Verteilung und Integration von Informationen im Verkehrsbereich

Seminar im Sommersemester 2004

Philipp Bender, Michael Klein, Jutta Mülle, Heiko Schepperle (Hrsg.)
{bender, kleinm, muelle, schepperle}@ipd.uni-karlsruhe.de

Interner Bericht 2004-10

Institut für Programmstrukturen und Datenorganisation (IPD)
Universität Karlsruhe (TH)
Am Fasanengarten 5
76128 Karlsruhe

Oktober 2004

ISSN 1432 - 7864

Kurzfassung

Verteilung und Mobilität spielen in der Verkehrstelematik eine große Rolle. Die verwendeten Datenquellen sind im Allgemeinen heterogen und von unterschiedlicher Qualität. Im Rahmen des Verbundprojektes OVID der Universität Karlsruhe (TH) bot das Institut für Programmstrukturen und Datenorganisation (IPD) im Sommersemester 2004 ein Seminar mit dem Titel „Verteilung und Integration von Informationen im Verkehrsbereich“ an. In diesem Seminar wurden Fragestellungen untersucht, die sich mit den Anforderungen und existierenden Techniken für hochgradige Verteilung und Mobilität von Datenquellen im Verkehrsbereich beschäftigten. Die dabei erzielten Ergebnisse werden in diesem Bericht vorgestellt.

Inhaltsverzeichnis

Vorwort

Das Ziel des seit November 2002 vom Bundesministerium für Bildung und Forschung geförderten Verbundprojektes OVID [OVI02] der Universität Karlsruhe (TH) ist es, verkehrsbezogene Probleme zukünftiger Informationswelten verstehen und neue Techniken nutzen zu lernen. Hierzu erfolgt der Aufbau einer Simulationsplattform zur Modellierung und Bewertung von verkehrsinfrastrukturellen, verkehrstelematischen und logistischen Maßnahmen im Verkehrs- und sozio-ökonomischen System.

Im Rahmen von OVID beschäftigt sich der Lehrstuhl für Systeme der Informationsverwaltung insbesondere mit folgenden Fragestellungen: Wie lassen sich verkehrstelematische Systeme mit Hilfe von Aggregierungstechniken, wie sie z.B. im Data Warehousing verwendet werden, besser unterstützen? Wie kann der Erhalt von Unvollkommenheit in den Daten solche Systeme verbessern? Welche Anforderungen müssen in verkehrstelematischen Systemen unter dem Aspekt der hohen Verteilung und Mobilität der an einem realen System beteiligten Datenquellen besonders berücksichtigt werden?

Da in OVID ausschließlich eine Simulationsplattform zur Modellierung und Bewertung erstellt wird, also die verkehrstelematischen Aspekte nur simuliert werden, spielt die Verteilung und die Mobilität von Datenquellen eine nicht so große Rolle wie das in einem real eingesetzten System der Fall wäre. Aus diesem Grund konzentrieren sich die Arbeiten bezüglich Verteilungs- und Mobilitätsaspekte am Lehrstuhl auf die Analyse der Anforderungen und der aktuell verfügbaren Techniken, die für Verteilung und Mobilität interessant sind. Dazu wurde im Sommersemester 2004 ein Seminar mit dem Thema „Verteilung und Integration von Informationen im Verkehrsbereich“ duchgeführt, dessen Ergebnisse im Folgenden vorgestellt werden.

Die einzelnen Beiträge wurden thematisch gruppiert. Der erste Teil beschäftigt sich mit kleinsten Datenquellen, also mit Sensoren. Im zweiten Teil geht es um mobile Datenquelle. P2P-Datenquellen sind im dritten Teil beschrieben. Die Integration von Datenquellen ist das Thema des vierten Teils. Der fünfte Teil beschreibt eine Auswahl an Anwendungen.

Philipp Bender, Michael Klein, Jutta Mülle, Heiko Schepperle

Teil I

Kleinste Datenquellen

Wenwo Huang

1

Aggregierungsanfragen in Sensornetzwerken

1.1 Einleitung

Motivation

Sensornetzwerke werden bereits in vielen Bereichen eingesetzt. Ein möglicher Einsatzbereich ist der Verkehr, Z.B. das Erkennen und Vermeiden unfallträchtiger Situationen. Unterschiedliche Sensoren (Licht, Temperatur, Luftdruck, atmosphärische Feuchtigkeit, Geräusch etc.) erfassen die Situation auf Straßen und bilden die Eingangsgrößen für eine gezielte Auswertung nach definierten Kriterien. Die Kriterien sind beispielsweise Verkehrsabläufe, Sichtverhältnisse, Hindernisse etc. Einzelne Sensordaten sind aber uninteressant. Eine Aggregation der Daten über viele Sensoren ist nötig.

Überblick

In Abschnitt 1.2 werden zuerst die Grundlagen von Sensornetzwerken erläutert. Dann werden verschiedene Anfragetypen kurz vorgestellt. Anschließend werden allgemeine Aggregationstechniken untersucht. In Abschnitt 1.3 wird zunächst ein Überblick über TinyDB gegeben. Danach wird die Bearbeitung von Aggregierungsanfragen mit Beispiel dargestellt. Die Anfrageoptimierungen werden ebenfalls diskutiert. In Abschnitt 1.4 wird zuerst ein Überblick über das Cougar-Projekt gegeben. Dann wird der Anfrageplan bei Cougar vorgestellt. Anschließend wird die Bearbeitung von Aggregierungsanfragen diskutiert.

1.2 Grundlagen

1.2.1 Sensornetzwerke

Ein Sensornetzwerk besteht aus einer großen Anzahl von Sensorknoten. Einzelne Sensorknoten werden mit anderen Knoten in ihrer Umgebung durch ein drahtloses Netzwerk verbunden und kommunizieren mit Knoten, welche räumlich entfernt sind, mit einem *multihop-routing*-Protokoll [YG03]. Moderne Sensoren verfügen über einen eigenen Prozessor. Sie besitzen Rechen- und Speicherkapazität.

Sensoren haben jedoch folgende Einschränkungen [YG03]:

- Kommunikation

  Das drahtlose Netzwerk bietet normalerweise nur begrenzte Bandbreite und begrenzten QoS (*quality of service*) für die Kommunikation der Sensoren an.

- Energieverbrauch

  Sensoren haben begrenzte Energieversorgung. Die effektive Lebensdauer eines Sensors wird von seiner Energieversorgung bestimmt.

- Berechnung

  Sensoren haben begrenzte Rechen- und Speicherkapazität. Diese Einschränkung beschränkt die Größe der Zwischenergebnisse, die in den Sensorknoten gespeichert werden können.

- Unsicherheit von Sensor-Messwerten

  Die Unsicherheit von Sensor-Messwerten liegt an den technischen Beschränkungen des Sensors und an Umweltgeräuschen.

1.2.2 Anfragetypen

Drei verschiedene Anfragetypen werden unterschieden [PW03]:

- Historische Anfragen

  Historische Anfragen sind Anfragen über historische Daten, welche aus dem Sensornetzwerk bezogen wurden. Z.B. für jeden atmosphärischen Feuchtigkeits-Sensor wird der durchschnittliche Messwert der atmosphärischen Feuchtigkeit im Jahr 2002 abgefragt.

- Schnappschussanfragen

  Schnappschussanfragen beschäftigen sich mit dem Zustand des Netzwerks zu einem festgelegten Zeitpunkt. Z.B. werden die aktuellen

Feuchtigkeitsmesswerte aller atmosphärischen Feuchtigkeits-Sensoren in Baden-Württemberg erfasst.

- Langlaufende Anfragen

  Diese Anfragen beschäftigen sich mit dem Zustand des Netzwerks über eine längere Zeitstrecke. Z.B. für die nächsten 2 Stunden werden alle 30 Sekunden Messwerte aller atmosphärische Feuchtigkeits-Sensoren in Baden-Württemberg ermittelt.

1.2.3 Allgemeine Aggregationstechniken

Eine Aggregation in SQL-basierten Datenbank-Systemen wird durch eine *aggregate*-Funktion und ein *grouping*-Prädikat definiert [MSJC02]. Die standardmäßigen *aggregate*-Funktionen sind COUNT, MIN, MAX, AVERAGE, SUM und zusätzlich noch benutzerdefinierte Funktionen (*UDFs: user-defined functions*).

Grouping ist auch eine standardmäßige Eigenschaft für Datenbank-Systeme. Ein *grouping*-Prädikat teilt die Messwerte der Sensoren in Gruppen nach einigen Attributen auf. Z.B. die Anfrage:

```
SELECT      TRUNC (temp/10), AVERAGE(light)
FROM        sensors
GROUP BY    TRUNC (temp/10)
HAVING      AVERAGE (light) > 50
```

Diese Anfrage teilt die Sensormesswerte in Gruppen nach den Temperatur-Messwerten auf und berechnet den durchschnittlichen Licht-Messwert für jede Gruppe. Der HAVING-Absatz entfernt die Gruppen, deren durchschnittliche Licht-Messwerte kleiner als oder gleich 50 sind.

Nachfolgend werden zwei Aggregationsmethoden vorgestellt: Server-basierte Methode und In-Netzwerk-Methode.

- Server-basierte Methode

  Abbildung 1.1 stellt eine server-basierte Methode dar. In Abbildung 1.1(a) wird ein Sensornetzwerk mit 6 Sensorknoten illustriert. Der Routing-Baum wird mit durchgezogenen Kanten dargestellt. (Wie ein Routing-Baum erzeugt wird, wird in Abschnitt 1.3.2 detailliert erläutert.)

  Sensoren progagieren Daten zum Wurzelknoten (Knoten mit Kennzeichen 0) entlang dem Routing-Baum. Jeder Knoten wird mit dem Abstand zum Wurzelknoten gekennzeichnet. Z.B. für den Knoten in Abbildung 1.1(b), der mit der Zahl 3 gekennzeichnet ist, braucht man drei Nachrichten, um Daten an den Wurzelknoten zu senden. Wie aus

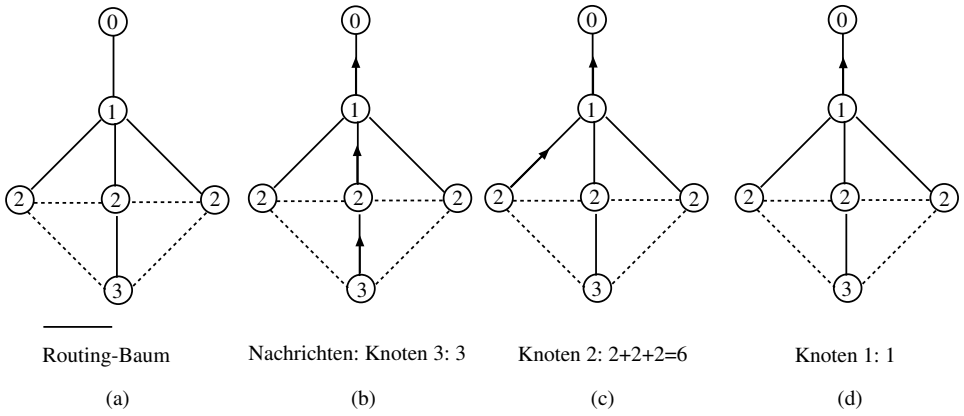


Abbildung 1.1: Server-basierte Methode

der Abbildung ersichtlich, braucht es insgesamt 10 (3 + 6 + 1) Nachrichten, um alle Daten des Sensornetzwerks zu aggregieren.

- In-Netzwerk-Methode

  Wie oben erwähnt, braucht es insgesamt 10 Nachrichten für die Aggregation des Sensornetzwerks bei der Server-basierten Methode. Für ein Sensornetzwerk mit nur 6 Knoten, sind aber 10 Nachrichten schon zu viel. Die Grundidee für die In-Netzwerk-Methode ist: Weil Kommunikation mehr Energie als Berechnung braucht, sollten die Verkehrsflüsse zwischen Knoten durch lokale Berechnung reduziert werden. Abbildung 1.2 stellt eine In-Netzwerk-Methode dar.

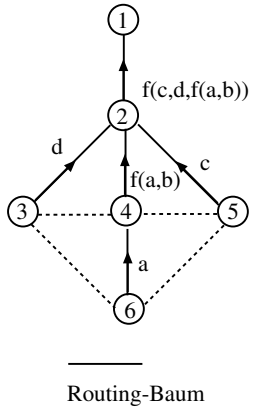


Abbildung 1.2: In-Netzwerk-Methode

Aggregate werden bei der In-Netzwerk-Methode partiell oder komplett von den Sensoren selbst berechnet, während die Daten durch das Netzwerk zum Host-PC geleitet werden. Wie in Abbildung 1.2 dargestellt

wird, werden die Verkehrsflüsse im gleichen Sensornetzwerk von 10 Nachrichten bei der server-basierten Methode auf 5 (1(a) + 1(d) + 1(f(a,b)) + 1(c) + 1(f(c,d,f(a,b)))) Nachrichten bei der In-Netzwerk-Methode reduziert.

1.3 TinyDB

Für die Implementierung der Aggregierungsanfragen werden in dieser Ausarbeitung zwei Einsätze untersucht: TinyDB der Universität Berkely und das Cougar-Projekt der Universität Cornell. In diesem Abschnitt wird TinyDB vorgestellt. Cougar-Projekt wird in Abschnitt 1.4 diskutiert.

1.3.1 Überblick

Die Plattform von TinyDB besteht aus sogenannten *Motes* und aus *TinyOS* [MSJC02]. Die Sensorgeräte in TinyDB werden als *Motes* bezeichnet. Ein solches Sensorgerät besteht aus einem Speicher (128KB Flash, 4KB RAM, 4KB ROM, 512 KB Externflash), einem Mikroprozessor (Amtel 8-bit 4 MHz), einem Funk (RF 916Mhz) und einer Menge von Sensoren.

TinyOS bietet Dienste, welche das Schreiben von Programmen vereinfachen. Diese Programme fangen Sensordaten ab, bearbeiten sie und übertragen die Nachrichten über den Funk. Die Nachrichten in der aktuelle TinyOS-Generation sind 30-Byte-Nachrichten. Jede Nachrichten hat eine Nachrichten-ID und jeder Sensor hat eine eindeutige Sensor-ID.

1.3.2 Bearbeitung von Aggregierungsanfragen

- Erzeugung eines Routing-Baums

  Für die Durchführung einer Aggregierungsanfrage sollte zuerst ein Routing-Baum vor der Aggregation erzeugt werden.

  Ein Sensorknoten wird nach Belieben als Wurzelknoten ausgewählt. Eine Nachricht für die Erzeugung eines Routing-Baums wird zuerst an dem Wurzelknoten geschickt. Der Wurzelknoten sendet dann den anderen Sensorknoten in seiner Umgebung eine Nachricht (Broadcast). In der Nachricht werden die Sensor-ID und die Stufe (die Distanz zum Wurzelknoten) spezifiziert. Für den Wurzelknoten ist die Stufe gleich 0. Der Sensor, der die Nachricht empfangen hat, markiert seine Stufe als die Stufe der Nachricht plus 1, wenn seine aktuelle Stufe größer als die Stufe der Nachricht ist. Der Sensor hält den Sender der Nachricht für seinen Vaterknoten und leitet die Nachrichten gleichzeitig weiter. Zu beachten ist: Ein Knoten wählt nur einen Sender nach bestimmten

Kriterien (z.B. geringe Kosten) als seinen Vaterknoten aus, wenn er Nachrichten von mehreren Knoten empfangen hat. Z.B. hat der Knoten 6 in Abbildung 1.3(a) Nachrichten von drei verschiedenen Knoten (Knoten 3, 4 und 5) empfangen. Er wählt nur einen Knoten (Knoten 4) wegen des kürzesten Pfads als seinen Vaterknoten aus. Durch dieses Verfahren wird ein Routing-Baum erzeugt.

- Eine Pipeline-Aggregation

  Nach der Erzeugung des Routing-Baums wird eine Aggregation durchgeführt. Hier wird eine Aggregationsmethode Pipeline-Aggregation vorgestellt. Diese Methode wird in Abbildung 1.3 illustriert.

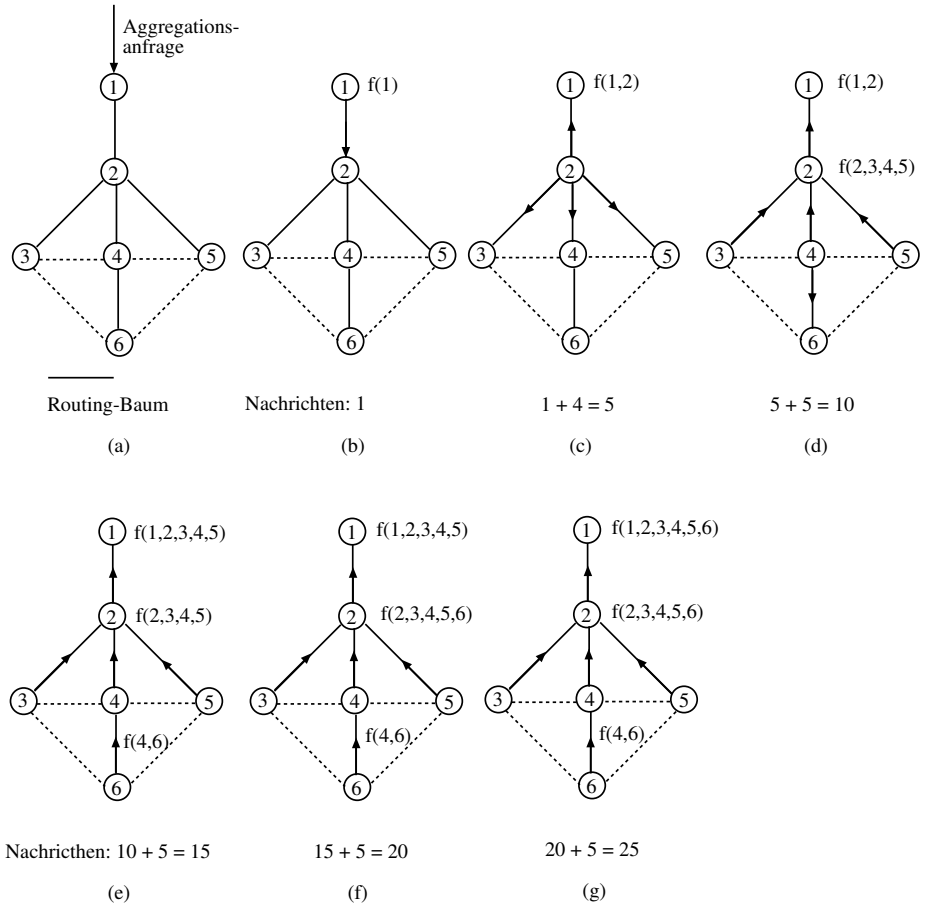


Abbildung 1.3: Pipeline-Aggregation

Abbildung 1.3(a) stellt die Baumstruktur des Sensornetzwerks dar. Am Anfang hat Knoten 1 eine Aggregationsnachricht bekommen und dann schickt er eine Nachricht an sein Kind Knoten 2 (Abbildung 1.3(b)). Knoten 2 hat die Nachricht bekommen und leitet sie an Kno-

ten 3, 4 und 5 weiter und gleichzeitig schickt er seine Daten nach Knoten 1. Am Knoten 1 werden die Daten von Knoten 1 und 2 aggregiert (Abbildung 1.3(c)). Knoten 3, 4 und 5 haben die Nachrichten von Knoten 2 bekommen und dann schicken sie ihre Daten an Knoten 2. Knoten 4 hat noch ein Kind, dann schickt er die Aggreationsnachrichten nach Knoten 6 weiter (Abbildung 1.3(d)). Zu beachten ist: Knoten 2 schickt gleichzeitig seine unveränderte Nachrichten nach Knoten 1, weil die neuen Aggregationswerte von Knoten 2, 3, 4 und 5 bis jetzt noch nicht am Knoten 2 berechnet werden. Nachdem der Knoten 2 die Daten von Knoten 3, 4 und 5 empfangen hat, werden dann die Aggregate am Knoten 2 berechnet.

Für diese Aggregationsmethode braucht es insgesamt 25 Nachrichten für Propagation und Aggregation. Wie vorne bereits erwähnt, braucht die Kommunikation mehr Energie als die Berechnung. Also sollte die Anzahl der Nachrichten so stark wie möglich reduziert werden. Es gibt dazu einige Optimierungsverfahren. In Abschnitt 1.3.3 werden zwei davon vorgestellt: Duplikatvermeidung und Testen von Hypothesen.

1.3.3 Anfrageoptimierung

- Duplikatvermeidung

  Der Grundgedanke dieses Verfahren ist: Sensoren übermitteln eine Nachricht nur wenn der Aggregatwert verändert ist. Die Vaterknoten könnten annehmen, dass die Aggregatwerte ihrer Kinder unverändert sind, wenn sie keine neue Nachrichten von ihren Kinderknoten bekommen [MSJC02].

  Für das in Abschnitt 1.3.2 gegebene Beispiel werden die unveränderten Nachrichten nicht an den Vaterknoten abgeschickt. Durch dieses Verfahren werden die Verkehrsflüsse im Sensornetzwerk deutlich von 25 Nachrichten auf 13 Nachrichten (Nachrichten für Propagation und Aggregation) reduziert.

- Testen von Hypothesen

  Der Grundgedanke dieses Verfahren ist: Es werden nur die Sensordaten abgefangen, die das Resultat des Aggregats beeinflussen. Ein Sensor kann die Messwerte von den Sensoren, die sich in derselben Ebene befinden, aufspüren und schickt seinen eigenen Messwert nicht an den Vaterknoten, wenn er weiß, dass sein Messwert das Resultat des Aggregate nicht beeinflussen kann [MSJC02].

  Dieses Verfahren kann die Verkehrsflüsse bei MAX- und MIN- Aggregierungsanfragen deutlich reduzieren. Aber bei SUM- und COUNT-Aggregierungsanfragen ist dieses Verfahren nicht verwendbar.

1.4 Cougar

In diesem Abschnitt wird das Cougar-Projekt der Universität Cornell diskutiert. Am Anfang wird ein Überblick gegeben. Dann wird die Bearbeitung von Aggregierungsanfragen vorgestellt. Anschließend wird die Synchronisation erläutert.

1.4.1 Überblick

Bei Cougar gibt es drei Sensorklassen im Sensornetz, nämlich Source-Knoten, Leiter-Knoten und Gateway-Knoten [YG03]. Source-Knoten liefern nur Daten und haben keine Berechnungsfähigkeit. Leiter-Knoten sind dazu bestimmt, die Daten von den ihnen unterstellten Sensoren zu sammeln und damit beispielsweise Aggregate zu berechnen. Die Ausgangsdaten der Sensoren müssen nur zum Leiter-Knoten geschickt werden und nicht zum Gateway-Knoten. Der Gateway-Knoten ist eine spezielle Art eines Sensorknoten. Jegliche Kommunikation von außerhalb des Sensornetzwerks muss über den Gateway-Knoten erfolgen. In Abbildung 1.4 wird ein Sensornetzwerkmodell bei Cougar dargestellt.

```
SELECT  AVG (S.value)
FROM    Sensor S
WHERE   S.loc IN Region A
```

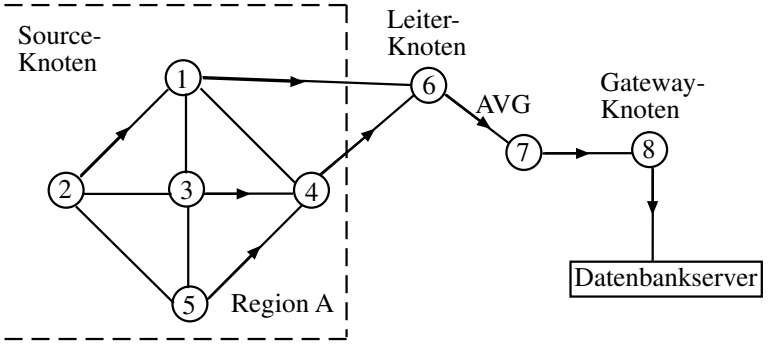


Abbildung 1.4: Ein Sensornetzwerkmodell bei Cougar

Die Sensor-Daten bei Cougar sind Datensätze mit einigen Feldern: ID, Standort, Zeitstempel, Sensortyp und abgelesener Messwert.

Nachdem der Benutzer eine Anfrage gestellt hat, wird vom Server ein Anfrageplan erstellt und optimiert. Ein Anfrageplan entscheidet, wie viele Berechnungen im Netzwerk durchgeführt werden, bestimmt die Rolle und die Verantwortlichkeit jedes Sensorknotens und spezifiziert, wie die Anfrage durchgeführt wird und wie die relevanten Sensoren koordiniert werden. Ein Anfra-

geplan wird in sogenannte Flussblöcken (*flow blocks*) zerlegt. Ein Flussblock ist eine Menge von Sensoren mit einem Leiter-Knoten.

1.4.2 Bearbeitung von Aggregierungsanfrage

Um die Sensordaten zum Leiter-Knoten zu schicken, gibt es drei verschiedene Verfahren [YG03]:

- Direktes Verschicken der Daten an den Leiter
  Jeder Source-Knoten sendet seine Daten zum Leiter-Knoten. Die Berechnung wird nur am Leiter-Knoten nach dem Empfang aller Sensordaten durchgeführt.

- Zusammenlegung von Paketen
  Bei drahtloser Kommunikation ist es teurer, mehrere kleine Pakete anstatt eines großes Pakets zu übermitteln. Bei diesem Verfahren wird ein großes Paket aus vielen kleinen zusammengelegt. Z.B. ein Sensorknoten hat die Daten-Pakete von anderen Sensorknoten empfangen. Weil die Daten-Pakete von Sensoren normalerweise klein sind, legt er die kleine Pakete mit seinem Daten-Paket zusammen und schickt das größere Paket zum Leiter-Knoten.

