassung und Ausblick

Odysseus ist ein Framework, welches ein Grundgerüst für die Anfrageverarbeitung auf Datenströmen bereitstellt. Da an eine solche Anfrageverarbeitung in Abhängigkeit des betrachteten Anwendungsszenarios sehr unterschiedliche Anforderungen gestellt werden können, bietet Odysseus die Möglichkeit der Anpassung der Anfrageverarbeitungsengine an spezielle Anwendungsszenarien. Insbesondere durch die Möglichkeit zur Handhabung verschiedener Datenmodelle bietet Odysseus eine Flexiblität, die unseres Wissens nach von keinem weiteren DSMS geboten wird.

Seine Flexibilität gegenüber verschiedenen Anwendungsszenarien erhält Odysseus aus seiner komponentenbasierten Architektur. Die einzelnen Komponenten dieser Architek-

tur stellen Verwaltungsstrukturen für die Anfrageverarbeitung auf Datenströmen als Fixpunkte des Frameworks bereit. So bietet eine logische und physische Algebra-Basis die Möglichkeit zum Aufbau und zur Restrukturierung von Anfragepläne. Diese Algebra-Basen können durch anwendungsspezifische Operatoren erweitert werden. In dieser Arbeit wurde außerdem gezeigt, wie durch die geeignete Nutzung von Strategien, Algorithmen für verschiedene Datenmodelle wiederverwendet werden können, ohne dabei spezielle Operatoren für jedes Datenmodell in Odysseus integrieren zu müssen.

Weiterhin bietet Odysseus eine Ausführungsumgebung für physische Anfragepläne. Diese Ausführungsumgebung kann durch PPS und ScS beliebig konfiguriert und in ihrem Verhalten beeinflusst werden, ohne dabei Änderungen an der Odysseus-Infrastruktur vornehmen zu müssen. Ähnlich verhält es sich mit einer Monitoring-Komponente, die die Überwachung von Anfragepläne zur Laufzeit erlaubt. Auch hier können beliebige Metadaten über die ausgeführten Operatoren berechnet werden. Sofern die jeweils genutzten anwendungsspezifischen Operatoren dies unterstützen, kann eine solche Überwachung ebenfalls realisiert werden, ohne die Odysseus-Infrastruktur anzupassen. Die Anfrageverarbeitung in Odysseus kann sowohl monolithisch als auch verteilt passieren.

Odysseus dient als Basis für Forschungsarbeiten im Bereich der Datenstromverarbeitung. Zurzeit bietet Odysseus eine algebraische Unterstützung zur Verarbeitung von relationalen und RDF-Daten. In zukünftigen Arbeiten wird Odysseus um spezielle Algebren zur Verarbeitung räumlicher Modelle sowohl in 2D als auch in 3D, zur Verarbeitung von Objektmodellen als auch von bewegten Objekten erweitert. Im Bereich der priorisierten Verarbeitung von Datenstrommelementen werden neue Konzepte, wie spezielle Scheduling-Strategien erforscht. Außerdem wird mit Odysseus zur Zeit der Einsatz von DSMS zur Sensordatenfusion evaluiert. Hierzu werden spezielle Algebren zur Sensordatenfusion entwickelt und in Odysseus auf Daten aus Fahrerassistenzsystemen angewandt.

## Literatur


[AAB+05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemeland Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, AnuragS Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, YingXing und Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conf. on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, Januar 2005.

[ABB+03] Arvind Arasu, Brian Babcock, Shivanath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma und Jennifer Widom. STREAM: The Standford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26, 2003.

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul und Stan Zdonik. Aurora: A new Model and Architecture for Data Stream Management. *The VLDB Journal*, 12(2), 2003.

[BBMD03] B. Babcock, S. Babu, R. Motwani und M. Datar. Chain: Operator scheduling for memory minimization in data stream systems. In *SIGMOD 2003*, Seiten 253–264. ACM New York, NY, USA, 2003.

[BFF+07] Irina Botan, Peter M. Fischer, Dana Florescu, Donald Kossmann, Tim Kraska und Rokas Tamosevicius. Extending XQuery with Window Functions. In *VLDB 2007*, 2007.

[BGJ08] A. Bolles, M. Grawunder und J. Jacobi. Streaming SPARQL-Extending SPARQL to Process Data Streams. *Lecture Notes in Computer Science*, 5021:448, 2008.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss und Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2003.

[CcR+03] Donald Carney, Ugur Çetintemel, Alex Rasin, Stanley B. Zdonik, Mitch Cherniack und Michael Stonebraker. Operator Scheduling in a Data Stream Manager. In *VLDB 2003*, Seiten 838–849, 2003.

[CHK+07] Michael Cammert, Christoph Heinz, Jürgen Krämer, Bernhard Seegerand Sonny Vaupel und Udo Wolske. Flexible Multi-Threaded Scheduling for Continuous Queries over Data Streams. In *First Int. Workshop on Scalable Stream Processing Systems*, 2007.

[DSTW02] J.P. Dittrich, B. Seeger, D.S. Taylor und P. Widmayer. Progressive merge join: A generic and non-blocking sort-based join algorithm. In *VLDB 2002*, Seiten 299–310. VLDB Endowment, 2002.

[Föl06] Florian Fölster. *Erfassung ausgedehnter Objekte durch ein Automobil-Radar*. Dissertation, Technische Universität Hamburg-Harburg, 2006.

[GAE06] Thanaa M. Ghanem, Walid G. Aref und Ahmed K. Elmagarmid. Exploiting predicate-window semantics over data streams. *SIGMOD Rec.*, 35(1), 2006.

[GAW+08] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu und Myungcheol Doo. SPADE: the system s declarative stream processing engine. In *SIGMOD '08: Proc. of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2008. ACM.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson und John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

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

[HH99] Peter J. Haas und Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In *SIGMOD 1999*. ACM Press, 1999.

[JBG+09] Jonas Jacobi, Andre Bolles, Marco Grawunder, Daniela Nicklas und Hans-Jürgen Appelrath. Priorisierte Verarbeitung von Datenstromelementen. In *BTW 2009*, 2009.

[KS05] Jürgen Krämer und Bernhard Seeger. A Temporal Foundation for Continuous Queries over Data Streams. In Jayant R. Haritsa und T. M. Vijayaraman, Hrsg., *COMAD*. Computer Society of India, 2005.

[lCPRK08] Álvaro Catalá-Prat, Ralf Reulke und Frank Köster. Early detection of hazards in driving situations through multi-sensor fusion. In *FISITA 2008*, 2008.

[NLRR02] Jean-Marc Nigro, Sophie Loreitte-Rougegrez und Mechèle Rombaut. Driving situation recognition with uncertainty management and rule-based systems. *Engineering Applications of Artificial Intelligence*, 15(3), 2002.

[Sc05] Michael Stonebraker und Ugur Çetintemel. Öne Size Fits All": An Idea Whose Time Has Come and Gone. *ICDE 2005*, 2005.