- Partielle Berechnung des Aggregats durch Zwischenknoten
  Im Gegensatz zum ersten Verfahren werden die Daten auf dem Weg schon aufbereitet durch eine Art von Unter-Leiter-Knoten (Zwischenknoten). Bei diesem Verfahren kann jeder Zwischenknoten die partielle Berechnung des Aggregats durchführen, während beim vorigen Verfahren, die Zwischenknoten nur kleine Paketen zu einem großen Paket zusammenlegen können. Sie haben also keine Berechnungsfähigkeit. Die Berechnung des Aggregats erfolgt nur am Leiter-Knoten.

1.4.3 Synchronisation

Die Verfahren *Zusammenlegung von Paketen* und *Partielle Berechnung durch Unterknoten* sind Beispiele der In-Netzwerk-Methode (Abschnitt 1.2.3). Synchronisation spielt eine wichtige Rolle in diesen beiden Verfahren. Die Aufgabe der Synchronisation zwischen den Knoten des Netzes lautet:

- Auf wie viele Sensormesswerte soll gewartet und wann soll die Zusammenlegung von Paketen oder die partielle Berechnung durch Unterknoten ausgeführt werden?

- Wie kann das Warten auf nicht erreichbare Knoten vermieden bzw. minimiert werden.

Hier werden zwei Synchronisationsverfahren vorgestellt: *Incremental Time Slot* und *Vorhersagen durch Vergangenheitsbetrachtung*

Incremental Time Slot

Jeder Knoten im Netz wartet eine bestimmte Zeit, bevor er entscheidet, ob die ihm untergeordneten Knoten erreichbar sind. Aber bei diesem Verfahren gibt es folgende Probleme [Chr03]:

- Es ist sehr schwierig die Wartezeit im Voraus zu bestimmen.
- Bei regelmäßigen Ausfällen und somit regelmäßigem Umstrukturieren des Baumes müssen allen Sensoren neue Wartezeiten zugewiesen werden.
- Die Sensoren sind niemals ganz zeitsynchron, wenn keine aufwändige Zeitsynchronisationsprotokolle benutzt werden.

Die Lösung für die Probleme ist:

Vorhersagen durch Vergangenheitsbetrachtung

Sobald ein Knoten p ein Paket von Knoten n erhält, wird n auf die Warteliste von p gesetzt. p (Vaterknoten von n) erwartet, andere Pakete von n in der nächsten Runde zu empfangen.

Die somit von p getroffene Vorhersage kann auf zwei Weisen fehlschlagen:

- Knoten n hat einen neuen Vaterknoten. Aber Knoten p (alter Vaterknoten von n) weiß es nicht, und hält somit Knoten n in seiner Warteliste und wartet auf die Pakete von Knoten n.
- n sendet seine Daten nicht an p, weil eine lokale Bedingung nicht erfüllt ist z.B. ein Mindestgrenze. Das Löschen von n aus der Warteliste von p wäre dann aber falsch.

Die Lösung für die Probleme ist: Knoten n sendet an Knoten p ein *notification packet*, wenn die Vorhersage seines Vaterknotens falsch ist [YG03]. Also im ersten Fall muss Knoten n seinen alten Vaterknoten p durch ein *notification packet* benachrichtigen, dass er einen neuen Vaterknoten ausgewählt hat und somit ihm keine Daten mehr senden wird. Im zweiten Fall muss Knoten n seinen Vaterknoten p informieren, dass er keine Daten senden wird, weil irgendeine Bedingung nicht erfüllt ist.

1.5 Fazit

In dieser Ausarbeitung wurden Grundlagen von Sensoren und Sensornetzwerken vorgestellt. Sensoren haben beschränkte Ressourcen. Einzelne Daten

von Sensoren sind uninteressant. Aggregation ist somit nötig.

Zwei Aggregierungsmethoden wurden diskutiert: Server-basierte Methode und In-Netzwerk-Methode. Weil Kommunikation mehr Energie als Berechnung braucht, sollten die Verkehrsflüsse zwischen Knoten durch lokale Berechnung reduziert werden.

Für die Implementierung der Aggregierungsanfragen wurden zwei Ansätze untersucht: TinyDB der Universität Berkely und das Cougar-Projekt der Universität Cornell. Für TinyDB wurde eine In-Netzwerk-Methode *Pipeline-Aggregation* mit Beispiel dargestellt. Zwei Optimierungsverfahren wurden ebenfalls beschrieben: *Duplikatvermeidung* und *Testen von Hypothesen.*

Für Cougar wurden zwei In-Netzwerk-Methoden kurz vorgestellt: *Zusammenlegung von Paketen* und *Partielle Berechnung durch Unterknoten* . Diese beide Verfahren brauchen Synchronisation. Zwei Synchronisationsverfahren *Incremental Time Slot* und *Vorhersagen durch Vergangenheitsbetrachtung* wurden vorgestellt.

Literaturverzeichnis

[Chr03] Björn Christmann. *Verarbeitung von Sensordaten - Seminar Data Warehousing im Verkehrsbereich in SS 2003.* IPD, Universität Karlsruhe (TH), 2003.

[MSJC02] Samuel Madden, Robert Szewczyk, Michael J.Franklin, and David Culler. *Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks.* 2002.

[PW03] Raymond Pon and Wesley W.Chu. *Challenges and Issues in Querying Sensor Networks.* 2003.

[YG03] Yong Yao and Johannes Gehrke. *Query Processing for Sensor Networks.* 2003.

Teil II

Mobile Datenquellen

Nils Hilt

2

Datenmodelle und Anfragen

2.1 Einleitung

Viele Firmen und Organisationen wollen bewegliche Objekte in einer Datenbank speichern. Stellen Sie sich einen Kurierdienst mit vielen Lieferwagen vor. In der Firmenzentrale soll ein Datenbankverwaltungssystem fortwährend die aktuelle Position aller Wagen speichern, damit zum Beispiel neue Lieferaufträge effizient an geschickt plazierte Lieferwagen verteilt werden können.
Aber nicht nur Speditionsunternehmen haben ein Interesse daran mobile Objekte zu verwalten, dies gilt auch für Rettungssdienste, die die aktuellen Positionen ihrer Krankenwagen wissen wollen, für Flugüberwachungen zur Vermeidung von Zusammenstößen von Flugzeugen in der Luft und auch für Taxiunternehmen, die ihren Fuhrpark möglichst Gewinn bringend einsetzen möchten.
Im Beispiel des Kurierdienstes, der die Positionen seiner Lieferwagen in einer Datenbank verwaltet, könnten folgende Anfragen interessant sein:

A1a Welche Lieferwagen befinden sich gerade maximal 500 Meter vom Informatikgebäude am Fasanengarten entfernt?

A1b Welche Lieferwagen befinden sich innerhalb der nächsten fünf Minuten maximal 500 Meter vom Informatikgebäude am Fasanengarten entfernt?

A2 Welche Lieferwagen werden sich in den nächsten 30 Minuten begegnen?

A3 Welche Lieferwagen sind unbeladen, bis sie das Informatikgebäude am Fasanengarten erreichen?

Diese vier Beispielanfragen sind alle von unterschiedlicher Charakteristik. Zur Beantwortung von **A1a** muss man nur die Wegstrecke der Lieferwagen

beachten, während bei Anfrage **A1b** zum räumlichen Aspekt der Bewegung noch eine zeitliche Bedingung hinzu kommt. Da bei **A2** Fahrtrouten und Zeiten mehrerer Lieferwagen betrachtet werden müssen, stellt diese Anfrage eine räumliche und zeitliche Join-Anfrage dar.

2.1.1 Probleme und Anforderungen an die Datenbank

In diesem Abschnitt wird untersucht, in wieweit die Verwaltung beweglicher Objekte durch traditionelle Datenbanksysteme unterstützt wird. Ein Objekt ist *beweglich*, wenn dessen Verwaltung explizit auf dessen Bewegung eingehen muss.

Modellierung der Bewegung In traditionellen Datenmodellen sind die Attributwerte statisch. Dies bedeutet, dass gespeicherte Werte konstant sind, bis sie explizit geändert werden. Zwischen zwei Updates ändern sich die Werte in der Datenbank nicht. Jedoch ändert sich die Position bei sich bewegenden Objekten andauernd. Für die Verwaltung beweglicher Objekte hat das nun zur Folge, dass die Positionsinformationen ständig aktualisiert werden müssen, falls man keine ungenauen und veralteten Position zulassen will.
Dies ist aus dreierlei Gründen nicht akzeptabel: Zum ersten belasten viele Updates das zugrunde liegende Datenbankverwaltungssystem. Und da wir uns in einem mobilen Umfeld bewegen, findet die Kommunikation zwischen Datenbank und den mobilen Objekten drahtlos statt. Die weiteren Gründe sind nun, dass zum zweiten nicht unbegrenzt Bandbreite zur Verfügung steht und zum dritten nicht garantiert werden kann, dass die mobilen Objekte immer eine Funkverbindung aufrecht halten können. Dies ist zum Beispiel dann der Fall, wenn sie sich gerade in einem Straßentunnel oder sonstigen Funklöchern befinden.
In Abschnitt 2.2 werden zwei Datenmodelle vorgestellt, die extra für die Modellierung von beweglichen Objekten entwickelt wurden und die Beschränkung auf statische Attributwerte aufheben.

Anfragen Betrachten wir wieder die Anfrage **A1b**. Wie schon nach den Beispielanfragen in der Einleitung erwähnt, enthält diese Anfrage eine räumliche und zeitliche Komponente. Der räumliche Aspekt wird durch *"maximal 500 Meter vom Informatikgebäude entfernt"* ausgedrückt, während *"innerhalb der nächsten fünf Minuten"* eine zeitliche Bedingung darstellt.
Da traditionelle Anfragesprachen wie SQL oder OQL keine speziellen zeitlichen und räumlichen Operatoren zur Verfügung stellen, sind sie auch nicht dafür geeignet, räumliche und zeitliche Anfragen zu formulieren. Ein weiterer Nachteil von bekannten Sprachen ist, dass diesen Anfragesprachen bestimm-

te Datenmodelle zugrunde liegen, die die Modellierung der Bewegung von Objekten nicht unterstützen.
Deshalb wird später in Abschnitt 2.3.2 mit der Future Temporal Logic (FTL) eine räumliche und zeitliche Anfragesprache vorgestellt.

2.2 Datenmodelle für mobile Objekte

In diesem Abschnitt werden zwei Datenmodelle für bewegliche Objekte vorgestellt. Das ist zum einem das Datenmodell *Moving Objects Spatio Temporal — MOST* , welches von der Gruppe um Ouri Wolfson an der University of Illinois entwickelt wurde [WXCJ98], [SWCD97]. Und zum zweiten ein Vorschlag für ein Datenmodell für Moving Objects Databases von Luca Forlizzi, Ralf Hartmut Güting und anderen von der Universita Degli Studi di L'Aquila (Italien) und der FernUniversität Hagen [FGNS00].

2.2.1 Datenmodell MOST

In der Einleitung wurde schon deutlich, dass die größte Einschränkung traditioneller Datenmodelle für die Verwaltung beweglicher Objekte statische Attribute sind. Statisches Attribut bedeutet, dass der Wert des Attributes konstant ist, solange er nicht explizit geändert wird.
Für die Speicherung von Positionsinformationen ist es aber nötig, dass der Attributwert sich kontinuierlich mit der Zeit ändert, ohne das explizite Updates benötigt werden. Deshalb führt Wolfson zusätzlich zu den statischen Attributen die so genannten **dynamischen Attribute** ein. Der Attributwert eines dynamisches Attribut wird als eine Funktion der Zeit berechnet. Ein dynamische Attribut A wird im MOST-Datenmodell durch drei Subattribute dargestellt:

- $A.updateTime$
- $A.updateValue$
- $A.function$

$A.updateTime$ ist der Zeitpunkt, an dem das dynamische Attribut zuletzt geändert wurde. Zu jedem Änderungszeitpunkt muss in $A.updateValue$ der gerade aktuelle Wert des Attributes eingetragen werden und in $A.function$ eine lineare Funktion angegeben werden, die die zukünftige Veränderung des dynamischen Attributes beschreibt.
Der Wert des Attributes A zu einem Zeitpunkt t_0, wobei $t_0 \geq A.updatetime$ ist, berechnet sich wie folgt:

$$A.value(t_0) = A.updateValue + A.function(t_0 - A.updateTime)$$

Eine explizites Update von A drückt sich entweder durch eine Veränderung von $A.updateValue$, von $A.function$ oder sogar von beiden Subattributen aus, wobei $A.updateTime$ immer automatisch mitgesetzt werden muss.

Bemerkung In diesem Kapitel über Datenmodellen und Anfragesprachen für bewegliche Objekte wird das dynamische Attribut ausschließlich für die Modellierung von räumlichen Koordinaten verwendet. Aber das MOST-Datenmodell ist nicht nur auf die Speicherung von Positionsinformation beschränkt, es kann in allen Bereichen eingesetzt werden, in denen zeitlich veränderliche Zusammenhänge modelliert werden sollen, zum Beispiel zur Darstellung von Temperaturveränderungen oder des Benzinverbrauches.

Modellierung nicht spurgebundener Bewegung

Der beschriebene Ansatz der dynamischen Attributen ist bereits gut geeignet, um die Bewegung von Flugzeugen oder Schiffen zu modellieren. Denn solche Transportmittel zeichnen sich dadurch aus, dass sie sich relativ geradlinig bewegen. Ihre aktuelle Position kann mithilfe zweier dynamischer Attribute — eines für die X-Koordinate, das zweite für die Y-Koordinate — dargestellt werden, während die geradlinige Bewegung problemslos in den linearen Funktionen $X.function$ und $Y.function$ gespeichert werden kann. $X.function$ beziehungsweise $Y.function$ nehmen jeweils den Anteil der Geschwindigkeit des Schiffes oder Flugzeuges bezüglich der X- und Y-Koordinate auf.
Falls sich die Richtung dann doch ändert, werden die dynamischen Attribute nach dem oben beschriebenen Muster aktualisiert. Da bei Flugzeugen und Schiffen eine Richtungsänderung nicht so häufig stattfindet, wird das zugrunde liegende Datenbanksystem und das Kommunikationsmedium nicht unnötig belastet.

Modellierung der Bewegung auf Straßen

Ausgangspunkt ist wieder das Anfangsbeispiel von einem Kurierdienst, der die aktuelle Position seiner Lieferwagen in einer Datenbank modellieren will. Mit Hilfe der dynamischen Attribute kann man die Position der Transporter — ähnlich wie beim Beispiel mit den Schiffen und Flugzeugen — durch zwei solcher Attribute für die X- und Y-Koordinate darstellen und bei Richtungsänderung diese aktualisieren.
Bei einer genaueren Betrachtung der Bewegung von Lieferwagen beziehungsweise der Bewegung von Autos allgemein fällt jedoch auf, das diese nicht geradlinig ist. Autos müssen ihre Bewegungskurve dem Straßensystem anpassen und dort ist es selten möglich nur geradeaus zu fahren. Jede Kurve

und jede Kreuzung bedeuten meistens eine Richtungsänderung.
Beim Einsatz von dynamischen Attributen zur Modellierung der Bewegung eines Lieferwagen hat dies zur Folge, dass bei jeder Kurve und den meisten Kreuzungen, die befahren werden, ein Aktualisieren der Datenbank nötig ist. Das bedeutet auch, dass durch den Einsatz der dynamischen Attribute nichts gewonnen wurde und man mit den gleichen Nachteilen zu kämpfen hat, wie sie schon in Abschnitt 2.1.1 erwähnt wurden.
Um dieses Problem in den Griff zu bekommen, schlägt Wolfson eine Erweiterung des MOST-Datenmodells um **das Konzept der Route** vor. Ein Positionsattribut L wird nun nicht mehr nur mit drei Subattributen dargestellt, sondern durch fünf:

- $L.updateTime$
- $L.x.updateValue$
- $L.y.updateValue$
- $L.speed$
- $L.route$

Wieder wird der Änderungszeitpunkt in $L.updateTime$ gesichert. Die gerade aktuelle Position und momentane Geschwindigkeit eines Wagens wird in $L.(x,y).updateValue$ beziehungsweise in $L.speed$ abgelegt. $L.route$ enthält einen Zeiger auf ein Strecken-Objekt, dass die gewünschte Fahrtroute repräsentiert.
Die Position eines Lieferwagens zum Zeitpunkt t_0 ($t_0 \geq L.updateTime$) lässt sich wie folgt berechnen:
$(L.updateTime - t_0) * L.speed$ ist die auf der Route zurückgelegte Strecke. Nun muss man nur diese Länge auf $L.route$ verfolgen, beginnend bei $L.(x,y).updateValue$, um die aktuelle Position des Lieferwagens zu erreichen.
Leider wird im Paper von Ouri Wolfson [WXCJ98] nicht genau beschrieben, wie man das Subattribut *Route* implementieren könnte.

2.2.2 Datenmodell für Moving Objects Databases

Der Gruppe um Luca Forlizzi geht das Problem der Modellierung der Bewegung mobiler Objekte allgemeiner an als Wolfson mit MOST. Ihr Datenmodell für Moving Objects Databases beschränkt sich nicht nur auf die Bewegung von punktförmigen Objekten wie im MOST-Datenmodell, sondern ermöglicht auch die Modellierung der Bewegung von Punkten, Linien oder auch Regionen [FGNS00]. Ein weiteres Ziel ist, dass sich Veränderung

der Form und Ausdehnung von Linien und Regionen mit der Zeit modellieren lassen.
Die Modellierung gliedert sich in zwei Schritten: zuerst im **abstrakten Modell** und anschließend die Umsetzung ins **diskrete Modell**. Das abstrakte Modell ist sehr einfach und fokussiert sich auf die relevanten Konzepte der Problemstellung. Leider ist es nicht für die Implementierung geeignet. Das diskrete Modell legt dann die interne Darstellung fest und wird jedoch auch komplexer.

Das abstrakte Modell

Das abstrakte Modell ist eine einfache Algebra, die einige grundlegende Datentypen zur Modellierung zur Vefügung stellt:

- ***int*** — Ganzzahl
- ***real*** — reelle Zahl
- ***Point*** — zweidimensionaler Punkt in der Ebene
- ***Line*** — Linie in der Ebene
- ***Region*** — Fläche, die auch Löcher enthalten darf, in der Ebene
- etc.

Zusätzlich zu den Datentypen gibt es einige Typkonstruktoren um komplexere Typen aufzubauen. Der wichtigste Typkonstruktor zur Modellierung der Bewegung — der ***moving()*** Operator — wird jetzt beispielhaft vorgestellt.
Moving() erweitert die Datentypen, so dass sich ihre Werte mit der Zeit verändern können. Ein ***moving(real)*** ist somit eine einfache stetige Funktion von der Zeit in den reelen Zahlenraum. Eine punktförmige Bewegung wird durch ***moving(Point)*** modelliert, hier kann sich die Position im Raum mit jedem Zeitschritt ändern. Schon hier kann man die Einfachheit des abstrakten Modells erkennen. Durch die Angabe eines ***moving(Point)*** erhält man ein punktförmiges Objekt, dass sich beliebig in der zweidimensionalen Ebene bewegen kann. Das Problem, wie dieses mobile Objekt in der Datenbank repräsentiert wird, wird noch nicht betrachtet. Als letztes bietet das abstrakte Modell noch einige Operationen, um verschiedene Datentypen ineinander überführen zu können:

- **trajectory:** ***moving(Point)*** $\mapsto$ ***Line***
  Trajectory bildet ein mobiles Objekt auf seine Fahrtroute ab.

- **length:** ***Line*** $\mapsto$ ***real***
  Length bestimmt die Länge einer Linie.

- **distance:** ***moving(Point)*** $\times$ ***moving(Point)*** $\mapsto$ ***moving(real)***
  Distance gibt die Entfernungen zwischen zwei mobilen Objekten zurück.

- etc.

Mit den Konzepten des abstrakten Modells lassen sich nun bewegliche Objekte modellieren und besonders mit den vorhandenen Operationen Anfragen formulieren. Für die Umsetzung der Modellierung in eine rechnerinterne Darstellung muss sich jetzt das diskrete Modell kümmern.

Das diskrete Modell

Wie schon erwähnt kümmert sich das diskrete Modell nun um die Umsetzung der abstrakten Konzepte in eine rechnernähere Repräsentation. Für die Umsetzung aller abstrakte Konzepte wird auf das Paper [FGNS00] verwiesen, hier wird nur beispielhaft gezeigt, wie man eine ***Line*** und den ***moving(Point)*** im diskreten Modell repräsentieren kann.
Eine abstrakte Linie wird im diskreten Modell durch eine Polylinie darge-

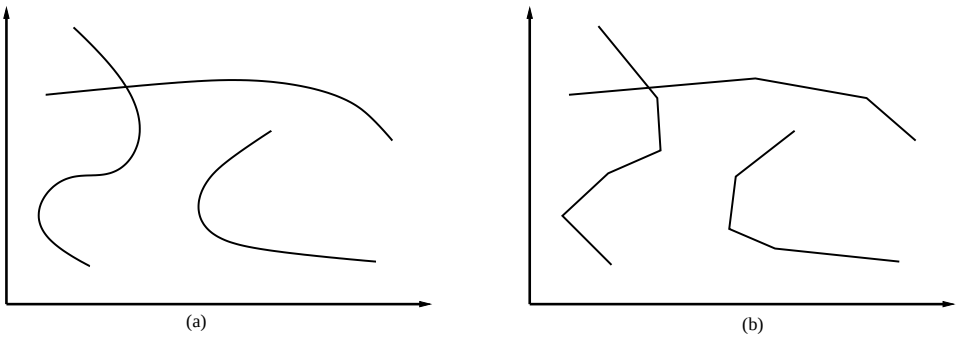


Abbildung 2.1: (a) abstrakte Linie — (b) diskrete Linie

stellt, wie man anhand der Abbildung 2.1 sieht. Eine diskrete Linie ist somit nur eine Annäherung an die gewünschte modellierte abstrakte Linie.
Auch die diskrete Darstellung des ***moving(Point)*** ist eine Annäherung an die wirkliche Bewegung des Objektes. Um diese zu erreichen, wird die durch ***moving(Point)*** dargestellte zu fahrende Strecke in mehrere Abschnitte, so genannte *Unittype*, eingeteilt. Ein *Unittype* ist ein Tupel (i, u), das aus einem Gültigkeitsinterval i und einer linearen Funktion

$u = (x_0, x_1, y_0, y_1)$ mit der Bedeutung $\quad x(t) = x_0 + t * x_1 \quad y(t) = y_0 + t * y_1$

besteht. Ein ***moving(Point)*** wird im diskreten Modell durch eine Liste von *Unittypes* dargestellt, wobei aufeinander folgende *Unittypes* benachbarte Gültigkeitsintervalle aufweisen müssen.

2.2.3 Vergleich beider Datenmodelle

Nachdem zwei Datenmodelle vorgestellt wurden, die für die Bewegung mobiler Objekte geeignet sind, stellt sich die Frage nach den Unterschieden und Gemeinsamkeiten dieser zwei Modelle.
Das MOST-Datenmodell ist ein sehr einfaches Modell, dass eigentlich nur eine geradlinige Bewegung — wenn man mal von der Erweiterung um das Konzept der Route absieht — von Punkten unterstützt. Das abstrakte Modell der Gruppe um Luca Forlizzi stellt einen absoluten Gegensatz dazu dar. Mit seiner mächtigen Ausdruckskraft sind vielerlei Bewegungen und Formveränderungen von Punkten, Linien und Regionen einfach darstellbar.
Betrachtet man aber nun die Umsetzung ins diskrete Modell, fällt besonders bei der Umsetzung des ***moving(point)*** die Gemeinsamkeit zum MOST-Datenmodell auf. Aus einer beliebigen Bewegung wird auch hier letztendlich nur eine lineare Bewegungskurve. Als kleiner Unterschied ist nur noch feststellbar, dass im Standard-MOST-Modell nur die gerade aktuelle Bewegungsrichtung in der Datenbank gespeichert ist, während das diskrete Modell von Forlizzi auch alle alten und noch folgende *Unittypes* mit den Bewegungsbeschreibungen vorrätig hat. Dies ist wiederum mit dem Konzept der Route aus dem MOST-Datenmodell vergleichbar, auch wenn sich Wolfson über die genauere Implementierung ausschweigt.

2.3 Anfragen über mobile Objekte

Bevor in Abschnitt 2.3.2 eine Anfragesprache für räumliche und zeitliche Anfragen vorgestellt wird, sind in 2.3.1 einige Begriffe zu klären.

2.3.1 Datenbankhistorie und Anfragearten

In traditionellen Datenbanksystemen existiert nur ein einziger Datenbasiszustand, die gerade aktuellen Werte aller Attribute. Die hier behandelten Datenbanken speichern aber nicht die gerade aktuelle Position von mobilen Objekten in der Datenbank, sondern eine lineare Funktion ihrer Bewegung. Damit stecken in der Datenbank nicht nur die gerade aktuelle Position, sondern auch implizit alle ihre zukünftigen und auch vergangene Positionen, falls man davon absieht, dass die dynamischen Attributen geändert worden sein könnten.

Es liegt hier also nicht nur ein einziger Datenbasiszustand, sondern eine unendliche Folge von Zuständen vor. In Datenbanken für mobile Objekte nennt man diese Folge von Zuständen **Datenbankhistorie**.
Da man nun nicht nur Anfragen an einen Datenbasiszustand stellen kann, sondern eine ganze Datenbankhistorie zur Verfügung hat, existieren drei verschiedene Arten der Anfragen: Momentananfragen, kontinuierliche und persistente Anfragen

- Eine **Momentananfrage** ist eine Anfrage zu einem festen Zeitpunkt t, die auf der gerade aktuellen Datenbankhistorie ausgewertet wird. Ein Beispiel aus dem Verkehrsbereich: Ein Autofahrer, der gerade auf der Autobahn unterwegs ist und eine Übernachtungspause einlegen will, könnte folgende Momentananfrage stellen: *"Welche Hotels liegen näher als fünf Kilometer zu meiner gerade aktuellen Position?"*

- Eine Folge von Momentananfragen nennt man **kontinuierliche Anfrage**. Nützlich ist sie zum Beispiel, falls bei der Frage nach den nahen Hotels gerade keines gefunden wird, der Autofahrer jedoch weiter fährt, seine Position sich also mit jedem Zeitschritt ändert. Dann wäre es wünschenswert, wenn die Frage nach den Hotels in jedem Zeitschritt neu gestellt wird, bis eines gefunden wird.
  Natürlich ist es ineffizient eine Anfrage andauernd zu stellen. Hier nützt man wieder die dynamischen Attribute aus, die eine Funktion der Bewegung speichern, daraus lässt sich die zukünftige Position berechnen. Eine kontinuierliche Anfrage wird dann nur einmal ausgewertet und jedes Antworttupel enthält zusätzlich ein Gültigkeitsinterval, aus dem sich die gerade gültigen und in den folgenden Zeitschritten gültigen Ergebnisse ableiten lassen. Eine Neuauswertung findet nur statt, falls sich die beteiligten dynamischen Attribute ändern.

- Eine **persistente Anfrage** ist eine Anfrage gegen vorrätig gehaltenen alten Datenbasiszustände. Nötig wird sie zum Beispiel um die folgende Frage zu beantworten: *"Welche Autos haben ihre Geschwindigkeit bezüglich der gerade aktuellen verdoppelt?"* Bei einer kontinuierlichen Anfrage würde ein Auto, dass zum Anfragezeitpunkt 30 km/h fährt, dann erst auf 50 km/h und anschließend auf 70 km/h beschleunigt, nicht gefunden werden.
  Leider reicht zur Beantwortung der persistenten Anfragen, dass Konzept der dynamischen Attribute nicht aus, die Datenbank muss neben der erwartet Bewegungskurve weitere Informationen speichern.

2.3.2 Future Temporal Logic — FTL

Nachdem bisher zwei Datenmodelle für die Modellierung beweglicher Objekte vorgestellt wurde, fehlt jetzt noch eine räumliche und zeitliche Anfragesprache. In diesem Abschnitt wird die *Future Temporal Logic*, die von Ouri Wolfson im Zusammenhang mit dem MOST-Datenmodell eingeführt wurde, vorgestellt.
Der grundsätzliche Anfrageaufbau in FTL sieht wie folgt aus:

```
RETRIEVE o[, v[, ... ] ]
WHERE Praedikat(o, v, ...)
```

Neben den traditionellen boolschen Operatoren (`AND, OR` etc.) stellt FTL noch die benötigten zeitlichen und räumlichen Operatoren zur Verfügung. Die wichtigsten räumlichen Operatoren sind `DIST`, der die Entfernung zwischen zwei Punkten in der Ebene berechnet, und `INSIDE`, der zurück gibt, ob ein Punkt oder ein mobiles Objekt innerhalb eines Polygon ist oder nicht.
Außerdem gibt es noch zwei elementare zeitliche Operatoren: `UNTIL` und `NEXTTIME`. Das Prädikat `f UNTIL g` ist wahr, falls in irgendeinem zukünftigen Zeitschritt das Prädikat `g` gültig ist und aber bis dahin auf jeden Fall `f` erfüllt ist. `NEXTTIME f` sagt nur aus, dass im nächsten Zeitschritt `f` erfüllt ist.

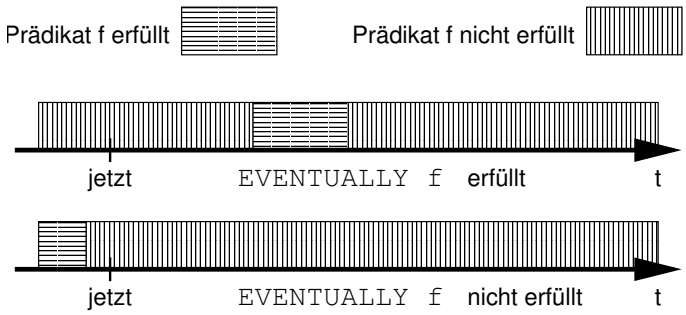


Abbildung 2.2: Eventually f

Zwei wichtige zeitliche Operatoren lassen sich aus `UNTIL` ableiten: `ALWAYS f` mit der Bedeutung, dass `f` für immer gültig sein muss, damit das Prädikat wahr ist (siehe Abbildung 2.3) und `EVENTUALLY f`, dass schon zu wahr wird, falls irgendwann einmal in der Zukunft `f` für mindestens einen Zeitschritt erfüllt ist (siehe Abbildung 2.2).

```
EVENTUALLY f = wahr UNTIL f
ALWAYS f = not(EVENTUALLY not(f))
```

Für diese beiden Operatoren kann der zu betrachtende Zeitraum, durch Anhängen eines Suffixes eingeschränkt werden. Während die beiden Opera-

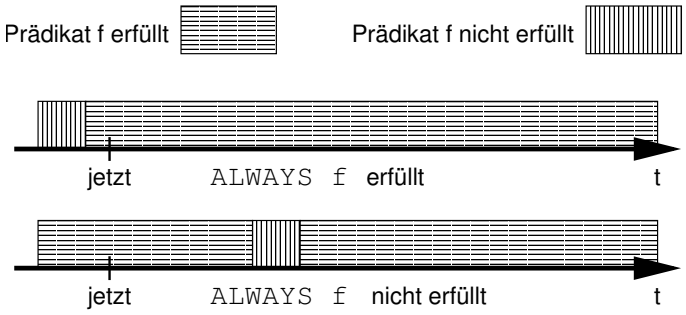


Abbildung 2.3: Always f

toren `ALWAYS` und `EVENTUALLY` zur Auswertung noch die gesamte Zukunft berücksichtigen, kann der Auswertungshorizont dann genauer spezifiziert werden.
Die drei nützlichsten Varianten sind: `ALWAYS_FOR_t f` (Das Prädikat `f` muss für die nächsten `t` Zeiteinheiten immer erfüllt sein), `EVENTUALLY_WITHIN_t f` (Innerhalb der nächsten `t` Zeiteinheiten muss das Prädikat `f` mindestens in einer Zeiteinheit erfüllt sein) und `EVENTUALLY_AFTER_t f` (wobei das Prädikat `f` nach `t` Zeiteinheiten irgendwann einmal erfüllt sein muss).

Beispiele

Zum Abschluss werden die in der Einleitung gestellten Beispielanfragen jetzt noch in der *Future Temporal Logic* formuliert:

A1a Welche Lieferwagen befinden sich gerade maximal 500 Meter vom Informatikgebäude am Fasanengarten entfernt?

```
RETRIEVE o
WHERE o.Typ == LKW
  AND DIST(o, 'Informatikgebaeude') <= 500
```

Da die Anfrage keine zeitlichen Einschränkung hat, wird nur der räumliche Operator `DIST` benötigt.

A1b Welche Lieferwagen befinden sich innerhalb der nächsten fünf Minuten maximal 500 Meter vom Informatikgebäude am Fasanengarten entfernt?

```
RETRIEVE o
WHERE o.Typ == LKW
  AND EVENTUALLY_WITHIN_5min (
```

```
                DIST(o, 'Informatikgebaeude') <= 500 )
```

Bei dieser Anfrage muss zusätzlich noch eine zeitliche Bedingung eingehalten werden (`EVENTUALLY_WITHIN_5min`).

A2 Welche Lieferwagen werden sich in den nächsten 30 Minuten begegnen, d.h. sich auf 5 Meter einander annähern?

```
RETRIEVE o, v
WHERE o.Typ == v.Typ == LKW
  AND EVENTUALLY_WITHIN_30min (
           DIST(o, v) <= 5 )
```

In der Einleitung wurde ja schon erwähnt, dass dies eine räumliche und zeitliche Join-Anfrage ist. In FTL kann man diese Tatsache erkennen, dass zwei Objekte aus der Datenbank wiedergefunden werden sollen (`o` und `v`), die über `DIST(o, v)` miteinander verknüpft sind.

A3 Welche Lieferwagen sind unbeladen, bis sie das Informatikgebäude am Fasanengarten erreichen?

```
RETIREVE o
WHERE o.TYP == LKW
  AND o.beladen == falsch UNTIL
      ( DIST(o, 'Informatikgebaeude') <= 0 )
```

Dies ist eine Anfrage, die des elementaren `UNTIL`-Operators bedarf.

2.4 Zusammenfassung

Dieses Kapitel beschäftigte sich mit dem Problem mobile Objekte effizient in einer Datenbank zu verwalten.
Nachdem festgestellt wurde, dass traditionelle Datenmodelle und Anfragesprachen dafür ungeeignet sind, wurden zwei Datenmodelle und eine Anfragesprache vorgestellt, die speziell für die Verwaltung mobiler Objekte entwickelt wurden. Zum einen war dies das *MOST-Datenmodell* und die *Future Temporal Logic — FTL*, die beide von Ouri Wolfson und anderen entwickelt wurden. Und zum anderen wurde das Datenmodell von der Gruppe um Luca Forlizzi vorgestellt, das die Modellierung der Bewegung von Punkte, Linien und Regionen ermöglicht.

Mit den vorgestellten Ansätzen ist es nun möglich, auch sich ständig ändernde Werte, wie zum Beispiel Positionsinformationen von beweglichen Objekten (LKWs, Schiffen etc.), effizient in einer Datenbank zu speichern und auch wiederaufzufinden.

Literaturverzeichnis

[FGNS00] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proceedings of ACM SIGMOD Int. Conf. On Management of Data*, pages 319–330, Dallas, Texas, 2000.

[Obr04] P. Obreiter. *Bewegliche Objekte in Datenbanken*, chapter 6, pages 79–99. Springer Verlag, Heidelberg, 2004.

[SWCD97] A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.

[WXCJ98] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. *Statistical and Scientific Database Management*, pages 111–122, 1998.

Ingo Beutler

3

Indexstrukturen

3.1 Einleitung

Dieses Kapitel beschreibt, wie räumliche bzw. räumlich-zeitliche Daten gespeichert und bearbeitet werden können. Im Beispiel des Speditionsunternehmens des vorherigen Kapitels werden folgende Beispielfragen bearbeitet:

- „Welcher Lieferwagen befindet sich innerhalb der nächsten 5 min in der Nähe (Umkreis von 2 km) vom Informatikgebäude am Fasanengarten?“
- „Welchen anderen Lieferwagenfahrer werde ich in den nächsten 30 Minuten treffen (um gemeinsam Pause zu machen)?“
- „Welche Lieferwagen sind unbeladen bis sie den Fasanengarten ereichen?“

Um diese Fragen beantworten zu können, wird eine Indexstruktur benötigt, die eine entsprechende Speicherung der Transporter ermöglicht.
Folgende auf einander aufbauende Indexstrukturen werden in diesem Kapitel vorgestellt und ihre Eignung bezüglich der Beispielanfragen überprüft und bewertet:

- B-Baum
- R-Baum
- R*-Baum
- TPR-Baum

Im Mittelpunkt stehen folgende Fragen: Anfragen welcher Art können beantwortet werden? Wie werden sie beantwortet? Sind die Vorgehensweisen effizient?
Am Ende des Kapitels wird eine Möglichkeit zur effizienteren Anfrageverarbeitung kurz skizziert.

3.2 B-Baum

3.2.1 Beschreibung

Der B-Baum ist eine der bekanntesten Indexstrukturen zur Speicherung von großen Datenmengen [BM72]. Es handelt sich hierbei um eine Struktur, bei welcher der Index hierarchisch aufgebaut ist (Baum). Durch Eigenschaften wie vollständige Ausgeglichenheit, ein Mindestfüllgrad der Knoten, Anpassung der Knotengröße an die Pagegröße des Rechnersystem usw. eignet er sich mit Blick auf Leistung und Skalierbarkeit. Er stellt grundlegende Operationen wie Einfügen, Löschen und Suchen zur Verfügung. Besonders eine Variante des B-Baums wird bevorzugt: der B^+-Baum. Während der B-Baum Schlüssel und Daten überall im Baum speichert, enthalten im B^+-Baum nur die Blätter Daten. Die internen Knoten bestehen nur aus Schlüsseln zum Finden der Daten. Mit Hilfe einer zusätzliche Verknüpfung der Blätter können die Daten auch sequentiell durchlaufen werden.

Sein Hauptnachteil bezüglich der Speicherung und Verarbeitung räumlicher Daten besteht in der eindimensionalen Indexierung. Eine Speicherung ist zwar möglich, erfordert aber zusätzliche Anpassungen bei der Speicherung vor allem mit Blick auf die Verarbeitung.

3.2.2 Beispiel-Anfrage

Um die Beispiel-Anfrage “Welche Transporter befinden sich in der Nähe des Fasanengarten?“ aus dem vorherigen Kapitel zu beantworten, könnte ein B-Baum wie in Abbildung 3.1 aufgebaut werden. Das Gebiet des Fasanengarten wird in der Abbildung durch den Hörsaal am Fasanengarten (HSaF) in Form eines Rechtecks dargestellt.

Die x-Achse wird als Index im Baum gewählt. Dadurch gibt es aber Probleme mit Transportern, die den selben x-Wert, aber verschiedene y-Werte besitzen. Die Anfrage “Welche Transporter befinden sich in der Nähe des Fasanengarten?“ kann die bezüglich der x-Achse nächsten Transporter liefern, diese müssen aber nicht die insgesamt nächsten sein, da der y-Wert nicht beachtet wird.

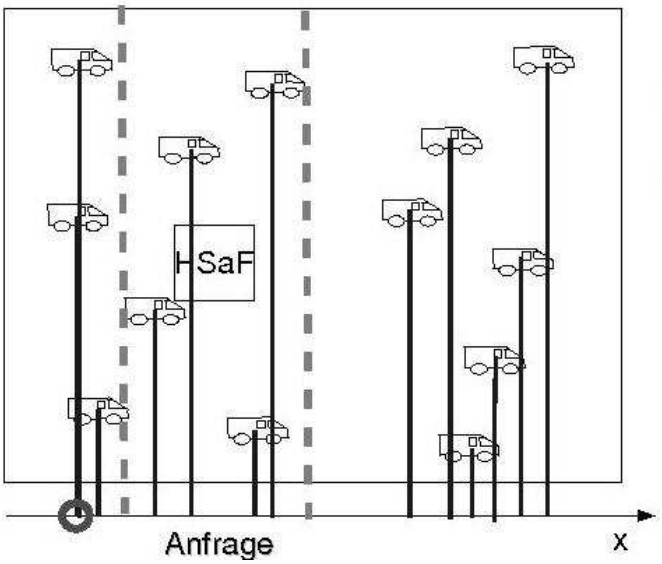


Abbildung 3.1: Beispiel-B-Baum

3.2.3 Bewertung

Bei der Speicherung von räumliche Daten mit Hilfe eines B-Baums entstehen viele Probleme, die außerhalb der Struktur gelöst werden müssen, weshalb sich diese Indexstruktur weniger eignet.

3.3 R-Baum

Um die Nachteile des B-Baums bei der Speicherung räumlicher Daten zu beheben, wird eine andere Indexierung benötigt. Alle anderen Eigenschaften sollen möglichst erhalten bleiben.

3.3.1 Beschreibung

Der R-Baum wurde 1984 von Antonin Guttman entwickelt [Gut84], [Sam90]. Er ist eine Erweiterung des B^+-Baumes zu einem mehrdimensionalen Index. Zur ausführlichen Beschreibung des R-Baums wird der Begriff des Minimum Bounding Rectangle (MBR) benötigt.

Definition(Minimum Bounding Rectangle): Ein MBR ist das kleinst mögliche achsenparallele Rechteck, das ein Objekt oder mehrere Rechtecke umschließt. Ein Beispiel hierfür findet sich in Abbildung 3.2.

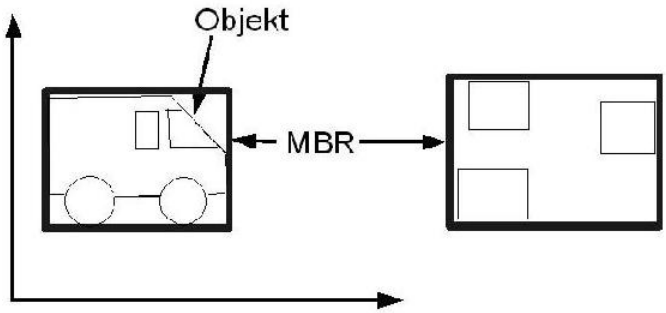


Abbildung 3.2: MBR

Struktur

Der R-Baum fasst alle Objekte im Raum in ineinander geschachtelten Minimum Bounding Rectangles, die jeweils in den Blättern des Baums gespeichert werden, zusammen. In der linke Hälfte der Abbildung 3.3 sind die Transporter T1 und T2 im Rechteck R1 zusammengefasst, welches seinerseits im Rechteck R6 einhalten ist. In der Baumstruktur rechts daneben sieht man, wie dieser Zusammenhang im R-Baum gespeichert wird: R6 zeigt auf den Knoten, der R1 enthält, und R1 auf den Blattknoten mit T1 und T2.

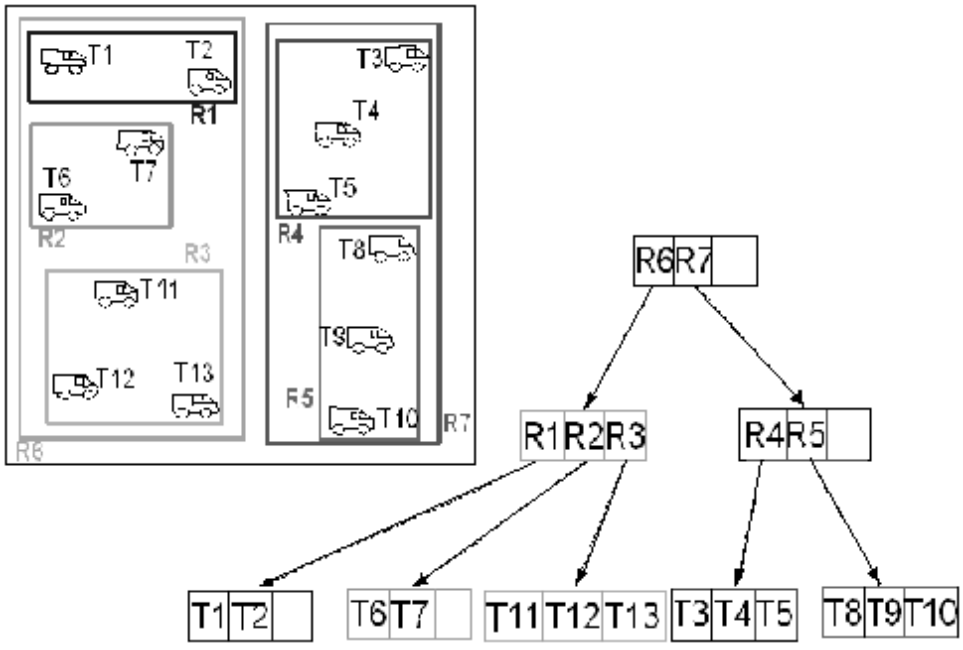


Abbildung 3.3: Beispiel-R-Baum

Das Kriterium nach dem entschieden wird, in welches Blatt ein Objekt ein-

gefügt wird, ist die Minimierung der Fläche zwischen dem äußeren umgebenden MBR und der darin enthaltenen MBRs: Abbildung 3.4 zeigt vier Rechtecke, die in einem größeren Rechteck liegen. Die Fläche, die beim Einfügen in den R-Baum minimiert werden soll, ist die gestreifte Fläche, die zwischen den inneren Rechtecken und dem äußerem Rechteck liegt. So wird ein einzufügendes Element immer in das Rechteck eingefügt, welches durch die Einfüge-Operation am wenigsten vergrößert werden muss.

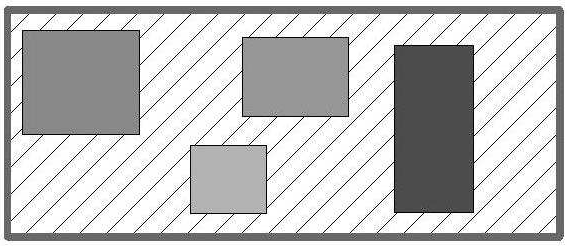


Abbildung 3.4: R-Baum-Ordnungskriterium

Algorithmen

Der R-Baum besitzt analog zum B-Baum die Algorithmen Einfügen und Löschen:

Einfügen Das Einfüge in den R-Baum ist ähnlich zum Einfügen in den B-Baum: Ein neuer Eintrag wird immer in das passende Blatt eingefügt (ChooseLeaf). Entsteht ein Überlauf wird der Knoten in zwei geteilt (NodeSplit). Dieser Überlauf kann sich möglicherweise nach oben fortsetzen, deshalb muss der Baum nach oben hin angepasst werden.

```
ChooseLeaf;
Fuege Eintrag hinzu;
if (Ueberlauf)
  NodeSplit;
  Passe den Baum nach oben hin an;
```

ChooseLeaf Finde einen Blattknoten für den neuen Eintrag E.

```
N = Wurzel;
while (N ist kein Blatt)
  N = Eintrag in N dessen Rechteck geringste Vergroesserung
  benoetigt um E einzufuegen;
return N;
```

NodeSplit Teilen eines Knotens. Da die Einträge des Knotens bestmöglich auf zwei neue Knoten aufgeteilt werden sollen, dies durch reines Ausprobieren aber exponentiellen Aufwand hat, werden drei Verfahren vorgestellt, um den Schnitt zu berechnen.

1. Exhausive

   Es werden alle Möglichkeiten durchprobiert und die beste gewählt (brute force).

   Dieses Verfahren muss $O(2^M)$ Aufteilungen berechnen (exponentieller Aufwand), wobei M die Anzahl der Einträge in einem Knoten ist. Es eignet sich nicht für große Knoten, da es zu langsam ist. Das Verfahren dient hauptsächlich zum Vergleich mit anderen Verfahren, um die Qualität der Ergebnisse zu messen.

2. Quadratic Cost

   Bestimme die zwei Elemente (E1, E2), die das größte MBR besitzen. Diese Elemente bilden den Ursprung der zwei resultierenden Knoten. Bereche jeweils das MBR von E1 bzw. E2 mit jedem anderen Element. Wähle das größte aus und füge das Element dem anderen Knoten hinzu.

   Dieses Verfahren versucht die bestmögliche Aufteilung zu finden, kann diese aber nicht garantieren. Dafür wächst der Aufwand auch nur quadratisch mit der Anzahl der Einträge im Knoten und linear mit der Anzahl der Dimensionen.

3. Linear Cost

   Bestimme die zwei Elemente (E1, E2), die das größte MBR besitzen. Diese Elemente bilden den Ursprung der zwei resultierenden Knoten. Füge die restlichen Elemente zufällig den beiden ausgewählten hinzu.

   Linear Cost ist eine weitere Heuristik, um einen Knoten bestmöglich in zwei zu teilen. Durch die Vereinfachung des zweiten Schritts des Quadratic Cost Verfahrens wächst der Aufwand sowohl mit der Anzahl der Einträge in einem Knoten, als auch mit der Anzahl der Dimensionen nur noch linear. Dafür kann die Qualität des Splits sehr schlecht werden.

Delete Löschen eines Knotens:

```
Finde den Knoten, der den Eintrag enthaelt;
Loesche den Eintrag;
if (Knotenunterlauf)
      loesche den kompletten Knoten und
      foege die enthaltenen Eintr{\"a}ge neu ein (Insert);
```

3.3.2 Beispiel-Anfrage

Jetzt kann die Beispielanfrage “Welche Transporter befinden sich in der Nähe des Fasanengarten?“ besser beantwortet werden. In Abbildung 3.5 ist der Hörsaal am Fasanengarten (HSaF) eingezeichnet.

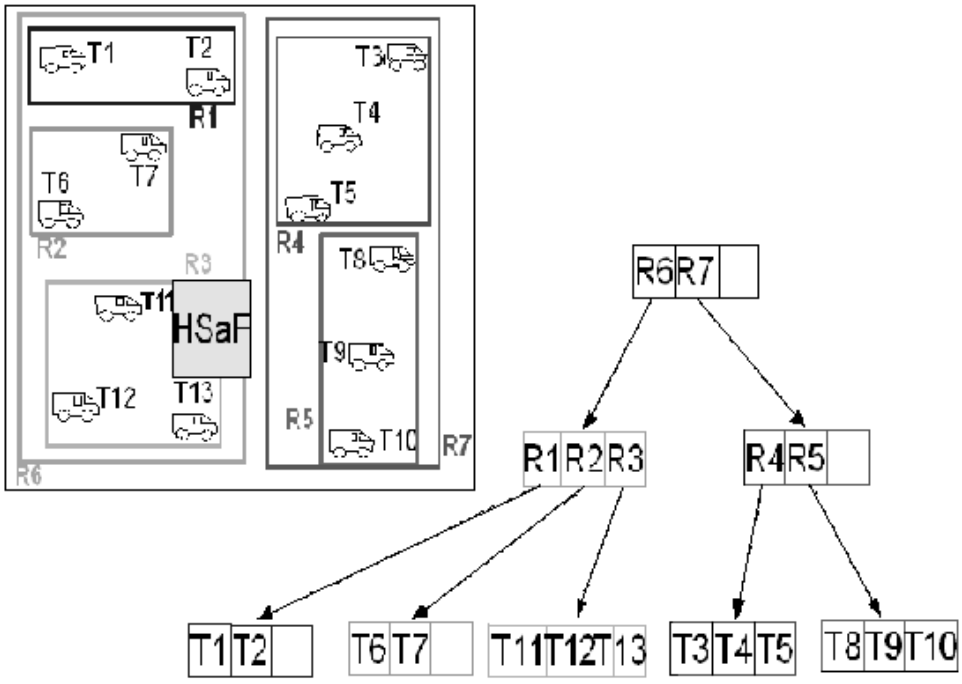


Abbildung 3.5: Beispiel-R-Baum

Um einen Transporter in der Nähe zu finden beginnt der Algorithmus im Wurzelknoten und schaut mit welchem der in der Wurzel enthaltenen Rechteck sich der HSaF überschneidet. Es müssen alle überschneidenden Einträge überprüft werden. Im Beispiel wird nur R6 geschnitten. Nun werden die in R6 enthaltenen Rechtecke auf Überschneidungen überprüft. Im Beispiel überschneidet R3 den HSaF. Da wir jetzt im Blattknoten angekommen sind, muss die Entfernung zu allen enthaltenen Transporten berechnet werden, um den nächsten zu finden.

3.3.3 Bewertung

Der R-Baum ist eine Erweiterung des B^+-Baumes, die sich sehr gut zur Speicherung von räumlichen Daten eignet. Anfragen auf diese Daten können mit Hilfe der Indexstruktur beantwortet werden. Da der R-Baum nur darauf abzielt die Fläche der MBRs zu minimieren, vernachlässigt er andere kritische Eigenschaften der Struktur, wie z.B. die Anzahl und Größe der

Überlappungen von MBRs. Dadurch verschlechtert sich die durchschnittliche Antwortzeit von Anfragen.

3.4 R*-Baum

3.4.1 Beschreibung

Der R*-Baum ist eine Weiterentwicklung des R-Baum [BKSS90]. Er versucht den R-Baum schneller und effizienter bezüglich räumlicher Anfragen zu machen. Die Unterschiede in der Struktur und in den Algorithmen werden im Folgenden beschrieben.

Struktur

Zu dem Kriterium des R-Baums (minimale Fläche) kommen folgende hinzu:

- Minimiere Überlappungen von Rechtecken

  Beim R-Baum kann es zur Überlappung von MBRs kommen. Bei einer Anfrage auf einen solchen Überlappungsbereich müssen alle überlappenden Rechtecke bearbeitet werden. Dadurch vervielfacht sich der Aufwand von Anfragen. Dieses Kriterium zielt darauf ab, Überlappungen zu vermeiden und somit den Aufwand bei Anfragen möglichst gering zu halten.

- Minimiere Umfang der Rechtecke

  Dieses Kriterium soll die Struktur des Baums verbessern. Quadratische Objekte können in einem Rechteck besser zusammengefasst werden, dadurch wird die Fläche der umgebenden Rechtecke geringer.

- Optimiere die Speichernutzung

  Die Speichernutzung kann verbessert werden, indem der Füllegrad der Knoten erhöht wird. Dadurch wird die Anzahl der Knoten geringer bei gleicher Datenmenge. So entsteht einer geringere Höhe des Baumes, was sich auf die Geschwindigkeit der Operationen im Baum positiv auswirkt.

Algorithmen

Die Algorithmen unterscheiden sich in folgenden Teilen:

ChooseLeaf Wenn der Algorithmus die vorletzte Stufe des Baums erreicht hat, wird nicht mehr nach der minimalen Flächenvergrößerung, sondern nach der minimalen Überlappung der MBRs gesucht.
Ziel der Änderung: Es soll ein Verringerung der Überlappungen der Rechtecke, welche die zu speichernden Elemente beeinhalten, erreicht werden.

Reinsert Entsteht beim Einfügen ein Überlauf, dann wird zuerst das vom Mittelpunkt des MBRs am entferntesten liegende Element gewählt und wiedereingefügt(mit Insert).
Ziel der Änderung: Der Füllegrad der Knoten wird erhöht und führt zu einer besseren Speicherauslastung.

SplitNode Entsteht dann wieder ein Überlauf, wird nach folgendem Algorithmus geteilt:

Bestimme die Achse, an welcher geteilt werden soll (ChooseSplitAxis).
Bestimme eine Aufteilung der Elemente in zwei Gruppen (ChooseSplitIndex).
Verteile Elemente auf beide Gruppen

ChooseSplitAxis Für jede Achse:
Sortiere die Einträge zunächst nach dem unteren, dann nach dem oberen Wert ihrer Rechtecke und bestimme alle Verteilungen. Berechne die Summe aller Umfangswerte pro Verteilungen.
Wähle die Achse mit dem minimalen Wert als Splitachse.

ChooseSplitIndex Entlang der gewählten Achse, wähle die Verteilung mit dem minimalen Überlappungswert.
Falls es mehrere Möglichkeiten gibt, wähle die Verteilung mit minimalem Flächenwert.

Ziel der Änderung: Das Verfahren zum Teilen eines Knoten wurde komplett verändert. Bei einem direkten Vergleich mit dem NodeSplit-Verfahren des R-Baums muss beachtet werden, dass die Qualität einer Aufteilung im R*-Baum von mehreren Kriterien abhängt. Das Verfahren zielt auf eine Optimierung der Einteilung in zwei Knoten bezüglich dieser Kriterien. Der Aufwand liegt im Bereich von O(M log M), wobei M die Anzahl der Einträge eines Knotens ist.

3.4.2 Beispiel-Anfrage

Die Anfragen werden genau wie im R-Baum bearbeitet.

3.4.3 Bewertung

Der R^+-Baum ist eine Weiterentwicklung des R-Baumes in Bezug auf Geschwindigkeit. Er enthält alle Vorzüge des R-Baums und wird deshalb diesem vorgezogen.

3.5 TPR-Baum

Bisher wurden nur zeitlich unveränderte räumliche Daten betrachtet. Um die Anfragen aus dem vorherigen Kapitel effizient beantworten zu können, muss die Indexstruktur unter anderem auch Anfragen bezüglich des zukünftigen Ortes von mobilen Objekten bearbeiten können. Dies war mit den bisherigen Strukturen nicht möglich.

3.5.1 Beschreibung

Der Time Parameterized R-Baum (TPR-Baum) passt den R*-Baum zur Speicherung von mobilen Objekten an [SJLL02]. Da die Objekte, wie im vorherigen Kapitel eingeführt, als Funktionen gespeichert werden, müssen die MBRs jetzt auch als Funktionen gespeichert werden, und zwar pro Seite eine Funktion.

Abbildung 3.6 zeigt einen TPR-Baum. Wie man sieht, besitzt dieser Baum genau die selben Elemente und die selbe Struktur, wie der R-Baum von Abbildung 3.3. Der Unterschied besteht in den Pfeilen, die sich an jedem Element und an jeder Seite eines Rechtecks befinden. Diese sollen die Richtung und die Geschwindigkeit des entsprechenden Teil, an dem sie sich befinden, darstellen. Beispielsweise fährt T11, T13 nach links und T12 nach rechts. Deshalb muss sich auch die rechte Seite von R3, welches T11, T12, T13 enthält, nach rechts bewegen und die linke Seite von R3 nach links, da sonst einer der Transporter herausfahren würde. Ebenso müssen sich auch die rechte und linke Seite von R6 bewegen. Generell bewegt sich eine Seite so, dass sich kein Element außerhalb seines umgebenden Rechtecks befinden kann.

Durch diese Erweiterungen verlieren MBRs ihre Minimalität und werden zu Bounding Rectangles (BRs), bzw. zu Time Parameterized Bounding Rectangles (TPBRs).

Struktur

Zusätzlich zu den Kriterien des R*-Baums werden nun die Geschwindigkeitsvektoren der Objekte betrachtet. Dadurch können geringere Geschwindig-

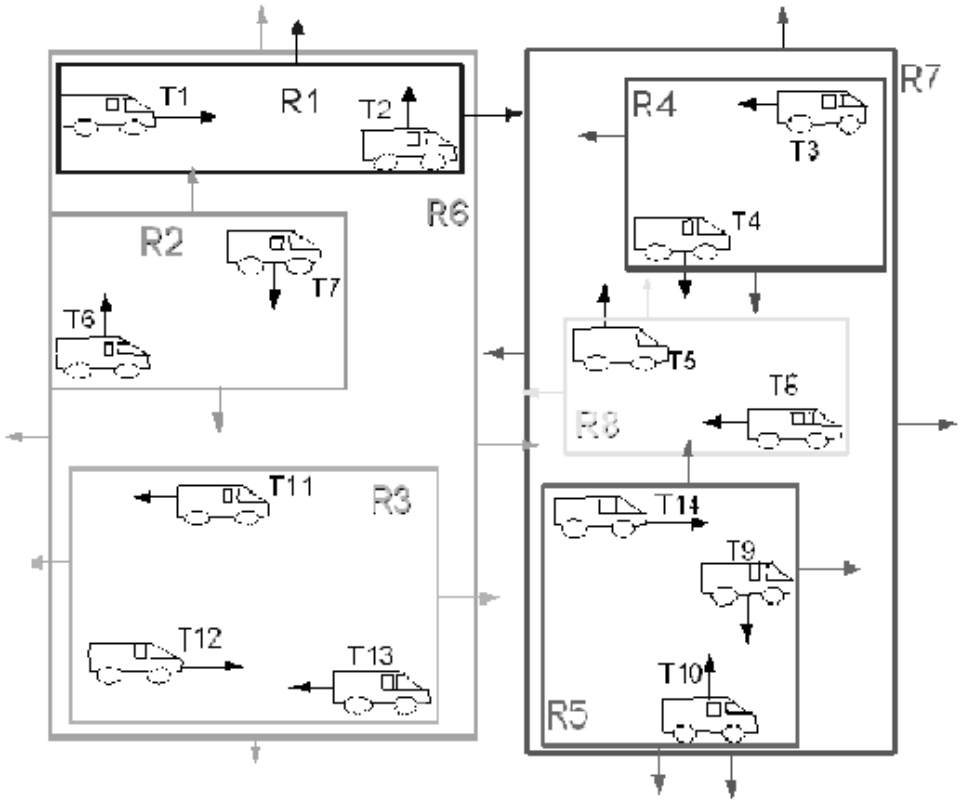


Abbildung 3.6: TPR-Baum

keitsunterschiede zwischen den Objekten eines Rechtecks erreicht werden, wodurch sich das Wachstum des Rechtecks verringert.

Würde in Abbildung 3.6 T12 auch nach links fahren und besäßen alle drei Transporter die selbe Geschwindigkeit, dann wäre das der Idealfall: R3 würde nicht wachsen. Würden alle drei Transporter mit unterschiedlichen Geschwindigkeiten nach links fahren, dann müsste die linke Seite von R3 die Geschwingkeit des schnellsten Transporter besitzen und die rechte die Geschwindigkeit des langsamsten Transporter. Je ähnlicher die Richtungen und Geschwindigkeiten der Einträge eines TPBRs sind, desto kleiner bleibt das TPBR, desto weniger Überschneidungen zwischen TPBRs gibt es.

Algorithmen

Der TPR-Baum übernimmt die Algorithmen des R*-Baum, wobei die Funktionen über die Eigenschaften (z.B. Flächenvergrößerung, Überschneidung, ...) von TPBRs durch ihre Integrale ersetzt werden. Es wird über die Zeit integriert.

Im Verfahren zum Teilen eines Knoten (NodeSplit) werden bei der Sortierung auch die Geschwindigkeitsvektoren berücksichtigt.

3.5.2 Beispiel-Anfragen

Mit dieser Indexstruktur können alle Anfragen aus dem vorherigen Kapitel beantwortet werden.

Die Anfrage "Welche Transporter befinden sich in der Nähe des Fasanengarten?" kann wie im R*-Baum beantwortet werden. Der einzige Unterschied ist, dass die BRs jedesmal für den Anfragezeitpunkt berechnet werden müssen.

Beispiel-Anfrage 1

„Welcher Lieferwagen befindet sich innerhalb der nächsten 5 min in der Nähe (Umkreis von 2 km) vom Informatikgebäude am Fasanengarten?". Das Anfragefenster Q wird zu einem sich bewegenden Fenster. Es bewegt sich entlang der Zeitachse. Abbildung 3.7 stellt die Anfrage dreidimensional dar. Das Anfragefenster Q wird zu einem Rechteck, das entlang der t-Achse wächst. Die Transporter werden zu Linien. Um die Antwort auf die Anfrage zu bekommen, muss berechnet werden, welche Linien das Rechteck schneiden. Dies sind die gesuchten Transporter: T1, T2, T6 und T7.

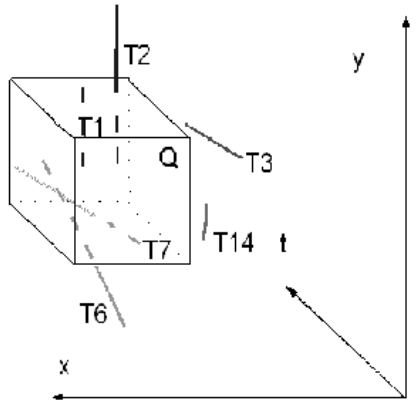


Abbildung 3.7: Beispiel-Anfrage 1

Beispiel-Anfrage 2

„Welchen anderen Lieferwagenfahrer werde ich in den nächsten 30 Minuten treffen (um gemeinsam Pause zu machen)?" In Abbildung 3.8 werden die Transporter wieder als Linien im Dreidimensionalen dargestellt. Die Antwort

auf die Anfrage sind die Linien, welche die „Ich“-Linie schneiden: T4 und T6.

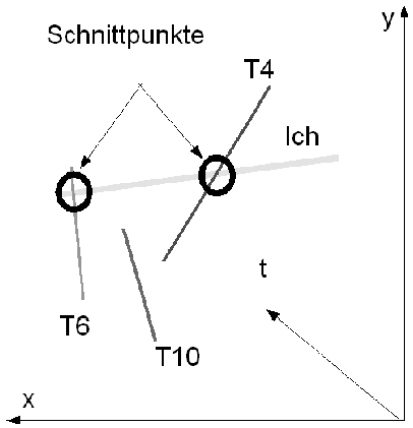


Abbildung 3.8: Beispiel-Anfrage 2

Beispiel-Anfrage 3

„Welche Lieferwagen sind unbeladen bis sie den Fasanengarten ereichen?“. Die Bearbeitung dieser Anfrage funktioniert analog zu Beispiel-Anfrage 1. Jetzt muss noch zusätzlich das Attribut „beladen“ gespeichert werden. In Abbildung 3.9 wird dies durch die Helligkeit der Linie unterschieden. Die Antwort darf nur die schneidenden Linien enthalten, welche „unbeladene“ Transporter darstellen: T7 und T10.

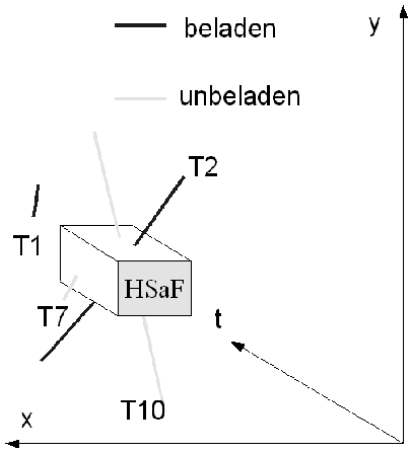


Abbildung 3.9: Beispiel-Anfrage 3

3.5.3 Bewertung

Der TPR-Baum ermöglicht:

- eine effiziente Speicherung von mobilen Daten

  Die Daten müssen nicht bei jeder Änderung der Position, sondern nur bei Geschwindigkeits- und Richtungsänderungen aktuallisiert werden

- die Beantwortung von Anfragen über die Zukunft

  Die Indexstruktur ermöglicht zukünftige Orte von mobilen Objekten zu errechnen.

3.6 Ausschlussregionen

Eine weitere Verbesserung stellen Ausschlussregionen dar [PJ01]. Hierbei handelt es sich um eine Erweiterung der Anfrageverarbeitung, die den Anfragebereich einschränkt.

3.6.1 Beschreibung

Es gibt Orte, an denen sich Objekte normalerweise nicht befinden, z.B. bei Transportern: Seen (ohne Fähre usw.). Bei einer Anfrage in der solche Regionen vorkommen, muss nicht der komplette Bereich abgesucht werden. Die Anfrage kann in kleinere Anfragen aufgeteilt werden.

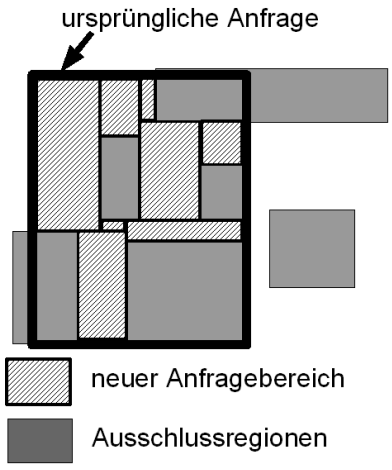


Abbildung 3.10: Ausschlussregionen

Abbildung 3.10 zeigt eine mögliche Aufteilung einer Anfrage über mehreren Ausschlussregionen. Die grauen Rechtecke stellen die Ausschlussregionen

dar. Die ursprüngliche Anfrage (fett umrandet) wird in acht kleinere Anfragen aufgeteilt, wobei die Ausschlussregionen, die im Anfragegebiet liegen, nicht mehr betrachtet werden müssen.

3.7 Zusammenfassung

Ein Überblick über die vorgestellten Indexstrukturen findet sich in Tabelle 3.1.

Indexstruktur	*Besonderheit*	*Vorteile*	*Nachteile*
B-Baum	hierarchische Anordnung von Daten	geeignet für große Datenmengen mit eindimensionaler Ordnung	weniger geeignet für räumliche Daten
R-Baum	Erweiterung des B-Baum zu einem mehrdimensionalen Index	geeignet für räumliche Daten	weniger geeignet für mobile Daten
R*-Baum	geschwindigkeitsoptimierter R-Baum	besser als R-Baum durch Minimierung von Überlappungen	weniger geeignet für mobile Daten
TPR-Baum	Erweiterung des R*-Baums zur Speicherung von mobilen Objekten	geeignet für mobile Daten	verbesserbar (siehe TPR*-Baum [TPS03])

Tabelle 3.1: Überblick Indexstrukturen

Literaturverzeichnis

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 322–331. ACM Press, 1990.

[BM72] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica*, 1(3):173–189, 1972.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[PJ01] Dieter Pfoser and Christian S. Jensen. Querying the trajectories of on-line mobile objects. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 66–73. ACM Press, 2001.

[Sam90] Hanan Samet. *The design and analysis of spatial data structures.* Addison-Wesley Longman Publishing Co., Inc., 1990.

[SJLL02] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing of moving objects for location-based services. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 463. IEEE Computer Society, 2002.

[TPS03] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *Proceedings of the 19th VLDB Conference*, pages 790–801, 2003.

Silke Weiß

4

Umgang mit Unsicherheit

4.1 Einleitung

Datenbanken beweglicher Objekte aktuell zu halten, ist nicht einfach, da sich die Lokationen dieser Objekte ständig ändern und häufiges Aktualisieren ineffizient ist.
Deshalb wird die Position sich bewegender Objekte im MOST-Datenmodell durch das Attribut *L.value* dargestellt, welches den Lokationswert als eine Funktion der Zeit beschreibt (siehe Thema 4). Dieser Wert beschreibt aber nicht genau die tatsächliche Position beweglicher Objekte, da Geschwindigkeitsschwankungen, Änderungen des Bewegungsvektors und die Unsicherheit der Verfügbarkeit des Aktualisierungsmechanismus Differenzen verursachen.
Für viele Transportunternehmen, Krankenhäuser, Flugüberwachungen oder Taxizentralen sind exakte Werte aber wichtig, damit sie einen Überblick über die Verfügbarkeit ihres Fuhrparks haben und planen können.

4.2 Geometrie

Die einzelnen geometrischen Begriffe: Trajektorie, erwartete Position, Karte, Route, Unsicherheitsschwelle, unsicheres Gebiet, mögliche Bewegungskurve und Trajektorievolumen werden in Abbildung 4.1 eingeführt.
Die folgenden Bedingungen sind wichtig für eine spätere Implementierung [TWZC02].

Trajektorie Tr / Bewegungsablauf: beschreibt die Position eines beweglichen Objektes. Das bewegliche Objekt befindet sich z.B. an der Position (x_i, y_i) zum Zeitpunkt t_i und bewegt sich während des Zeitverlaufes $[t_i, t_{i+1}]$ mit konstanter Geschwindigkeit auf einer geraden Linie weiter zu

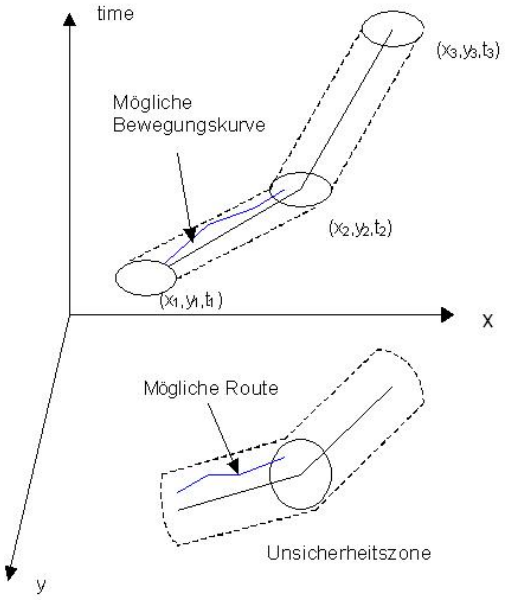


Abbildung 4.1: Die 2D / 3D Geometrie einer Trajektorie

(x_{i+1}, y_{i+1}). Die Trajektorie kann vergangene und zukünftige Bewegungen des Objektes darstellen.
Erwartete Position: es wird angenommen, dass ein Objekt den kürzesten Weg zwischen (x_i, y_i) und (x_{i+1}, y_{i+1}) nimmt.
Karte: eine Karte ist ein Graph.
Route: Anfang- und Endpunkt sind gegeben, Berechnung des Weges mit den geringsten Kosten in der Karte durch eine äußere Routine, welche in den meisten geographischen Informationssystemen vorhanden ist.
Unsichere Trajektorie: (Tr, r) Tr umgeben von r, r ist immer positiv und gibt die Größe der Unsicherheitsschwelle an. Aufenthaltsbereich, in welchem sich das Objekt, laut Datenbank befinden darf. Wenn das Objekt diesen Aufenthaltsbereich verlässt, muss es sich melden (im Bild 4.1 oben, der ganze Schlauch).
Unsicheres Gebiet: die Tr umgeben von r, r ist der Radius, um welchen das Objekt von der Position (x, y) abweichen darf (im Bild 4.1 unten).
Mögliche Bewegungskurve: stellt mögliche Routen dar, welche ein Objekt nehmen kann, ohne eine Aktualisierung tätigen zu müssen, d.h. das bewegliche Objekt muss sich im unsicheren Gebiet befinden.
Trajektorievolumen: sind alle Punkte (x_i, y_i) die im unsicheren Gebiet liegen. Das Trajektorievolumen wird in 2D Unsicherheitszone genannt.

4.3 MOST / FTL bei Unsicherheit

Das MOST-Datenmodell wird um das Attribut *L.maxDeviation* erweitert, welches die Größe der Abweichung angibt. Dies bedeutet, dass bei einer Anfrage des Standortes eines beweglichen Objekts nicht nur der Datenbankeintrag *L.value* ausgegeben wird, sondern auch die Abweichung, in welcher sich das Objekt bewegen kann.
Die Abweichung ist die Differenz zwischen dem tatsächlichen Wert des beweglichen Objektes zum Zeitpunkt t und dem Lokationswert der in der Datenbank zum Zeitpunkt t gespeichert ist. Eine Aktualisierung von *L.value* wird erst vorgenommen, wenn die Abweichung von *L.value* zum tatsächlichen Lokationswert *L.maxDeviation* übersteigt, siehe Abbildung 4.5.
Häufige Aktualisierungen sind die Folge einer zu niedrig gewählten Grenze, eine zu hoch angesetzte Grenze verursacht größere Unsicherheit, da Unsicherheit im Laufe der Zeit steigt. Ungenauigkeit wird in Kauf genommen, um häufige Aktualisierungen zu vermeiden, da diese hohe Kosten verursachen. Ein optimaler Wert *L.maxDeviation* sorgt für ein gutes Verhältnis zwischen der Ungenauigkeit und den Aktualisierungen. In der Praxis ist das Finden eines solchen Wertes jedoch schwierig, wie später in Abschnitt 4.5 erkennbar.

Die Muss-Semantik und Kann-Semantik sind Erweiterungen der FTL-Anfragesprache, sie werden durch die Schlüsselworte definitely / possibly ausgedrückt. Während bei der Muss-Semantik der Bewegungsablauf des beweglichen Objektes komplett im Anfragebereich liegen muss, reicht es bei der Kann-Semantik aus, wenn das bewegliche Objekte den Anfragebereich schneidet. Abbildung 4.2 verdeutlicht dies nochmals: Der Anfrager mit possibly erlaubt “mögliche“ Ergebnisobjekte, während der Anfrager mit definitely nur „sichere“ Ergebnisobjekte zulässt, deshalb müssen hier alle Ergebnisobjekte exakt im Unsicherheitsintervall liegen.

Ein Anfragebeispiel, welches die Muss-Semantik ausdrückt, lautet:

- Welche LKWs werden sich **mit Sicherheit** innerhalb der nächsten 5 Minuten dem Informatikgebäude am Fasanengarten auf 2 km annähern.

Dies wird in FTL wie folgt abgefragt:

```
Retrieve␣LKW
Where␣definitely_eventually_within_5min
DIST(LKW,␣Informatikgeb{\"a}ude)␣<=␣2km)
```

Bei der Kann-Semantik lautet dies entsprechend:

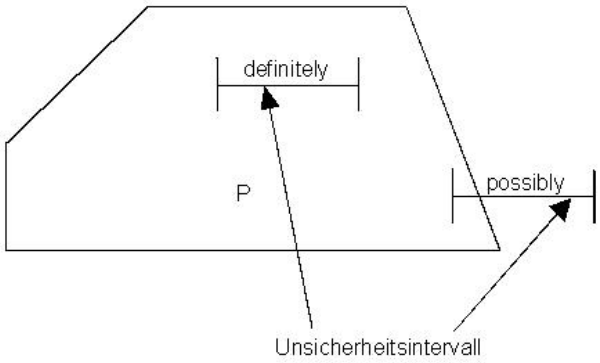


Abbildung 4.2: Muss-, Kann-Semantik: P ist der Anfragebereich, welchen das Objekt bei der Muss-Sematik nicht verlassen darf, wenn es als Lösung ausgegeben werden soll, bei der Kann-Sematik werden auch Objekte ausgegeben, welche den Anfragebereich nur schneiden.

- Welche LKWs werden sich **möglicherweise** innerhalb der nächsten 5 Minuten dem Informatikgebäude am Fasanengarten auf 2 km annähern.

In FTL:

```
Retrieve␣LKW
Where␣possibly_eventually_within_5min
(DIST(LKW,␣Informatikgeb{\"a}ude)␣<=␣2km)
```

4.3.1 Räumlich, zeitliche Operatoren

In Abbildung 4.3 sind alle räumlich-zeitlichen Operatoren graphisch dargestellt. Die einzelnen Bilder zeigen jeweils eine mögliche Bewegungskurve die den Operator erfüllt. Die Region R begrenzt den Anfragebereich und das bewegliche Objekt darf sich in (Tr, r) aufhalten.
In Folgendem wird beschrieben, welche Bedingungen erfüllt sein müssen, um die einzelnen Operatoren zu realisieren [TWZC02].

Possibly_Sometime_Inside $((Tr, r), R, t_1, t_2)$**:**
Es existiert ein Aufenthaltsort, der die Anfrageregion zum Zeitpunkt $t \in [t_1, t_2]$ schneidet.
Sometime_Possibly_Inside $((Tr, r), R, t_1, t_2)$ ist analog.

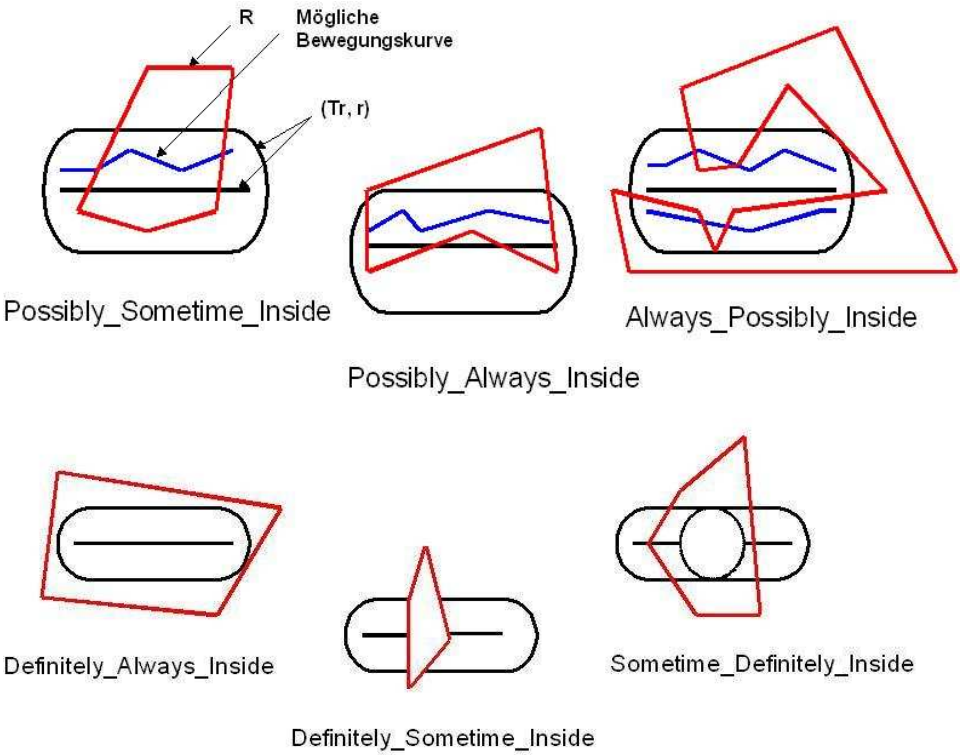


Abbildung 4.3: Operatoren für räumlich, zeitliche Anfragen

Possibly_Always_Inside $((Tr, r), R, t_1, t_2)$**:**
Zu jedem Zeitpunkt $t \in [t_1, t_2]$ gibt es eine Folge von Aufenthaltspunkten, die alle im Anfragebereich liegen.

Always_Possibly_Inside $((Tr, r), R, t_1, t_2)$**:**
Zu jedem Zeitpunkt $t \in [t_1, t_2]$ muss ein Aufenthaltsort existiert, der in der Anfrageregion liegt.
Ist Possibly_Always_Inside $((Tr, r), R, t_1, t_2)$ erfüllt, so ist auch Always_Possibly_Inside $((Tr, r), R, t_1, t_2)$ wahr. Eine Umkehrung dieser Eigenschaft gilt nur bei konvexem Anfragebereich.

Always_Definitely_Inside $((Tr, r), R, t_1, t_2)$**:**
Zu jeder Zeit $t \in [t_1, t_2]$ muss der Aufenthaltsort des Objektes im Anfragebereich sein.
Definitely_Always_Inside $((Tr, r), R, t_1, t_2)$ ist analog.

Definitely_Sometime_Inside $((Tr, r), R, t_1, t_2)$**:**
Es existiert ein Zeitpunkt $t \in [t_1, t_2]$, zu welchem der Aufenthaltsort in der Anfrageregion liegt.

Sometime_Definitely_Inside $((Tr, r), R, t_1, t_2)$**:**

Es existiert ein Zeitpunkt $t \in [t_1, t_2]$, zu welchem jeder mögliche Aufenthaltsort des Objektes im Anfragebereich liegt.

Wenn Sometime_Definitely_Inside $((Tr, r), R, t_1, t_2)$, wahr ist, so ist auch Definitely_Sometime_Inside $((Tr, r), R, t_1, t_2)$ erfüllt, Umkehrung ist analog oben.

Die Beziehungen der Operatoren sind in Abbildung 4.4 zusammengefasst. Always_Definitely_Inside ist die einschränkenste Operation. Wenn sie erfüllt

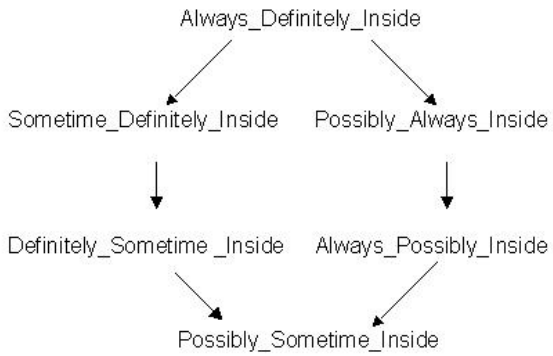


Abbildung 4.4: Beziehungen der räumlich-zeitlichen Operatoren

ist, sind auch alle anderen Operatoren erfüllt. Die Pfeile können als Folgerung interpretiert werden, d.h. wenn die obere Operation wahr ist, folgt daraus, dass die danach folgenden Operatoren auch erfüllt sind.

4.4 Indexierung bei Unsicherheit

Indexierung wird benötigt, um effizient nach beweglichen Objekten in einer Datenbank suchen zu können, so dass nicht alle Lokationen untersucht werden müssen. Sie hilft bei der Konstruktion eines geeigneten Raumes, damit die Schnittmenge möglichst effizient gefunden werden kann. In Kapitel 5 wurde die Indexierung ohne Unsicherheit mit den Indices (x, t, v) betrachtet, in folgendem wird die Indexierung mit Unsicherheit beschrieben, die Indices werden um den Schwellenwert s erweitert. Dieser Schwellenwert s ermöglicht die Berücksichtigung der jeweiligen Unsicherheitspräferenzen.
Abbildung 4.5 zeigt eine Value Time Indexierung, welche effizient bei häufigen Anfragen ist [WXCJ98]. Das bewegliche Objekt wird durch die O-Fläche repräsentiert. Das Objekt fährt entlang der y-Achse, (y_0, y_1) stellt das Un-

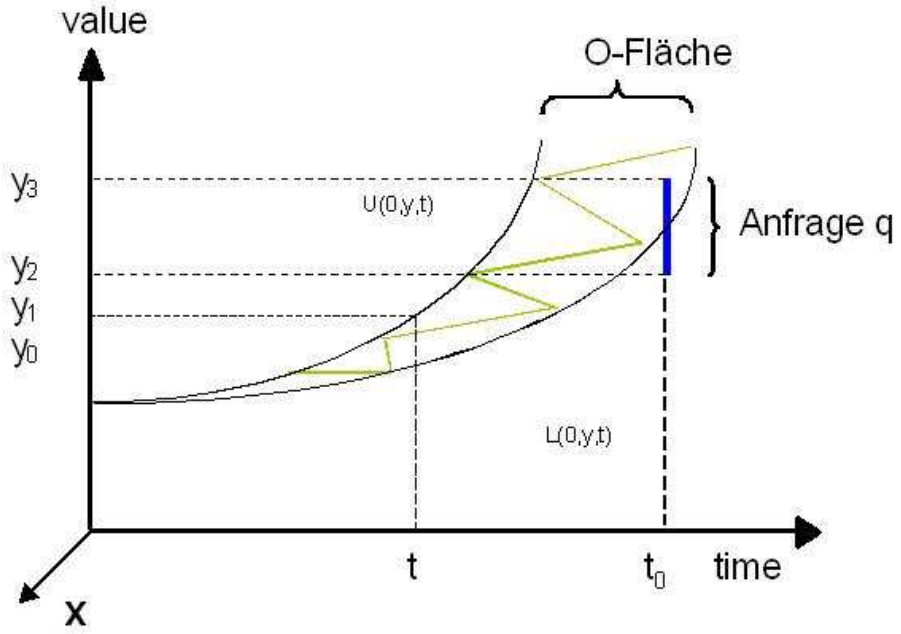


Abbildung 4.5: Value Time Indexierung

sicherheitsintervall zum Zeitpunkt t dar. Der Anfragebereich q ist durch die dicke senkrechte Linie zum Anfragezeitpunkt t_0 gekennzeichnet.
Mögliche Anfragen lauten:
1. Welche LKWs werden sich **mit Sicherheit** zum Zeitpunkt t_0, zwischen y_2 und y_3 befinden.
2. Welche LKWs werden sich **möglicherweise** zum Zeitpunkt t_0, zwischen y_2 und y_3 bewegen.
Bei der ersten Anfrage muss q den Unsicherheitsbereich des Objektes völlig überdecken, während bei der zweiten Anfrage ausreicht, wenn q das Objekt schneidet.
Hier dargestellt: Kann-Semantik, da der Anfragebereich das Unsicherheitsintervall des Objektes nur schneidet.

4.5 Bestimmung der Grenze L.maxDeviation

Durch Minimierung der Informationskosten wird ein optimaler Wert der Grenze *L.maxDeviaton* berechnet.
Die Informationskosten entstehen durch Addition der Abweichungs-, der Aktualisierungs- und der Unsicherheitskosten.
Die Abweichungskosten bestehen aus der Größe und der Dauer der Abweichung. Es sind die Kosten, die durch falsche Entscheidungsfindung entste-

hen. Die Abweichung ist die Distanz zwischen tatsächlichem Lokationswert und dem Lokationswert der in der Datenbank gespeicherten ist ($L.value$), sie kann höchstens den Wert $L.maxDeviation$ annehmen. Es ist erkennbar, dass eine kleine Schranke $L.maxDeviation$ geringere Abweichungskosten verursacht als eine groß Schranke, da das Objekt einen eingeschränkteren Abweichungsspielraum hat.
Aktualisierungskosten entstehen beim Versenden aktueller Lokationsdaten, es sind z.B Ressourcekosten. Häufiges Aktualisieren, welches durch Wahl einer kleinen Schranke $L.maxDeviation$ begünstigt wird, verursacht hohe Aktualisierungskosten. Deshalb sollte $L.maxDeviation$ im Gegensatz zu oben, bei Minimierung der Aktualisierungskosten möglichst groß gewählt werden.
Die Unsicherheitskosten hängen von der Größe und Dauer der Unsicherheit ab. Sie entstehen, da höhere Unsicherheit weniger Information dem Anfragenden überbringt. Zudem werden die Unsicherheitskosten noch gewichtet. Ein Gewichtungsfaktor über 1 führt zu hohen Unsicherheitskosten, wenn der Faktor kleiner 1 gewählt wird sinken sie. Hier spielt die Wahl des Gewichtungsfaktor eine größere Rolle, als die Wahl des Wertes $L.maxDeviation$. Da kleine Werte für $L.maxDeviation$ durch einen Gewichtungsfaktor größer 1, große Unsicherheitskosten verursacht, wiederum verursacht ein hoher Wert $L.maxDeviation$ bewertet durch einen Gewichtungsfaktor kleiner 1 geringere Unsicherheitskosten.
Zusammenfassend, ist aus obigem ersichtlich, dass es schwierig ist, einen optimalen Wert für $L.maxDeviation$ zu finden, da sich $L.maxDeviation$ unterschiedlich auf die einzelnen Kosten auswirkt.

In Abbildung 4.5 ist die anpassbare Abschätzung graphisch dargestellt. Folgendes Szenario beschreibt ein solches Verhalten:
Ein LKW- Fahrer fährt z.B. von Stuttgart nach Karlsruhe. $L.maxDeviation$ und seine Geschwindigkeit wurden vor Fahrtbeginn festgelegt. Eine geänderte Verkehrssituation erlaubt dem LKW-Fahrer schon nach kurzer Zeit eine höhere Geschwindigkeit als die vorher festgelegte. Dies wird im Schaubild durch die höhere Steigung dargestellt. Wegen der veränderten Geschwindigkeit stößt der LKW nach einiger Zeit an die Grenze $L.maxDeviaton$, dieses Ereignis ist durch den Pfeil markiert. Der LKW-Fahrer muss nun seine Fahrdaten aktualisieren und $L.maxDeviation$ neu berechnen.

4.5.1 Aktualisierungsstrategien

Es gibt drei verschiedene Aktualisierungsstrategien [Obr]. Die Geschwindigkeitsabschätzung, die anpassbare Abschätzung und die Abschätzung mit Erkennung von Verbindungsabbrüchen. Sie unterscheiden sich durch unterschiedliche Festlegung des Wertes $L.maxDeviation$.
Bei der Geschwindigkeitsabschätzung wird vor Beginn der Tour $L.maxDeviation$ einmal festgelegt und nicht mehr verändert.

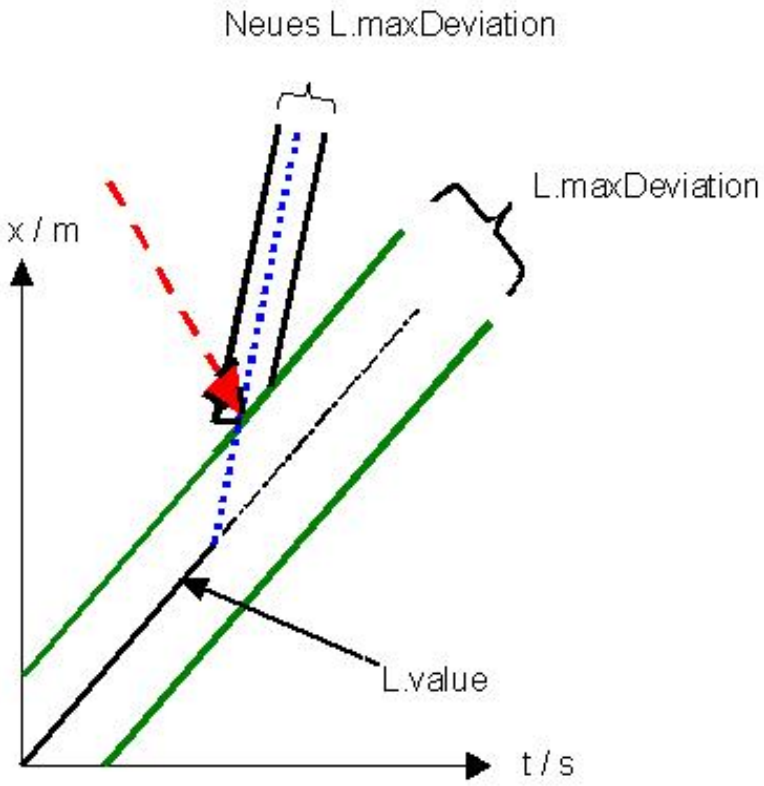


Abbildung 4.6: Anpassbare Abschätzung: *L.maxDeviation* änderbar

Bei der anpassbaren Abschätzung wird *L.maxDeviation* bei Erreichung der Grenze immer wieder neu berechnet. *L.maxDeviation* ist nach jeder Aktualisierung optimal. Die anpassbare Abschätzung ist besser als die Geschwindigkeitsabschätzung und die Erkennung von Verbindungsabbrüchen, da sie durch die ständige Anpassung die geringsten Informationskosten erzeugt.
Die Erkennung von Verbindungsabbrüchen zeichnet sich dadurch aus, dass sich *L.maxDeviation* kontinuierlich verringert, so dass sich nach einem bestimmten Zeitablauf das bewegliche Objekt auf jeden Fall melden müsste. Wenn diese Aktualisierung ausbleibt, schließt die Datenbank daraus, dass das Objekt sich nicht melden kann, weil es sich z.B. in einem Funkloch befindet.

4.6 Domino-Prototyp

Die Implementierung der bisher erklärten Probleme wurde im Domino-Prototyp verwirklicht [WXCJ98]. Hierbei wurde folgendermaßen vorgegangen:
Das MOST-Datenmodell und die FTL-Anfragesprache wurden auf Sybase oder MS-Access aufgesetzt, somit wurde das Datenbankmanagementsystem um das MOST-Datenmodell und die FTL-Anfragesprache ergänzt. DOMINO arbeitet auf einer zentralen Datenbank. Über eine graphische Benutzeroberfläche können Anfragen und Triggers eingegeben werden. Außerdem können die Ergebnisse visualisiert werden. Die Triggers, Anfragen und Antworten werden vom MOST- und FTL-System untersucht und eventuell verändert, falls dies notwendig sein sollte. Triggers sind Ereignisauslöser, sie können die lokale Datenbank feuern und aktualisieren, sobald die Abweichung erreicht wurde. Zudem sorgen sie dafür, dass $L.maxDeviation$ neu berechnet wird. Über Datenbanktriggers werden Aktualisierungsstrategien ermöglicht. Die Ermittlung der tatsächlichen Position des Objektes erfolgt über GPS.
Im Prototyp wurden verschiedene Aktualisierungsstrategien implementiert, es können aber auch neue Strategien definiert werden.
Außerdem können die verschiedenen Aktualisierungsstrategien analysiert werden, um spätere Verbesserungen realisieren zu können. Über eine graphische Benutzeroberfläche kann eine Geschwindigkeitskurve eingegeben werden. Danach werden verschiedene Aktualisierungsstrategien simuliert bevor das System die Informationskosten, die Anzahl der Aktualisierungen, die Abweichung und die Unsicherheit für die angegebenen Aktualisierungsstrategien berechnet. Zudem werden Tools für die Verknüpfung und die graphische Ausgabe der Ergebnisse bereitgestellt.
Abbildung 4.7 zeigt die graphische Benutzeroberfläche vom DOMINO - Prototyp. Sie zeigt drei Trajektories in Cook Country, Illinois und einen Anfragebereich, welcher im Bild grau markiert ist. Dieser Anfragebereich wird erst nach der Anfrageeingabe des Endbenutzers angezeigt. Die Anfrage lautet hier:

```
Retrieve␣Objekt
Where␣Possibly_Sometime_Inside_the␣Region
R␣[(y_i,␣y_i+1),(12:15,12:30)]
```

Jede Trajektories entspricht einer Route mit verschiedenen Haltepunkten. Die Indices neben den Haltestellen geben Eintreffzeitpunkt und Verweildauer an. Von den drei Tajektories, erfüllt nur eine das Anfragekriterium, diese Trajektorie ist durch einen Kreis markiert. Die anderen beiden schneiden den Anfragebereich entweder nicht oder zur falschen Zeit[TWZC02].

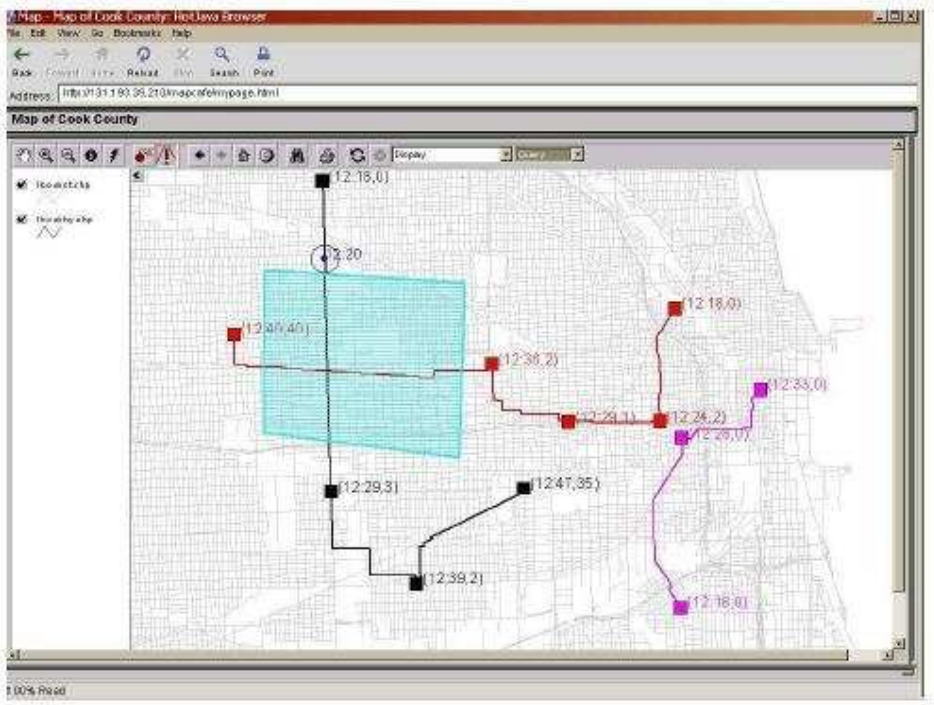


Abbildung 4.7: Graphische Benutzeroberfläche von DOMINO

4.7 Weitere Konzepte

In diesem Kapitel gilt folgendes neues Szenario:
Ein LKW-Fahrer fährt auf der A8 von Karlsruhe nach Stuttgart. Er hat ein intelligentes Computersystem an Bord, welches die Kommunikationsschnittstelle zwischen LKW und Kontrollcenter darstellt.
1. Anfrage: **Wo** gibt es Rastplätze?
Der Computer gibt daraufhin alle Rastplätze mit Restaurant auf dieser Strecke aus.
2. Anfrage: **Wann** ist die Rast am besten? Vor oder nach der nächsten Stadt, um Verkehrsandrang zu vermeiden.
Um diese Anfrage beantworten zu können, muss das Kontrollcenter aktuelle Informationen über die zukünftige Verkehrslage haben, um Vorhersagen ausführen zu können. Das Kontrollcenter kann dies aber nur, wenn sich alle Verkehrsteilnehmer, bevor sie den Service anfragen, registrieren, d.h. ihren Fahrplan angeben, ansonsten ist eine Vorhersage schwierig.
Die beweglichen Objekte sind Abbildung 4.8 als Linienabschnitte gezeichnet. Das Rechteck stellt die Anfrageregion dar, hier ist R $([t_1, t_2] * [y_1, y_2])$.
Eine mögliche Anfrage lautet:
Gebe alle Objekte aus, welche in einem bestimmten Bereich in der Zukunft

sind, hier: gebe alle Objekte aus, welche zwischen t_1 und t_2 sich im Bereich $[y_1, y_2]$ befinden.
Bevor Anfragen an die Datenbank gestellt werden, muss noch die Annahme getroffen werden, dass bewegliche Objekt ihre Änderungen selbst anzeigen.

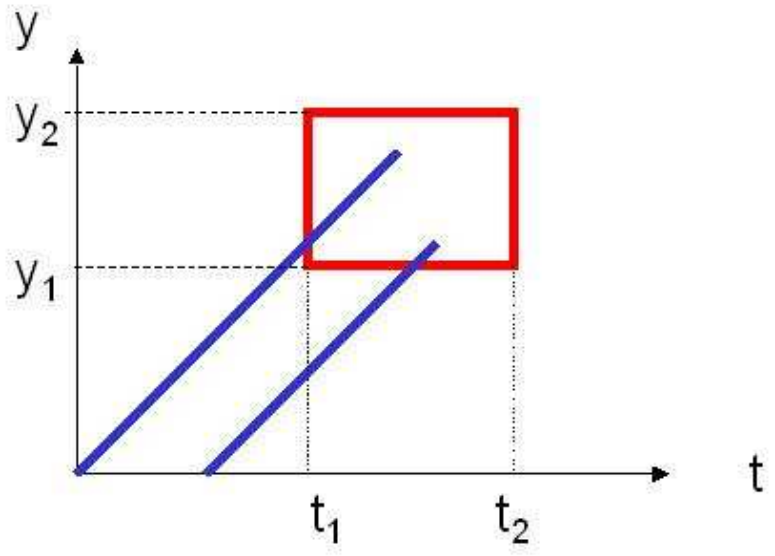


Abbildung 4.8: Anfrage an zukünftige Verkehrslage: Darstellung der Objekte als Linienabschnitte, Anfragebereich $R = ([t_1, t_2] * [y_1, y_2])$

Bei dieser Modellierung spielen die folgenden vier unabhängigen Variablen des beweglichen Objektes eine große Rolle:

1. Startposition (s)
2. Ziel (e)
3. Startzeit (t_s)
4. Anfangsgeschwindigkeit (v_s)

Es gibt zwei Modelle, durch welche die bestmögliche Konfiguration der Variablen gefunden werden kann, um eine effiziente Indexstruktur zu erhalten. Das erste Modell ist das so genannte Modell mit 2 Freiheitsgraden, das zweite Modell das SV-Modell [DCAEA01].

4.7.1 Modell mit 2 Freiheitsgraden

Beim Modell mit 2 Freiheitsgraden können zwei der vier Variablen variieren, die anderen Variablen sind fest. Es können daher sechs verschiedene Kombinationen der vier Variablen erzeugt werden.

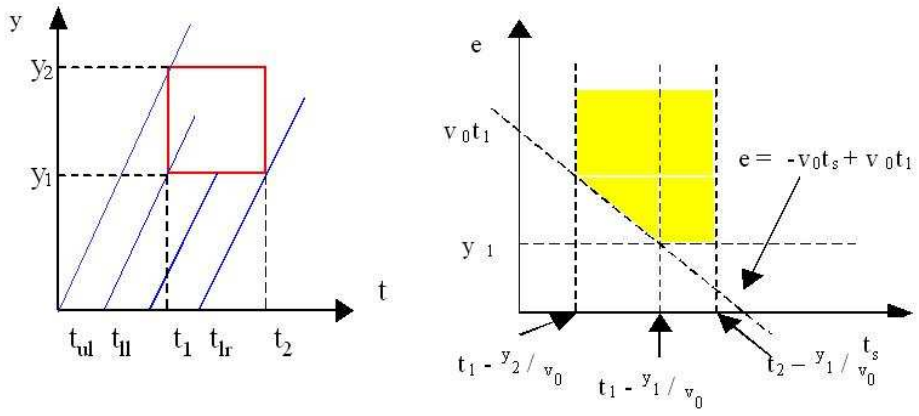


Abbildung 4.9: Modell mit 2 Freiheitsgraden

Im nächsten Abschnitt wird nun die Kombination aus Ziel(e) und Startzeit (t_s) variieren, Startposition(s)($s = 0$) und Anfangsgeschwindigkeit (v_s) sind konstant, beschrieben.
Abbildung 4.9 zeigt im linken Bild den Originalraum und im rechten Bild das Ergebnis des Modells. Durch folgende Rechnung wird der Originalraum auf den Ergebnisraum abgebildet.
Betrachte das linke Bild. Die beweglichen Objekte starten alle an der Startposition y_0 und sie haben alle dieselbe Steigung, da die Geschwindigkeit konstant ist. Nun wird berechnet, welche Bedingungen die Objekte erfüllen müssen um das Anfragerechteck (Region in der Zukunft) zu schneiden. t_{ul}, ist der t - Achsenabschnittspunkt der Linie, welches den Punkt (t_1, y_2) des Anfragerechteckes schneidet und t_{lr} ist der t - Achsenabschnittspunkt der Linie, welche durch den Punkt (t_2, y_1) geht.
Wenn das Anfragerechteck geschnitten werden soll, muss der t - Achsenabschnittspunkt größer oder gleich t_{ul} und kleiner oder gleich t_{lr} sein. Da Objekte die zu einem früheren oder späteren Zeitpunkt starten das Anfragerechteck nicht mehr schneiden.
Indexierung erfolgt nach Jagadisch[DCAEA01] durch die so genannte Gleichung der unendlichen Linie sie wird um das Liniensegment, welches durch den Punkt (t_1, y_2) geht erweitert. Daraus folgt $y - y_2 = v_0(t - t_1)$ tausche nun (t, y) gegen $(t_{ul}, 0)$ mit $(s = 0)$, daraus folgt $0 - y_2 = v_0(t_{ul} - t_1)$ durch

Umstellung wird $t_{ul} = t_1 - y_2/v_0$ erzeugt, dies ist die Ergebnisgerade, sie ist im linken Bild eingezeichnet. Analoge Anwendung auf den Punkt (t_1, y_1) ergibt $t_{lr} = t_1 - y_1/v_0$.
Betrachte nun den Zielort(e) genauer, t_{ll} und t_{lr} sind die Liniensegmente, welche die horizontale Linie des Anfragerechteckes zwischen t_1 und t_2 schneiden. Der Zielort muss größer als y_1 sein, diese Bedingung reicht aus um das Rechteck zu schneiden. Wenn y_1 nicht geschnitten wird, so wird der Anfragebereich nicht erreicht. Auch diese Gerade y_1 schränkt den Ergebnisraum ein und ist somit im linken Bild eingezeichnet.
Zum Schluss wird noch die vertikale Linie t_1 des Anfragerechteckes zwischen t_{ul}und t_{ll} untersucht. Um diese Linie zu schneiden, muss der Startzeitpunkt des Objektes zwischen t_{ul}und t_{ll} liegen. Auch hier wird analog oben indexiert, die Gleichung der unendlichen Linie wird um das Liniensegment erweitert, welches (t_1, e) schneidet, daraus folgt $y - e = v_0(t - t_1)$ nun wird (t, y) gegen $(t_s, 0)$ getauscht und der Wert e der Linie am Punkt t_1 ist $e = v_0 * t_s + v_0 * t_1$. Wenn wir die bisherigen Geraden um diese erweitern erhalten wir den kompletten Ergebnisraum der Anfrage. Er ist nicht ganz rechteckig, dies veranlaßt zur Kritik an diesem Modell, da rechteckige Anfragebereiche für die Bäume aus Kapitel 5 wichtig sind.

4.7.2 SV-Modell

Beim SV Modell werden alle Variablen analysiert, um eine angemessene Konfiguration und Indexstruktur zu bekommen. In Folgendem wird eine angemessene Konfiguration unseres Anfangsbeispiels erstellt. Die Startposition(s) kann ohne Einschränkung des Problems auf s = 0 gesetzt werden, da durch Transformation der Startzeit(t_s) nach $newT_s = -s/(v_0 + t_s)$ die beweglichen Objekte mit den verschiedenen Startpositionen so repräsentiert werden, als würden sie bei $s = 0$ beginnen. Diese Transformation verursacht jedoch Overhead.
Erklärung anhand folgendem Beispiel: Annahme, Objekt m mit $s = 10$ wird durch $newT_s = 0$, transformiert auf $s = 0$, nun sollen alle Objekte ausgegeben werden, welche eine Startposition zwischen 3 und 5 haben.
Das Objekt m ist Teil der Lösung der Anfrage, obwohl m eigentlich noch gar nicht existiert. Deshalb wird Filtertechnik bei Anfragen an die Vergangenheit benötigt.
Eine Konvertierung von variierendem Ziel zu festem Ziel verursacht ein noch komplexeres Problem, deshalb lässt man das Ziel variabel, dies setzt aber voraus, dass die Anfangsgeschwindigkeit v_0 konstant gesetzt werden muss. Dies ist aber keine große Beschränkung unseres Problems, da Autos nur eine begrenzte Anzahl an Geschwindigkeiten annehmen können. Es werden mehrere Instanzen von fixen Geschwindigkeiten gebildet. Die Objekte werden dann den jeweiligen Instanzen zugeteilt. Sollte ein Objekt in keine Instanz

passen, wird es in die ähnlichste Instanz gepackt und einfach öfters aktualisiert.
Das Ergebnis des Beispieles ist die Kombination, welche beim Modell mit 2 Freiheitsgraden beschrieben wurde. Es ist einem Rechteck am ähnlichsten.

4.8 Zusammenfassung

Wie bereits erwähnt, müssen Benutzer von Datenbanken, in welchen Lokationen beweglicher Objekte gespeichert werden, mit Ungenauigkeit rechnen. Die Datenbank ermöglicht aber durch das Attribut $L.maxDeviation$ eine gewisse Einschränkung der Unsicherheit auf ein bestimmtes Gebiet. $L.maxDeviation$ kann durch die Wahl der verschiedenen Aktualisierungsstrategien: Geschwindigkeitsabschätzung, anpassbare Abschätzung und Erkennung von Verbindungsabbrüchen individuell aktualisiert werden. Ein optimaler Wert für $L.maxDeviation$ wird durch die Minimierung der Informationskosten berechnet. Außerdem können die Anfrager durch die verschiedenen Anfrageoperatoren aus Abschnitt 4.3.1, den Genauigkeitsgrad der Antwort festlegen. Bei der Indexierung spiegelt sich die Unsicherheit im Schwellenwert s wieder, welcher die Objektfläche vergrößert. Anfragen an zukünftige Verkehrssituationen haben wir in Abschnitt 4.7 kennen gelernt, wichtig hierbei ist Registrierung, da ansonsten Vorhersagen schwierig getätigt werden können. Das Modell mit 2 Freiheitsgraden und das SV-Modell hilft bei der Berechnung einer geeigneten Indexstruktur.

4.9 Bewertung und Ausblick

Wie mit Unsicherheit umgegangen wird liegt letztendlich beim Klienten der Datenbank. Wieviel Wert er auf Sicherheit legt, kann er wie oben erwähnt, durch die Schranke $L.maxDeviation$ festlegen und auch auf die Informationskosten kann er durch die Wahl des Gewichtungsfaktors Einfluss nehmen.
Weitere Anwendungsgebiete sind die Erstellung digitaler Schlachtfelder, Wettervorhersagen, Touristenservice, Mobile Telefonbenutzer, Transport- und Flugüberwachungen. Durch Abfrage zukünftiger Positionen können z.B. nicht feindliche Helikopter erkannt werden, Vorhersagen für den Zeitbedarf der Strecke von Hotel zum Flughafen können ein termingerechtes Einchecken erleichtern usw.

Literaturverzeichnis

[DCAEA01] Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Storage and retrieval of moving objects. pages 173–184. Springer Verlag, 2001.

[Obr] Philipp Obreiter. Bewegliche objekte in datenbanken. pages 79–99.

[TWZC02] Goce Trajcevski, Ouri Wolfson, Fengli Zhang, and Sam Chamberlain. The geometry of uncertainty in moving objects databases, 2002.

[WXCJ98] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. pages 111–122, Juli 1998.

Teil III

P2P-Datenquellen

Daniel Steudel

5

Weltweit verteilte Datenbanken

5.1 Einleitung

5.1.1 Die Situation

In der heutigen Informationstechnologie (IT) sind Datenbank-Management-Systeme (DBMS) zu einem sehr wichtigen Bestandteil geworden. Neben großen, kommerziellen Anbietern wie Oracle, IBM oder Microsoft haben sich mittlerweile auch freie Systeme wie PostgreSQL und MySql durchgesetzt. Wenn man versucht zu erklären, warum DBMS eine derart wichtige Rolle in der IT spielen, werden Themen wie Ausfallsicherheit und langfristige Speicherung dazu gehören.

Durch die zunehmende Globalisierung werden aber neben diesen grundlegenden Eigenschaften auch andere Funktionen wichtig. Es ist keine Seltenheit, dass selbst mittelständische Unternehmen eine Außenstelle in Amerika haben. Auch neue betriebswirtschaftliche Methoden wie Supply Chain Management (SCM) fordern immer mehr Leistung von Datenbanken, die die schon angesprochenen, lokalen Datenbanken leisten können.

Weltweite Verteilung von Datenspeichern ist aber nicht nur aus unternehmerischer Sicht eine interessante Perspektive und wünschenswert, sondern betrifft auch den „Normalbürger“ in einer immer mobiler werdenden Gesellschaft. Es wird immer wichtiger, Daten am heimischen Computer abzuspeichern und diese dann an einem anderen Ort wieder abzurufen, ohne auf Datenträger zurückgreifen zu müssen.

5.1.2 OceanStore

Die von der University of California in Berkley implementierte, weltweit verteilte Datenbank trägt den Namen OceanStore und ist eine Referenzentwicklung die sich derzeit noch im Aufbau befindet. Aktuell sind laut [Kub03] 98 Computer in 42 Institutionen in Nordamerika, Australien und Europa im Einsatz. Jedoch konzentriert man sich auf die Skalierbarkeit und die Ausfallsicherheit des Systems.

5.2 Verteilung von Daten weltweit

5.2.1 Was wird benötigt

Das weltweite Verteilen von Daten stellt an die Entwicklung besondere Ansprüche und enthält einige Hürden, die die Entwicklungen lokaler Datenbanken nicht kennen. In der lokalen Welt besitzen sie einen zentralen Rechner, auf den sie im besten Fall sogar physisch zugreifen können. Dieser Rechner besitzt eine Platte, auch wenn diese eine Platte aus einem Redundant Array of Independant[1] Disks (RAID) besteht. Im Normalfall ist der Rechner in ein Local Area Network (LAN) eingebunden, von dem aus Anwendungen und Endbenutzer auf den Datenbestand zugreifen. Diese Voraussetzungen bestehen bei der weltweiten Verteilung nicht. Es ist sogar wünschenswert, dass man nicht einmal Rechner, die an dem System teilnehmen kennen oder sogar vertrauen muss.

Da man, wie gesagt, nicht auf die Annehmlichkeiten verzichten will, sollte die Datenbank Lokalität der Daten gewährleisten; das aber an jedem Ort weltweit. Des Weiteren muss sie Ausfall- und Datensicherheit trotz unsicherer Komponenten bieten. Natürlich ist es ebenfalls wichtig, selbst bei konkurrierenden Zugriffen die semantische und syntaktische Korrektheit zu garantieren. Wenn man sich den Katalog der Forderungen ansieht, wirkt ein solches System unmöglich.

5.2.2 Identifikation der Daten

Die erste und grundlegendste Aufgabe besteht nun darin, die gespeicherten Daten weltweit eindeutig identifizierbar zu machen. Dazu sollte zuerst darüber nachgedacht werden, wie viele Daten man eigentlich eindeutig benennen muss, um herauszufinden wie genau dieser Name auszusehen hat. Wenn wir davon ausgehen, dass wir weltweit ungefähr 10^{10} Benutzer zu

[1] Die frühere Bezeichnung lautet „Inexpensive“, da die ursprüngliche Idee war Kosten durch preiswertere Hardware zu senken und trotzdem die Sicherheit zu erhöhen

bewältigen haben, von denen jeder einzelne 1.000 Datenobjekte speichern will, kommen wir auf eine Gesamtzahl von 10^{14} Objekten.

Eine Möglichkeit bestünde nun in der zentralen Vergabe von fortlaufenden Nummern. Dieser Ansatz ist jedoch unpraktikabel, da wir dafür spezielle Server in der Systemarchitektur bereitstellen müssen, die diese Aufgabe erfüllen. Zum einen schränkt das natürlich die Ausfallsicherheit des gesamten Systems ein und außerdem hätte man damit nur den „schwarzen Peter“ an den Server für die Nummernvergabe weitergereicht. Dem obliegt es in diesem Ansatz die weltweite Eindeutigkeit zu wahren.

Besser ist es, einen Raum zu definieren, in dem jeder Server von sich aus einen eindeutigen Namen finden kann. Die Frage, die uns nun beschäftigt ist: Wie muss ein solcher Raum aussehen?

Mit Sicherheit ist es sinnvoll die Menge der Binärzahlen als Grundmenge zu benutzen. Weiterhin muss der Raum so groß sein, dass es auch mit einem relativ simplen Algorithmus möglich ist, einen eindeutigen Namen zu finden. Würden wir Namen mit variabler Länge zulassen, hätten wir das Problem herauszufinden, ob es in unserem Universum ein Präfix unseres Objektes gibt, oder ob wir einen Präfix eines anderen Objekts gefunden haben. Daraus ergibt sich ein Aufwand von $o(log\ n)$ bei einer optimalen Datenstruktur. Wahrscheinlicher ist jedoch ein Aufwand von $O(n^2)$.

Es ist offensichtlich, dass man sich damit unnötige Komplexität auferlegt. Viel besser wäre ein Raum aus Binärzahlen fester Länge. Damit vereinfachen wir unser Problem dazu eindeutige Zahlen in einer Menge von Zahlen zu finden, dass allgemein als Hashing bekannt ist. Dafür gibt es schon einige Algorithmen wie zum Beispiel den SHA-1 Algorithmus. Dieser generiert aus einer Datenmenge variabler Länge einen Datenblock fester Länge. Laut [Kub01] besitzt ein mit SHA-1 generierter Name von 160 Bit eine Wahrscheinlichkeit von 20^{-20}, dass zwei Objekte denselben Schlüssel bekommen. Dieser Algorithmus ist effizient und kann unabhängig berechnet werden.

In OceanStore werden als Eingabedaten für den Algorithmus die Objektdaten benutzt, die zuvor vom Benutzer verschlüsselt worden sind. Diese Methode besitzt demzufolge einige Seiteneffekte. So ändert sich der Name eines Objektes sobald sich die Daten ändern, was weiterhin eine Verwaltung der veralteten Objekte nach sich zieht.

5.2.3 Finden des Speicherorts

Obwohl nun bekannt ist, wie die Objekte eindeutig benannt werden, besteht ein weiteres Problem darin herauszufinden, wo die Objekte gespeichert oder wieder gefunden werden. Wie bereits bei der Suche nach einem geeigneten Algorithmus für die Namensvergabe, ist es auch beim Routing wünschens-

wert, dass Knoten autark entscheiden können, welcher Server das Objekt besitzt und wohin eine Nachricht weitergeleitet werden muss.

Ein Ansatz, der in vielen Systemarchitekturen verfolgt wird ist die Verwendung eines Verzeichnisdienstes, der zentral die Zuordnung von Objektname und Speicherort verwaltet. Dieser Ansatz ist jedoch wenig praktikabel, da erstens eine sehr hohe Last auf wenigen Servern entsteht und man bei einer Verteilung auf mehrere Server eine verteilte Datenbasis benötigt und wir uns somit im Kreis drehen würden.

OceanStore verwendet einen Dienst, der einen Aufsatz auf TCP/IP darstellt und das Routing übernimmt. Dieser Dienst heißt Tapestry und ist eine Peer-To-Peer (P2P) Routingarchitektur, die die gewünschten Anforderungen erfüllt. Der Vorteil ist, dass Tapestry denselben Namensraum verwendet wie der Algorithmus zur Namensvergabe. Damit kann man die Objekte auf den, dem Namen entsprechenden, Serverknoten ablegen und kann lokal entscheiden.

Knotennamen werden unter Tapestry als eine Zeichenkette aus Zahlen verstanden. Wenn man nun jede Zahl dieser Zeichenkette als Ebene auffasst, kann man nach [Kub00a] folgende Aussagen treffen:

- Es gibt eine Zeichenkette α bestehend aus Zahlen zur Basis b, die den Knoten A identifiziert
- Es gibt Präfixe β von α derart, dass $\beta \circ \delta \equiv \alpha$. Wobei $|\beta|$ die Länge von β bezeichne und $\beta \in b^{|\beta|}$, $\delta \in b^{(|\alpha|-|\beta|-1)}$
- Für jeden Präfix $\beta \circ j$ mit $j \in [0, b-1]$ gibt es eine Menge $\mathcal{N}^A_{\beta,j}$ von Nachbarknoten zu A deren Bezeichner den Präfix $\beta \circ j$ mit A teilen
- jeder Knoten A besitzt b Tabellen der Form $\mathcal{N}^A_{\beta,j}$
- $\mathcal{N}^A_{\beta,j}$ ist nach oben durch $R > 1$ beschränkt (d.h. $|\mathcal{N}^A_{\beta,j}| < R$)

Wenn $l = |\beta| + 1$ die Ebene des Routings angibt, dann besteht der Algorithmus darin, dass sukzessiv jede Stelle $k = 0 \ldots |\alpha|$ von Knoten zu Knoten über die jeweilige Menge $\mathcal{N}^A_{\beta,j}$ der Ebene l aufgelöst wird. Durch die Anwendung dieses Algorithmus kann unabhängig von den Knoten der Primärknoten eines Datenobjektes sowohl bei der Speicherung als auch beim Suchen bestimmt werden.

Wie man leicht erkennen kann, besteht mit diesem Algorithmus zwar die Möglichkeit der deterministischen Wegewahl, jedoch werden die Daten wahrscheinlich in den seltensten Fällen in der Nähe zu liegen kommen. Außerdem speichert nur ein Rechner die Daten, was zu Lasten der Ausfallsicherheit geht, da ein „Single Point of Failure“, also eine singuläre Fehlerquelle, besteht.

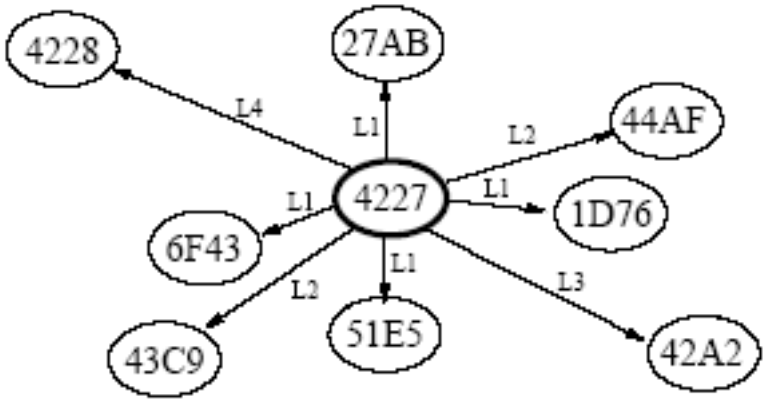


Abbildung 5.1: Ebenenverweise eines Knotens

Um diese Nachteile zu beseitigen wird zugelassen, dass sich zu einem Zeitpunkt mehrere identische Datenobjekte im System auf verschiedenen Knoten befinden. Dazu wird das primäre Datenobjekt repliziert und auf beliebigen Servern abgelegt. Diese Vorgehensweise zieht, obwohl sie das Problem der Lokalität löst, neue Probleme mit sich:

1. Woher ist der Speicherort des Replikats bekannt
2. Wie viele Replikate sind optimal
3. Was passiert mit den Replikaten, wenn die Daten veraltet sind

Auf den letzten Punkt wird in einem der folgenden Kapitel eingegangen, weshalb hier nur die ersten zwei Punkte näher erläutert werden. Die Frage nach der Bekanntheit des neuen Speicherorts ergibt sich aus der Voraussetzung, dass Repliken des Primärobjektes auf beliebigen anderen Knoten zu liegen kommen können.

Wenn ein Replikat auf einem Server erstellt wird, so müssen die aktuellen Daten vom Primärknoten bezogen werden. Dazu wird nach dem oben vorgestellten Verfahren der Primärknoten aufgesucht und auf jedem Zwischenknoten ein Verweis auf den Knoten gesetzt, der das Replikat anfordert. Somit ist sichergestellt, dass das Replikat wieder gefunden werden kann. Um nun auch wirklich ein Replikat, das sich in der Nähe befindet zu finden, wird der Routingalgorithmus um eine vorgelagerte Abfrage erweitert, die als erstes prüft, ob es einen Knoten gibt, der sich in der Nähe befindet und ein Replikat besitzt. Wenn dem so ist, wird dieses anstatt dem Primärobjekt aufgesucht. Andernfalls wird das ursprüngliche Verfahren fortgesetzt.

Die Frage nach der optimalen Anzahl der Replikate kann niemand besser beantworten als das System selbst. Am praktikabelsten wäre es, wenn Replikate nach Bedarf erzeugt und wieder verworfen werden. Wenn man nun

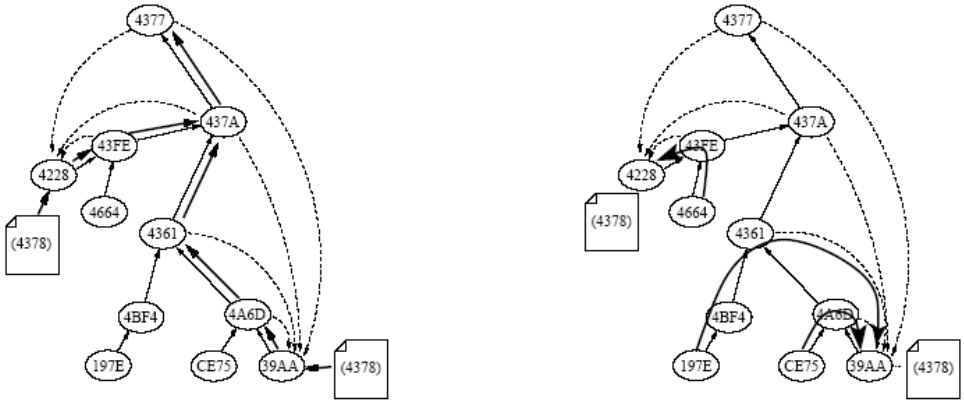


Abbildung 5.2: Einfügen und Abfragen von Replikaten

von einem Server ausgeht, der diverse Anfragen nach einem bestimmten Objekt bekommt, das er selbst nicht speichert, so wird er die Anfragen an den entsprechenden Server weiterleiten müssen. Kommen mit der Zeit mehr Anfragen, die der Server nicht mehr in akzeptabler Zeit beantworten kann, mag es besser sein das Objekt lokal vorrätig zu haben und die Anfragen direkt mit diesem Objekt zu beantworten. Das heißt, dass man über Quality of Service Parameter eine dem Knoten entsprechende Metrik bestimmen kann, mit der Objekte sowohl erzeugt als auch wieder gelöscht werden sollen.

5.2.4 Daten aktualisieren

In diesem Abschnitt wird die letzte, noch unbeantwortete Frage aus dem vorigen Abschnitt wieder aufgegriffen. Rückwirkend betrachtet besteht das Problem der Datenaktualisierung nun nicht mehr nur, wie anfangs beschrieben darin, dass eine Änderung der Daten eine Änderung der Objektidentität bewirkt. Wie gerade gesehen müssen auch die Replikate von einer Aktualisierung der Daten in Kenntnis gesetzt werden, um Anfragen nicht mit veralteten Daten zu beantworten.

Es soll an dieser Stelle erneut auf die Wichtigkeit des Primärknotens hingewiesen werden. Wenn man das Konzept des Primärknoten zu einem Primärverband erweitert, erhält man somit eine weitere Verbesserung der Ausfallsicherheit. Da nun nicht mehr lediglich ein Knoten für die Speicherung der Daten verantwortlich ist, kann ein Objekt nicht mehr aufgrund eines einzelnen Servers ausfallen.

Durch diese Maßnahme entsteht jedoch zusätzlicher Aufwand an Kommunikation, da die Datenänderung bei konkurrierendem Zugriff serialisiert werden muss. Für dieses Unterfangen müssen sich alle Knoten des Primärverbunds auf eine endgültige Reihenfolge der Aktualisierung einigen. Hierzu

kann man einen aufwändigen und langsamen Algorithmus billigend in Kauf nehmen, da es sich bei diesem Primärverbund um verhältnismäßig wenige Knoten handelt.

In der OceanStore-Architektur wird ein „byzantine agreement protocol“ verwendet, mit dem es möglich ist, dass sogar unsichere Server in einem Primärverbund teilhaben können. Bei diesem Protokoll werden m Eingabedaten in n mit $n > m$ Dateneinheiten transformiert, von denen aus m Dateneinheiten wieder das Originaldatum hergestellt werden kann.

Damit wäre geklärt, wie man die Korrektheit einer Aktualisierung sicherstellt. Jedoch besteht weiterhin das Problem die Replikate auf den aktuellen Stand zu bekommen. Aus diesem Grund werden Aktualisierungsanfragen nicht ausschließlich an den Primärverbund gesendet sondern zusätzlich noch an beliebige weitere Replikate. Diese Replikate verbreiten den Verfall untereinander, während der Primärverbund über die richtige Aktualisierungsstrategie befindet. Danach werden die Objekte an die entsprechenden Knoten anhand der Verweiskette verteilt.

Bis jetzt wurde vieles über Aktualisierungsstrategien gesagt, jedoch wurde noch nicht erwähnt wie diese Daten aussehen. Im Falle von OceanStrore sind die Daten in Blöcke unterteilt, und da OceanStore von unsicheren Knoten ausgeht haben wir gesehen, dass die Daten auf dem Dienstnehmer verschlüsselt werden. Wenn die Daten aber verschlüsselt sind, hat dies Konsequenzen für die Aktualisierung, da die Knoten keinerlei Prüfungen auf den Daten vornehmen können. Folglich müssen die Aktualisierungen in verschlüsselter Form stattfinden können. [Kub00b] machen acht Operationen aus, die für OceanStore relevant für Aktualisierungen sind:

compare-version	Vergleichen von Versionen zweier verschlüsselter Blöcke
compare-size	Größen von Blöcken vergleichen
compare-block	Vergleich auf Gleichheit verschlüsselter Blöcke
search	Suchen nach einer verschlüsselt übergebenen Zeichenkette in einem Block

Damit diese Operationen zur Verfügung gestellt werden können, werden in [Kub00b] verschiedene Dinge vorausgesetzt, unter anderem, dass eine Verschlüsselung gewählt wird, die „positionsabhängige Chiffren“ bereitstellt. Damit kann das System zu den Datenobjekten Metadaten speichern, die ein *compare-block* überhaupt erst zulassen.

Wenn wir der Reihe nach durchgehen, werden wir feststellen, dass ein *compare-version* und ein *compare-size* eine einfache Operation auf den unverschlüsselten Metadaten darstellt. Die etwas kompliziertere Operation *compare-block* wird durch die angeforderten Voraussetzungen ebenfalls verhältnismäßig einfach, indem der Dienstnehmer nun einfach einen verschlüsselten

Block sendet, den er verglichen haben will. Das Thema *search* haben unter anderen [Bri04] behandelt und Möglichkeiten vorgestellt, verschlüsselte Daten zu durchsuchen.

[Kub00b] stellen noch vier weitere Operationen vor

replace-block	Ersetzen eines verschlüsselten Blocks durch einen neuen
insert-block	Einfügen eines zusätzlichen verschlüsselten Blocks
delete-block	Löschen eines beliebigen verschlüsselten Blocks
append	Anhängen eines verschlüsselten Blocks

Sowohl *replace-block* als auch *append* seien genau wie *compare-block* einfach mit Hilfe der unverschlüsselten Metadaten anzubieten. Für *insert-block* und *delete-block* stellen sie Verfahren vor, die die Daten in zwei Teile, einen Datenteil und einen Indexteil, aufspalten. Für das Einfügen oder das Löschen müsse man somit lediglich Referenzen der Indexblöcke auf die Datenblöcke anpassen. Sie betonen jedoch, dass der Bereich des Änderns von Daten zur Zeit noch Gegenstand vieler Untersuchungen wäre. Meiner Meinung nach kann diese Vorgehensweise so auch nicht in das System einfließen, da es sonst eine Möglichkeit gäbe Daten in dem System zu ändern, ohne die Objektidentität anpassen zu müssen. Diese Tatsache widerspricht aber der eingangs erwähnten Strategie der Generierung von Objektidentifikatoren und der deterministischen Routingstruktur.

5.2.5 Sicherheit

Das letzte Thema, das in dieser Arbeit angesprochen werden soll, widmet sich der Authentizität von Daten und der Autorisierung des Zugriffs. Diesen zwei zentralen Punkten beim Entwurf einer Mehrbenutzerdatenbank sollte eine besondere Aufmerksamkeit entgegen gebracht werden. OceanStore bewältigt diese Aufgabe mit sehr einfachen aber effektiven Mitteln und soll deshalb als Fallbeispiel für unsere Überlegungen Pate stehen.

Wenn man nochmals einen Blick auf die Voraussetzungen von OceanStore wirft, wird immer wieder betont, dass man von einer unsicheren Umgebung ausgeht. Das heißt, dass sowohl die Dienstnehmer als auch teilnehmende Knoten potenzielle Sicherheitslücken darstellen. Es wurde bereits mehrfach erwähnt, dass die Dienstnehmer die Daten sozusagen „Offline“ verschlüsseln und diese dann ins System einstellen. Damit hat man bereits erreicht, dass es irrelevant ist, ob ein Knoten, der die Daten speichert sicher ist oder nicht. Bei geeigneter Wahl der Verschlüsselungsmethode und der Schlüssellänge ist es praktisch unmöglich die Daten zu entschlüsseln.

Genau betrachtet ist dies auch der Punkt, an dem die Authentizität ansetzt.

Wenn wir uns überlegen, wie man Authentizität zum Beispiel bei Mailsystemen sicherstellt, kommt man schnell auf Signaturen. Es liegt also nahe, so etwas auch in ein Datenbanksystem zu integrieren. Wenn nun der Nutzer nach dem Verschlüsseln der Daten diese auch noch mit einem Schlüssel signiert und diese Signatur als Metadaten an das Objekt anhängt, kann man zur Authentifizierung bereits bestehende Sicherheitstopologien wie VeriSign oder andere verwenden. Es ergibt sich dadurch nicht einmal ein Aufwand für die Entwicklung einer speziellen Sicherheitsarchitektur.

Nachdem wir nun sicher sein können, von wem die Daten sind, sollten wir überlegen, wie wir Freunden und Kollegen ermöglichen, unsere Daten anzusehen und zu verändern. Wenn man unverschlüsselte Daten verwenden würde, wäre es einfach den Zugriff über „Access Control Lists“ zu regeln. Man würde einfach in dieser Liste die Benutzerkennungen der zugelassenen Benutzer speichern und das System würde sich dann darum kümmern. Jedoch wo würde man diese Listen speichern? Unter Umständen würde diese Liste auf einem unsicheren Knoten liegen, der damit diesen Ansatz zerstören könnte.

Da die Benutzer jedoch aus den schon bekannten Gründen die Daten verschlüsseln, wäre die einzige Möglichkeit die Daten so zu verschlüsseln dass nur der Besitzer und die autorisierten Benutzer die Daten lesen können. Wenn wir also ein „Public Key“ Verfahren benutzen, um die Daten zu chiffrieren besteht das Problem, dass ich die Daten mit dem öffentlichen Schlüssels des Empfängers, also des anderen zugelassenen Benutzers chiffrieren müsste. Das ist jedoch unmöglich, da ich nicht für jeden autorisierten Benutzer ein Datenobjekt speichern will und kann. Deshalb müsste man die Daten mit einem öffentlichen Schlüssel eines universellen autorisierten Benutzers verschlüsseln und diesen Schlüssel dann auf sicheren Wegen an die entsprechenden Benutzer verteilen.

Die OceanStore-Entwickler schweigen sich in ihren Veröffentlichungen zu diesem Thema aus. Da OceanStore aber derzeit auch noch in der Entwicklung steckt mag es auch sein, dass man sich dieser Problematik bis dato nur theoretisch zugewendet hat. Gehen wir also davon aus, es gibt einen Weg einen universellen Schlüssel sicher zu verteilen. Damit wäre das nächste Problem, dem man sich zuwenden muss, wie man diese Rechte einem Benutzer wieder entziehen kann. Das einfachste wäre es, einen neuen universellen Schlüssel zu generieren, und diesen dann an die noch autorisierten Benutzer zu verteilen. Dadurch könnte man zumindest eine neue Version des Datenobjektes sichern. Der alte Datenstand eines Objektes kann jedoch nicht dagegen abgesichert werden, dass der Benutzer, dem das Recht diese Daten zu lesen entzogen wurde den ursprünglichen Stand lesen kann. Dieses ist aber ein generelles Problem, das besteht wenn man das „Caching“ von Daten zulässt. Man kann den Cache zwar auffordern seine Daten zu löschen,

ob er der Aufforderung aber tatsächlich nachkommt kann nicht garantiert werden.

5.3 Fazit

Alles in allem ist es fraglich, ob eine Datenbank wie OceanStore, deren Entwickler ursprünglich von einer Datenbank im Einsatz mit ubiquitären Systemen ausgingen in OVID sinnvoll ist. Sicherlich sind wie in der Einführung erwähnt, einige Anwendungsgebiete denkbar in denen eine Datenbank verwendet werden kann, die primär zur Speicherung von Daten an einem Ort und dem Abruf dieser Daten von einem anderen konzipiert ist. Viele Anwendungen sind jedoch darauf angewiesen einen Datenbestand auch durchsuchen zu können oder Daten dem Benutzer anzubieten, von denen er vorher nicht wusste. Diese Anforderungen kann OceanStore nicht erfüllen.

Nichts desto trotz sollte diese Arbeit auch ein wenig Aufschluss darüber geben, welche Punkte bei dem Entwurf einer verteilten Datenbank beachtet werden müssen. Die Entwicklung einer weltweit aufgesetzten Datenbank ist mit Sicherheit sinnvoll und es gibt auch etliche Bereiche, in denen man eine solche Datenbank benötigt.

Für OVID müsste für den Teil der Verarbeitung von Verkehrsdaten eine Datenbank entwickelt werden, die sich vorwiegend auf den Aspekt der Datenaktualisierung konzentriert, und dafür eventuell den Aspekt der weltweiten Verteilung etwas aufgibt, da die Verkehrsdaten doch eher lokal anfallen und meist auch benötigt werden.

Literaturverzeichnis

[Bri04] R. Brinkman. Efficient tree search in encrypted data. Technical report, Univeristy of Twente, Enschede, the Netherlands, http://www.ub.utwente.nl/webdocs/ctit/1/000000f3.pdf, 2004.

[Kub00a] J. Kubiatowicz. Distributed object location in a dynamic network. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/papers/pdf/tocs03.pdf, 2000.

[Kub00b] J. Kubiatowicz. Oceanstore: An architecture for global-scale persistent storage. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/talks/ASPLOS-OceanStore.pdf, 2000.

[Kub01] J. Kubiatowicz. Maintenance-free global data storage. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/papers/pdf/ieeeic.pdf, 2001.

[Kub03] J. Kubiatowicz. Oceanstore status and directions. Technical report, Univeristy of California, Berkley, http://oceanstore.cs.berkeley.edu/publications/talks/tahoe-2003-01/kubitron(underscore)overview.pdf, 2003.

Teil IV

Integration von Datenquellen

Andreas Walter

6

Dienstorientierte Integration

6.1 Einleitung

Heutzutage haben Unternehmen verschiedene Datenbestände in unterschiedlichen Umgebungen. Beispielsweise hat ein Transportunternehmen Datenbestände zu Benzinverbrauch, Kundendaten, Fahrpläne, Straßenkarten usw. Die Dateneingabe geschieht dabei an unterschiedlichen Orten, unter anderem als GPS-Daten von Meßstellen und Fahrzeugen oder durch Protokolle von Fahrkartenautomaten. Von dem eingesetzten System zur Verwaltung dieser Daten fordert man dann unter anderem, dass Daten gefiltert oder aggregiert werden können oder auch eine Abfrage über mehrere Datenbestände möglich ist.

Es stellen sich an das Gesamtsystem eine Reihe von Herausforderungen: Man hat es mit unterschiedlichen Betriebssystemen, verschiedenen Datenbanksystemen und möglicherweise auch mit mehreren Arten von Datenmodellen zu tun. Der dabei vorhandene Datenbestand wird immer umfangreicher. Durch stark wachsende Nutzung über das Internet werden immer mehr Anfragen an das Gesamtsystem gestellt.

Eine Lösung für diese Herausforderungen stellt die dienstorientierte Integration von Datenquellen dar. Dabei werden sämltiche Datenquellen als Dienstgeber betrachtet. Zuerst werde ich die Anforderungen an eine dienstorientierte Integration von Datenquellen (Abschnitt 6.2) allgemein herausarbeiten und die Umsetzung dieser Anforderungen anhand zweier Plattformen, dem Datagrid (Abschnitt 6.3) und der NEXUS-Plattform (Abschnitt 6.4) vorstellen.

6.2 Anforderungen an dienstorientierte Integration

6.2.1 Dienstorientierte Betrachtung von Ressourcen

Bei der dienstorientierten Betrachtung von Ressourcen werden sämtliche Systemkomponenten, also Ressourcen, Speichersysteme, Programme, Netzwerke und auch Datenbanken, als Dienste betrachtet (Abbildung 6.1). Jeder Dienst kennzeichnet sich dann durch einen eindeutigen Namen in der Systemumgebung. Über eine Dienstschnittstelle stellt er seine zur Verfügung stehenden Dienstmerkmale Anwendern zur Verfügung, mittels Dienstbeschreibungen beschreibt er seine Dienstmerkmale.

Ein Datenbankdienst bietet als Dienstmerkmale zum Beispiel Operationen zum Lesen von Inhalten aus Datenbanken an und stellt diese Dienstnehmern zur Nutzung bereit. Falls diese mit der Schnittstelle des Datenbankdienstes umgehen können, rufen sie die Dienstmerkmale darüber auf.

Abbildung 6.1: Betrachtung von Systemkomponenten als Dienste

6.2.2 Vorteile der dienstorientierten Betrachtung von Ressourcen

Durch die Betrachtung von Ressourcen als Dienste ergeben sich eine Reihe von interessanten Vorteilen gegenüber der spezifizierten Nutzung der Ressourcen.

Definiert man eine gemeinsame Schnittstelle, die alle Dienste kennen sollen, wird dadurch die Wirkung aller Dienste nach außen hin einheitlich. Lokal wird der Dienst an das zu Grunde liegende Betriebssystem angepaßt, beispielsweise durch Installation eines für das Systen verfügbaren Webservers, falls der Dienst auf Basis von WebServices laufen soll. Ein Datenbankdienst wird zusätzlich an das verwendete Datenbanksystem angepaßt, wie zum Beispiel durch Definition von passenden Anfragen zu den Dienstmerkmalen.

Die Entkapselung hinter einer gemeinsamen Schnittstelle führt zu Virtuali-

sierung der Dienste. Die einzelnen Dienste werden über einen Dienstnamen aufgerufen, wodurch der tatsächliche Ort des Dienstes in der Umgebung transparent wird. Weiterhin erreicht man damit den konsistenten Zugriff auf Ressourcen über verschiedenen, heterogene Plattformen. Wie die Schnittstelle des Dienstes implementiert werden soll, bleibt den jeweiligen Betreibern des Dienstes überlassen. Wichtig ist, dass die Funktions- und Verhaltensweisen der gemeinsam definierten Schnittstelle umgesetzt und gewährleistet werden können und einheitlich beschrieben werden, nicht aber, wie die tatsächliche lokale Umsetzung erfolgen soll.

Eine sehr wichtige, weitere vorteilhafte Eigenschaft bei der Betrachtung von Ressourcen als Dienste ist, dass der Aufbau von höherwertigen Diensten durch Komposition von vorhandenen Diensten möglich ist. Beispielsweise ergibt die Komposition von Datenbanken einen Datenbankverbund. Gibt es, wie im OVID-Projekt, Datenbank-Dienste mit Bezug jeweils zu einem deutschen Bundesland, so ergibt die Komposition dieser Dienste zu einem Dienst ´Datenbank-Dienst Deutschland gesamt‘ ein höherwertigeren Dienst, der dann Anfragen zu ganz Deutschland ermöglicht.

6.2.3 Anforderungen

Um eine effektive Nutzung von Diensten innerhalb einer Systemumgebung zu erreichen, müssen die Dienste untereinander kommunizieren können und mit Ergebnisdaten umgehen können.

Hierzu muss zum einen die Definition der Schnittstellen standardisiert werden, denn jeder Dienst muss einheitliche Methodennamen und Parameter haben, damit die Schnittstelle von anderen Diensten einheitlich genutzt werden kann. Ebenso müssen die Dienste ein einheitliches Verständnis für die Ergebnisdaten der genutzten Dienste haben. Eine einheitliche Datenausgabe kann man zum Beispiel durch Verwendung eines XML-Standards erreichen. Auch das Verständnis für Fehlermeldungen muss einheitlich sein, damit bei auftretenden Fehlern entsprechend reagiert werden kann.

Wie bereits erwähnt, erhält jeder Dienst einen Namen, unter dem er von Dienstnehmern aufgerufen werden kann. Zum einen ist es hierbei notwendig, dass die Dienste einen eindeutigen Namen erhalten, um eindeutig adressiert werden zu können. Weiterhin muss es ein Verfahren zum Auffinden von Diensten geben. Im Allgemeinen gibt es hierzu einen speziellen Dienst, der in Form von Gelben Seiten ein Verzeichnis führt, das die Namen aller verfügbaren Dienste und deren Dienstmerkmale auflistet und im System abrufbar macht.

Speziell bei Datenbanksystemen ist es notwendig, dass gewisse Qualitätsmerkmale wie zum Beispiel eine gesicherte Ressourcenzuteilung für eine

gewisse Zeit oder ein Schutz der Ressource durch Authentifizierung und Autorisierung eingehalten werden. Auch ein Datenbankdienst muss diese Qualitätsmerkmale einhalten können. Der Dienst sollte also zum Beispiel Zustände haben, die sich merken, ob ein Dienstnehmer autorisiert ist, den Dienst zu nutzen, und auch Verfahren beinhalten, wie die Ressourcen gesichert für eine gewisse Zeit an einen Dienstnehmer gebunden werden können.

In den nun folgenden Kapiteln ´DataGrid‘ und ´Die NEXUS-Plattform‘ werden zwei Architekturen vorgestellt, welche die Vorteile der dienstorientierten Integration von Datenbanken nutzen, und Verfahren beinhalten, welche die genannten Anforderungen erfüllen.

6.3 Dienstorientierte Integration in einem Data-Grid

Im folgenden wird die dienstorientierten Integration von Datenbanken mittels der Open Grid Services Architecture-Data Access and Integration (OGSA - DAI) Plattform vorgestellt [FKNT02]. Dies ist eine Architektur, welche auf dem Standard von Grids eine Infrastruktur aufbaut, die sowohl fähig ist, Dienste zu integrieren wie auch den Zugriff auf Datenbanken zu ermöglichen.

6.3.1 Virtuelle Organisationen

Eine Gruppe von Organisationen, Firmen oder Individuen, welche an verschiedenen Orten sitzen und gewisse Ressourcen miteinander teilen möchten, wird virtuelle Organisation genannt. Jedes Mitglied der virtuellen Organisation sitzt an einem anderen Ort und verfügt über eine gewisse Menge von Ressourcen, welche sie über ein Netzwerk anderen Mitgliedern der Organisation zugänglich machen wollen.

In Abbildung 6.2 besteht die virtuelle Organisation aus vier einzelnen Organisationen, welche über eine Satellitenverbindung miteinander verbunden sind. Organisation 1 möchte seine Ressourcen C1 und D1 den anderen Mitgliedern zur Verfügung stellen, Organisation 2 möchte die Ressource E2 teilen, Organisation 4 die Ressourcen A3 und D3 und die Organisation 4 möchte die Ressourcen B4 und C4 teilen. Die Menge aller zugänglich gemachten Ressourcen in diesem Netz ergibt die Gesamtmenge an Ressourcen dieser virtuellen Organisation.

Zumeist hat man es mit einer Menge von heterogenen Systemen in der virtuellen Organisation zu tun. Dadurch entsteht die Notwendigkeit nach einer Plattform, welche diese heterogenen Systeme nahtlos integrieren kann.

6.3.2 Open Grid Architectur

Open Grids ermöglichen die gemeinsame, koordinierte Nutzung und das Teilen von Ressourcen innerhalb und zwischen virtuellen Organisation. Die Open Grid Architectur definiert dabei die zu verwendende Technologie und Infrastruktur eines Open Grids und unterstützt die gemeinsame, koordinierte Nutzung von verschiedenen Ressourcen innerhalb einer virtuellen Organisation. Dies geschieht durch die Definition von einheitlichen Verfahren zum Teilen von Ressourcen wie auch durch die Einrichtung von einheitlichen Zugangsverfahren für die einzelnen Nutzer der Ressourcen der virtuellen Organisation.

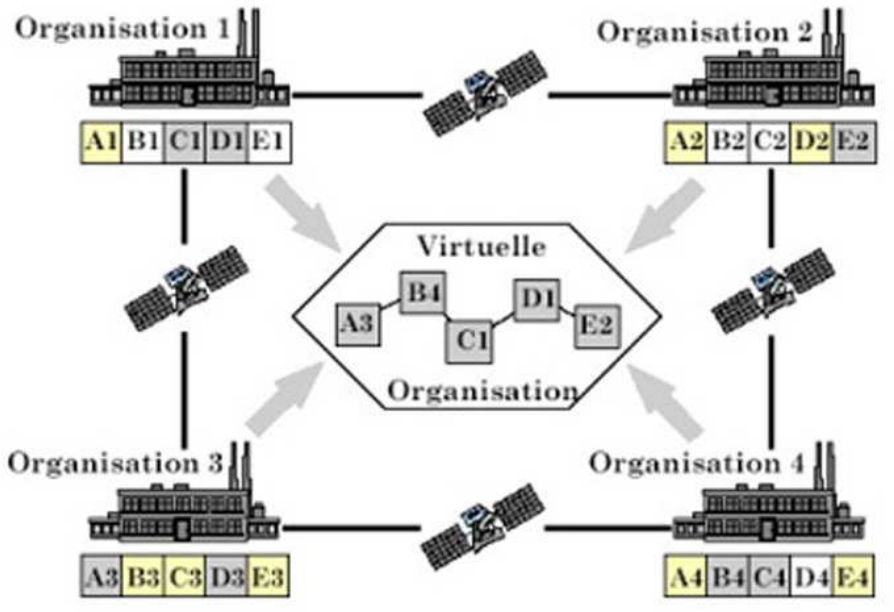


Abbildung 6.2: Virtuelle Organisation

6.3.3 Open Grid Services Architecture

Die Open Grid Services Architecture (OGSA) basiert auf der Open Grid Architektur. Diese Spezifikation wurde so erweitert, dass die einzelnen Komponenten des Grids dienstorientiert integriert werden können. Um dies zu ermöglichen hat man das standardisierte Konzept von Web Services mit der Open Grid Architektur kombiniert. Als Ergebnis werden alle Komponenten der OGSA als so genannte Grid Services integriert.

Web Services

Durch Web Services wird eine Technik definiert, um Ressourcen als Dienste zu beschreiben, die in einem Netzwerk erreicht werden sollen. Dabei gibt es standardisierte Verfahren, um zum einen die vorhandenen Dienste aufzufinden und zum anderen diese nutzen zu können.

Webservices sind neutral zu Programmiersprachen, der verwendeten Systemsoftware und gegenüber den verwendeten Programmiermodellen. Es gibt drei wichtige Standards für Webservices, das Simple Object Access Protocol (SOAP), die Web Service Description Language (WSDL) und sowie UDDI. Alle drei Standards sind XML-basierend.

Im Hinblick auf OGSA bieten Web Services zwei wichtige Funktionen. Zum

einen die Möglichkeit, das Beschreiben und dynamische Auffinden von Diensten in heterogenen, verteilten Netzen zu unterstützen und zum anderen, dass damit eine Architektur zur Verfügung gestellt werden kann, die aufgrund der Nutzung durch Web Services bereits akzeptiert und verbreitet ist, wie beispielsweise durch Entwicklungsumgebungen wie Microsoft .NET.

Simple Object Access Protocol (SOAP)
Das Simple Object Access Protocol ist ein standardisiertes Protokoll für die Kommunikation zwischen Dienstgeber und Dienstnehmer auf Basis von Web Services. Es definiert eine Konvention für für den Nachrichtenaustausch. SOAP ist unabhängig vom verwendeten Transportprotokoll und kann daher auf http, JMS, FTP etc. verwendet werden.

Web Service Description Language (WSDL)
Die Web Service Description Language dient zur Beschreibung der Dienstmerkmale eines WebServices. Es beschreibt die Funktion der Dienstmerkmale und gibt an, welche Eingabeparameter jeweils notwendig sind und welche Ausgabedaten damit erzielt werden.

Universal Description, Discovery, and Integration (UDDI)
UDDI bildet das Verzeichnis zum Auffinden von Diensten und Dienstmerkmalen. Ein entsprechendes XML-Dokument kann Sammlungen von Dienstmerkmalbeschreibungen und Verweise zu anderen Quellen mit Dienstmerkmalbeschreibungen enthalten. Eine Dienstbeschreibung ist im Allgemeinen ein Verweis auf das Web Service Description Dokument des entsprechenden Dienstes

Open Grid Service Infrastructur

Die Open Grid Service Infrastructur (OGSI) stellt die Basisinfrastruktur dar, auf der die Dienste der Open Grid Service Architektur aufbauen. Die Infrastruktur hat als Basis einen Web Service. Definiert werden hierbei die Standardschnittstellen sowie das Verhalten der Grid Services. Der Web Service wird zusätzlich erweitert um die Möglichkeit der asynchronen Ereignisbenachrichtigung, Zustände zur Verwaltung von Referenzen und Lebenszeiten sowie den Mustern Factory, Registry und Collection zur Erstellung und Verwaltung von Instanzen des Grid Services.

Grid Services

Grid Services setzen auf der Open Grid Service Infrastructur auf. Sie sind unabhängig von der zugrunde liegenden Plattform und der verwendeten Implementierung. Jeder Grid Service enthält die Methoden der Standardschnittstelle

Port Typ	Operation	Beschreibung
Grid Service	Find Service Data	Anfrage von Informationen über den Grid Service
	SetTerminationTime	Setzt die Laufzeit für den Service
	Destroy	Zerstört den Service
Notification Source	Subscripe to	Anmelden zu Nachrichten über serviceverwandte Ereignisse
Notification Sink	DeliverNotification	Ermöglicht asynchrones Erreichen von Empfangs-bestätigungen
Registry	RegisterService	Registrierung von Grid Services
	UnregisterService	Abmelden eines Grid Services
Factory	CreateService	Erstellt neue Service Instanz
HandleMap	FindByHandle	Gibt Grid Service Referenz zurück

Tabelle 6.1: Standardschnittstelle eines Grid-Services

Zwei wichtige Eigenschaften dieser Schnittstelle sind dabei Reliable Service Invocation und Lifetime Management.

Reliable service invocation
Innerhalb eines Netzes kann nicht garantiert werden, dass Nachrichten den Empfänger erreichen. Um dieses Problem zu lösen können innerhalb des Grid Service interne Zustände eingerichtet werden, die anzeigen können, ob eine Nachricht einmal, öfters oder nie ankam. Innnerhalb der Standardschnittstelle eines Grid-Services (Tabelle 6.1) sind die Operationen ´Subscripe to‘ und ‘Deliver Notification‘ für diese Zustände verantwortlich.Auf dieser Basis kann eine Reihe von höherwertigen Operationen wie zum Beispiel Transaktionen aufgebaut werden.

Lifetime-Management
Durch das Lifetime-Management kann die Zuteilung von Ressourcen für eine gewisse Zeit garantiert werden. Die Operationen ´SetTerminationTime‘ und ´Destroy‘ der Standartschnittstelle sind hierfür verantwortlich. Es wird eine gewisse Zeit t angegeben, für die der Grid Service erstellt werden soll. Nach Ablauf dieser Zeit wird der Grid Service abgemeldet und die daran gebundene Ressource wieder frei gegeben.

6.3.4 Open Grid Services Architecture - Data Access and Integration

Mit der Open Grid Services Architecture - Data Access and Integration wurde eine Erweiterung der OGSA geschaffen, um Datenbanksysteme in die Architektur integrieren zu können [CFK+01].

Die Erweiterung sorgt dafür, dass Daten aufgefunden und abgerufen werden können, die sich über verschiedene, heterogene Datenbanksysteme verteilen. Dabei können verschiedene Datenmodelle aus verschiedenen Quellen integriert werden unter der Garantie von Qualitätsmerkmalen. Die Schnittstelle der Erweiterung ist ebenfalls standardisiert.

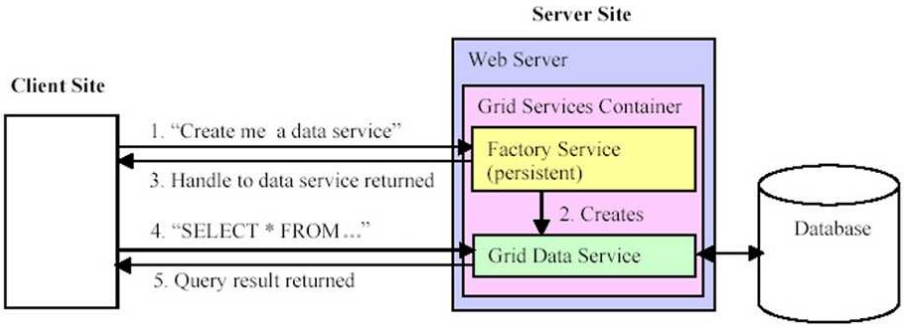


Abbildung 6.3: Grid Data Services

Die Erweiterung zur Einbindung von Datenbanksystemen besteht aus der Grid Data Service Factory und dem Grid Data Service.

Grid Data Service Factory

Bei der Grid Data Service Factory (Abbildung 6.3) handelt es sich um eine Umsetzung des Musters Factory. Die Factory setzt auf einer Datenquelle auf und kann Grid Data Service Instanzen erzeugen, welche dann für die eigentliche Abfrage der Daten zuständig sind. Weiterhin liefert die Factory Informationen über die erreichbare Datenquelle in Form von Metadaten.

Grid Data Service (GDS)

Beim Grid Data Service (GDS) handelt es sich um die zentrale Komponente einer OGSA-DAI. Der Service besteht aus einer Standardschnitstelle, über die der Datenzugriff, die Datenintegration und Datenabfrage einer OGSA ermöglicht wird.

Die Schnittstelle definiert Aktivitäten, welche auf eine Datenquelle und mit daraus abgefragten Daten durchgeführt werden können. Sie besteht aus den Aktivitäten zur Abfrage von relationalen Datenbanksystemen, Abfrage von XML-basierenden Datenbanksystemen, Angaben zur Auslieferung und Transformation der Daten und Aktivitäten zur Abfrage von Daten aus Dateisystemen. Weiterhin können selbstdefinierte Aktivitäten definiert werden.

Aktivitäten zur Abfrage relationaler Datenbanksysteme

Aktivität	Beschreibung
sqlQueryStatement	Führt SELECT Abfrage aus
sqlUpdateStatement	Führt UPDATE Abfrage aus
sqlStoredProcedure	Führt eine Funktion aus

Beispiel einer SQL-Anfrage

Ein Beispiel soll den Ablauf einer SQL-Anfrage in einem OGSA-DAI zeigen. Als Basis dient die Datenquelle eines Busunternehmens namens FIRST. Abgefragt werden soll die Position eines Busses mit Nummer ´FIRST1234‘ im OGSA-DAI des Unternehmens.

Die SQL-Anfrage lautet

```
String SQLTag= "SELECT XPOS,YPOS FROM BUSSES WHERE Name=
'FIRST1234'";
```

Zuerst muss der entsprechende Dienst gefunden werden. Dies geschieht über das Verzeichnis der Web Services UDDI. Wenn der entsprechende Dienst gefunden wurde, wird dieser kontaktiert.

Im zweiten Schritt wird ein Grid Data Service Factory und ein Grid Data Service für den Dienstnehmer erstellt.

An diesen Grid Data Service wird dann im dritten Schritt die SQL-Anfrage übergeben. Dieser führt die Anfrage aus.

```
SQLQuery query = new SQLQuery(SQLTag);
Response response = service.perform( query );
```

Das Ergebnis in response enthält das Ergebnis der Anfrage formatiert als Web Row Set. Ein Web Row Set entspricht einem SQL Row Set wie in JDBC in einer XML-Repräsentation.

Das XML-Dokument besteht aus den Bereichen Properties, das allgemeine Informationen zum Datenbanksystem angibt, den Metadaten, mit denen die

Spalten des Abfrageergebnisses beschrieben werden und dem Bereich Data, in dem die eigentlichen Daten enthalten sind.

Abfrageergebnis als WebRowSet

```
<properties>
    z.B.
    <max-field-size>1048576</max-field-size>
    <max-rows>0</max-rows>
    ...
</properties>
<metadata>
    z.B.
    <column-count>2</column-count>
    <column-definition>
    <column-name>XPOS</column-name>
    <column-index>1</column-index>
    ...
    </column-definition>
    ...weitere Spaltenbeschreibungen
       fuer z.B. YPOSITION folgen...
</metadata>
<data>
    <currentRow>
        <columnValue>3475</columnValue>
        <columnValue>1234</columnValue>
    </currentRow>
</data>
```

Zusammenfassung Data-Grid

Innerhalb eines Data-Grid können also Datenbanksysteme dienstorientiert integriert werden. Die Datenbanksysteme werden dabei durch die Erweiterung 'OGSA-DAI' innerhalb einer bewährten und bestehenden Grid-Service Architektur eingebunden. Alle Anforderungen an eine dienstorientierte Integration von Datenbanken sind erfüllt. Die jeweils vorhandenen Web-Services sorgen für eine einheitliche Schnittstelle und ein Auffinden von gewünschten Diensten, die Erweiterung der Web Services um Zustände können weiterhin Qualitätsmerkmale eingehalten werden können. Das einheitliche Verständnis für die ausgegebenen Daten wird über die XML-basierenden WebRowSets gewährleistet.

Insgesamt erscheint die Architektur etwas umständlich, da doch recht viel bezüglich Schnittstellen, Architektur und Infrastruktur bewerkstelligt wer-

den muss, bis überhaupt einmal der Abruf von Daten aus Datenbanksystemen möglich ist. Dies erklärt sich durch die Entwicklung des Data Grids, bei der es zuerst die Open Grid Architektur gab, die dann erweitert wurde um Belange zur dienstorientierten Betrachtung der Komponenten und schließlich noch einer Erweiterung, damit die erstellte Architektur auch den Umgang mit Datenquellen bewerkstelligen kann.

6.4 Die NEXUS-Plattform

Bei der von der Universität Stuttgart gegründeten NEXUS-Plattform handelt es sich um eine offene, verteilte Umgebung für ortsbezogene Dienste und Anwendungen [NGS+01]. Dabei werden ähnliche Ziel verfolgt, wie es auch im World Wide Web der Fall ist. Die Plattform ist offen für eine Vielzahl von Anbietern von Daten und Anwendungen. Damit diese untereinander kommunizieren können, müssen sie eine einheitliche Sicht auf das System haben. Dies wird in der NEXUS-Plattform über das Augmented World Model gewährleistet.

6.4.1 Beispiel Navigationssystem

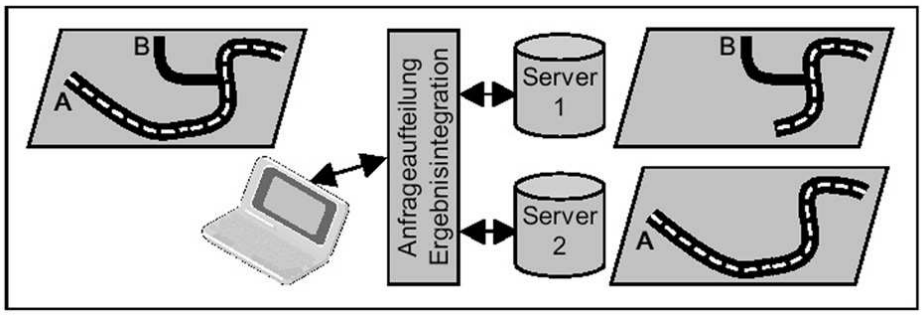


Abbildung 6.4: Beispiel Navigationssystem

Mittels der NEXUS Plattform kann ein Navigationssystem dadurch aufgebaut werden, dass Anbieter Teile der benötigten Streckendaten zur Verfügung stellen (siehe Abbildung 6.4). Für einen Streckenabschnitt könnte es dann zwei verschiedene Provider geben. Ein Provider A speichert den Streckenabschnitt einer Überlandstrasse sowie einer Nebenstraße. Ein Provider B speichert Informationen über den Weiterverlauf der Überlandstraße. Ein Benutzer des Navigationssystems, der eine Anfrage zur Navigation über die komplette Überlandstraße stellen möchte, kontaktiert die NEXUS-Plattform. Beide Provider werden abgefragt. Der Inhalt des Ergebnisses für

die Anwendung sind die Navigationsangaben für die komplette Überlandstraße.

Auch in der NEXUS-Plattform hat man es wieder mit unterschiedliche Datenbestände zu tun, die sich an verschiedenen Orten befinden und aus heterogenen Datenbasen bestehen können. Die Teile der NEXUS-Plattform und wie hierbei die Integration von heterogenen Datenbanksystemen umgesetzt wird, zeigen die folgenden Abschnitte.

6.4.2 Ablauf der Beispielanfrage in NEXUS

1. Der Nutzer gibt den Zielpunkt B in den PDA ein

2. Die aktuellen Position A und die Zielposition B werden an den nächsten verfügbaren NEXUS-Knoten mit der Anfrage nach einem passenden Navigationsdienst übertragen.

3. Der kontaktierte NEXUS-Knoten fragt das Area Service Register an und ermittelt die verfügbare Dienstgeber für die Anfrage. Im genannten Beispiel wären das die Provider A und B.

4. Der NEXUS-Knoten reicht die Anfrage an die ermittelten Dienstgeber in Dienstebene weiter.

5. Die Anfrageergebnisse aus der Dienstebene werden vom NEXUS-Knoten entgegen genommen.

6. Die Teilergebnisse werden ausgewertet und daraus für die Anwendung ein Gesamtergebnis erstellt.

7. Der NEXUS-Knoten reicht Gesamtergebnis an die Anwendung weiter.

8. Die Anwendung wertet das Ergebnisdokument aus und zeigt Ergebnis an.

6.4.3 Architektur der NEXUS - Plattform

Die NEXUS-Plattform besteht aus drei Schichten (siehe Abbildung 6.5). Die oberste Schicht wird gebildet durch die Anwendungen. Darunter liegt die so genannte Förderationsebene, innerhalb der sich NEXUS-Knoten befinden und ein Adressverzeichnis der vorhandenen Dienstgeber. Die Dienstgeber befinden sich dann in Form von Spatial Model Server innerhalb der Dienstebene. Im folgenden werden diese Ebenen im Detail vorgestellt.

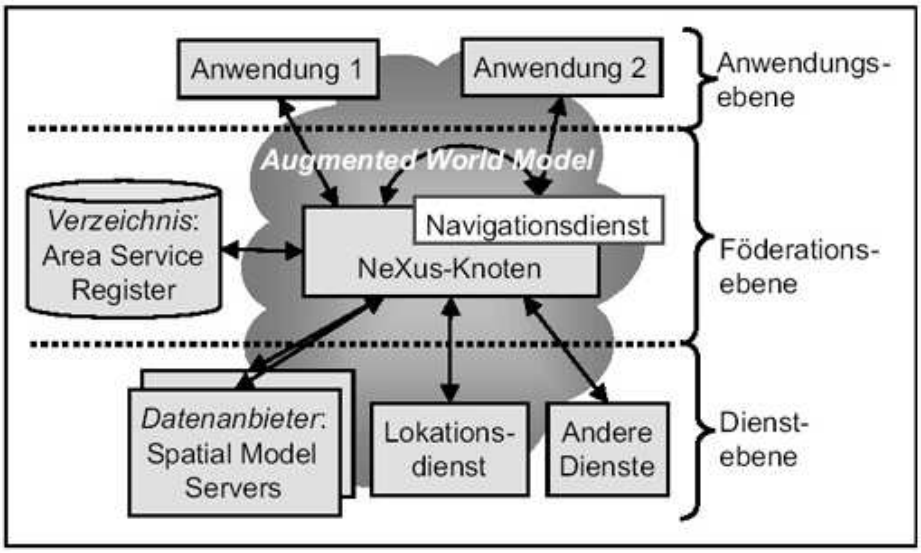


Abbildung 6.5: Architektur der NEXUS - Plattform

Die Anwendungsebene

Innerhalb der Anwendungsebene befinden sich die Anwendungen der NEXUS-Plattform. Zumeist laufen die Anwendungen auf kleinen Geräten wie PDAs oder Mobiltelefonen ab. Zusätzlich enthalten die Geräte zumeist einen GPS-Sensor zur Positionsbestimmung. Zum Stellen von Anfragen kontaktieren die Anwendungen den nächsten verfügbaren NEXUS-Knoten im System. Mögliche Anfragen könnten hierbei ´Bestimmung eines Weges von A nach B‘ oder ´Adresse des nächstes Restaurants sein‘. Als Ergebnis der Anfrage erhält die Anwendung eine XML-Datei, zusätzlich ist auch die Übertragung von Bildinformationen wie zum Beispiel die Grafik einer Landkarte möglich. Aufwändige Berechnungen werden aufgrund der begrenzten Rechenleistung des Systems der Anwendung an die Förderationsebene ausgelagert.

Die Förderationsebene

Die Förderationsebene ermöglicht einen einheitlichen Blick für die Anwendungsschicht auf die NEXUS Dienste. Sie besteht aus NEXUS-Knoten und dem Area Service Register. Unter förderieren versteht man das Zusammenschließen von bestimmten Elementen, im Falle der NEXUS-Plattform sind dies ortsbezogene Daten, die in lokalen Teilbeständen zum Abruf bereit stehen.

NEXUS-Knoten

Die NEXUS-Knoten in der Förderationsebene nehmen die Anfragen aus der Anwendungsebene entgegen, ermitteln die zur Ausführung notwendigen Dienste in der Dienstebene und reichen an diese die Anfrage weiter. Aus den Teilergebnissen der Dienstgeber erstellen sie dann ein für die jeweilige Anwendung lesbares Gesamtergebnis.

Area Service Register
Das Area Service Register ist das Adressverzeichnis der vorhandenen Dienstgeber in der Dienstebene. Es enthält die Adressen der Dienstanbieter, Informationen über die Art von gespeicherten Datentypen wie auch Informationen zum davon abgedeckten Gebiet.

Die Dienstebene

In der Dienstebene befinden sich die tatsächlichen Dienste der NEXUS-Plattform. Von verschiedenen Anbietern werden Spatial Model Server in die Dienstebene integriert. Jeder Spatial Model Server meldet sich dann beim Area Service Register der Förderationsebene an und veröffentlicht seine vorhandenen Datentypen. Jeder dieser Server bietet der Förderationsebene eine einheitliche Schnittstelle an, die Daten werden im einheitlichen Format des Augmented World Models übertragen.

Weiterhin gibt es in der Dienstebene einen Lokationsdienst. Er dient für mobile Objekte wie Personen oder Fahrzeuge mit häufiger Positionsänderung und berechnet deren Position zu einer Zeit t.

Augmented World Model

Über das Augmented World Model haben alle Komponenten der NEXUS-Plattform eine gemeinsame Weltsicht [VGH+02],[VB02]. Das Modell wurde für ortsbezogene Daten und zur Förderation von heterogenen Datenquellen entwickelt und besteht aus einer objektorientierten Datenmodellierung. Es enthält die wichtigsten Objektklassen für ortsbezogene Anwendungen. Daraus werden dann Objektinstanzen des Schemas gebildet, die Augmented World Objects.

Objektorientierte Datenmodellierung

Die objektorientierte Datenmodellierung des Augmented World Models besteht aus dem Standard Class Schema und dem Extended Class Schema. Das Standard Class Schema, festgelegt von der Universität Stuttgart als Betreiber der NEXUS-Plattform besteht aus einem Satz von Objektklassen und seinen Attributen. Alle Nexus-Dienste müssen mit diesem Schema

übereinstimmen. Dadurch kann eine Anwendung einen beliebigen NEXUS-Knoten der Förderationsebene kontaktieren und nach vordefinierten Klassen und Attributen fragen. Das Extended Class Schema erweitert durch Vererbung das Standard Class Schema um detaillierterer Objekte. Somit können Provider von Daten die bestehenden Schema um eigene, für die Anwendung benötigte Objekte erweitern.

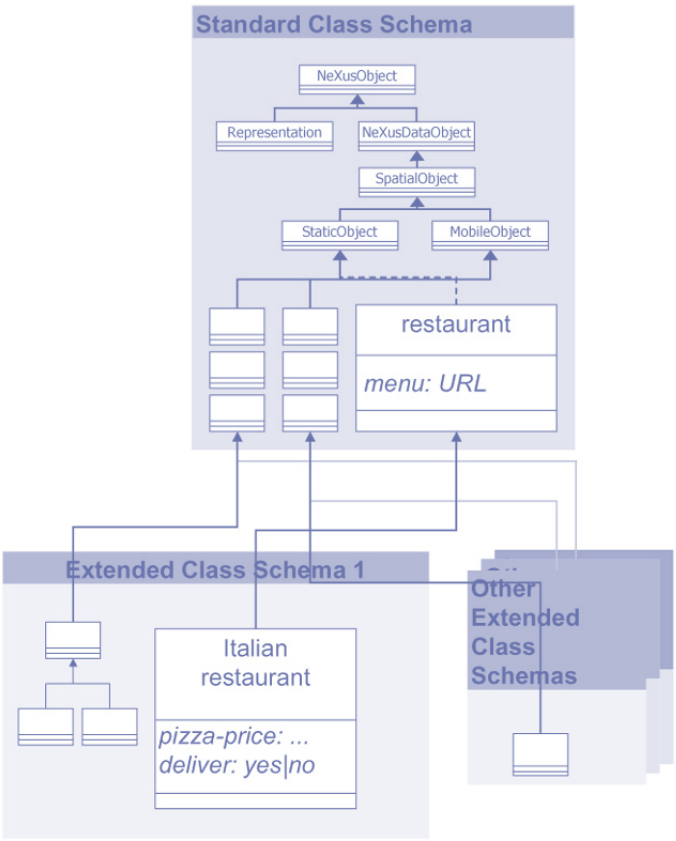


Abbildung 6.6: Beispiel Klassenschema

Gibt es beispielsweise ein Standard Class Schema ´Restaurant‘ (siehe Abbildung 6.6, [B103]) bestehend aus dem Attribut ’URL der Speisekarte’, so könnte es ein erweitertes Schema ’Pizza Restaurant’ geben, dass von der Klasse ´Restaurant‘ erbt und weiterhin detailliertere Angaben wie ’Pizza-Preis’ und ’Lieferservice’ enthält.

Die Ausgabe und Abfrage von Daten geschieht über die XML-basierte Augmented World Modelling Language zur Ausgabe der Daten und der Augmented World Query Language zur Abfrage von Daten. Alle Komponenten

der NEXUS-Plattform stellen Anfragen und erhalten Ergebnisse auf Basis dieser XML-Sprache. Somit wird ein einheitliches Verständnis für die Daten innerhalb der NEXUS-Plattform gewährleistet.

6.5 Zusammenfassung

Sowohl die Open Grid Services Architecture - Data Access and Integration wie auch die NEXUS-Plattform ermöglichen die dienstorientierte Integration von heterogener Datenquellen mit unterschiedlichen Datenmodellen. Im Folgenden werden die Umsetzungen der in Kapitel 2 genannten Anforderungen an eine dienstorientierte Integration von Ressourcen noch einmal zusammengefasst und die jeweiligen Umsetzungen in den beiden Plattformen genannt.

Verzeichnis zum Auffinden der Daten
In OGSA-DAI befindet sich das Adressverzeichnis im UDDI der Web Services. In der NEXUS-Plattform findet man das Adressverzeichnis das Area Service Register in der Förderationsebene.

Einheitliches Datenmodell
Das einheitliche Datenmodell wird in beiden Fällen durch XML-Repräsentationen gewährleistet. Im OGSA-DAI ist dies durch Web Row Sets der Fall, innerhalb der NEXUS-Plattform ist es das Augmented World Model.

Einheitliche Schnittstelle
Die einheitliche Schnittstelle in OGSA-DAI wird durch den Grid Data Service definiert. Die einzelnen Instanzen von Grid Data Services laufen innerhalb eines Web - Servers. Das Standard Class Schema bildet in der NEXUS-Plattform die einheitliche Schnittstelle.

Einheitliches Verfahren zur Garantie von Qualitätsmerkmalen
Über das Lifetime-Management von Data Grid Services kann beispielsweise die Zuteilung von Ressourcen garantiert werden. Eine einheitliche Authentifizierung der Anwendungen in der NEXUS-Plattform gegenüber der Förderationsebene vermeidet den Zugriff von unberechtigten Anwendern.

Literaturverzeichnis

[B103] Forschungsgruppe Nexus B1. B1 – Modellierung und Verwaltung des Umgebungsmodells. http://www.nexus.uni-stuttgart.de/de/forschung/dokumente/docs-teilprojekte/poster-teilprojekt-b1.pdf, Dezember 2003.

[CFK+01] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[FKNT02] Ian Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. http://www.globus.org/research/papers/ogsa.pdf, June 2002.

[NGS+01] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A model-based, open architecture for mobile, spatially aware applications. In C.S. Jensen, M. Schneider, B. Seeger, and V.J. Tsotras, editors, *Proc. Of the 7th Intl. Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*, page 117, Redondo Beach, CA USA, July 2001.

[VB02] Steffen Volz and Jan-Martin Bofinger. Integration of spatial data within a generic platform for location-based applications. In *Proc. Of the Joint Intl. Symposium on Geospatial Theory, Processing and Apllications*, Ottawa, Canada, January 2002.

[VGH+02] Steffen Volz, Matthias Großmann, Nicola Hönle, Daniela Nicklas, and Thomas Schwarz. Integration mehrfach repräsentierter Straßenverkehrsdaten für eine föderierte Navigation. *it+ti*, 44(5):260–267, Oktober 2002.

Prudence KOUAM

7

Integration sich widersprechender Datenquellen

7.1 Einleitung

Verkehrsinformationen stammen aus unterschiedlichen Quellen, sie sind mit einer gewissen Unsicherheit behaftet und können sich widersprechen. Alle diese Informationen müssen trotzdem gespeichert und verarbeiten werden, deswegen muss man den Grad des Vertrauens abbilden. Ich werde in dieser Arbeit zeigen, wie man sich widersprechende Datenquellen integrieren kann.

Beispielsweise jemand ruft eine Verkehrsleitzentrale an und sagt, dass es einen Unfall auf der A5 gegeben habe. Einige Minuten später ruft eine andere Person an und behauptet, dass auf der A5 alles in Ordnung sei, das heisst, dass kein Unfall auf der A5 passiert sei. Wie lassen sich beide Aussagen in einer Datenbank abbilden? Eine Möglichkeit, solche Informationen zu modellieren, ist die Anwendung sogenannte CISET-Ansatz.

7.2 CISET

7.2.1 Konfidenzindex (Confidence Indexe, CI)

Der CISET-Ansatz dient dazu, sich widersprechende Informationen zu integrieren. CISET bedeutet Konfidenzindexmenge *(Confidence Index Set)*. Deswegen beginnen wir unsere Diskussion, indem wir zuerst den Konfidenzindex *(Confidence Index)* vorstellen.

Definition: Sei L ein verteilter Verband und seien α, β Elemente von L. L ist das Einheitsintervall [0,1] und das Paar $a = \langle \alpha, \beta \rangle$ wird Konfidenzindex

genannt. α stellt die negative Aussage (contra) und β die positive Aussage (pro) dar.

Beispiele für Konfidenzindizes sind:
$\langle 0,0\rangle, \langle 1,0\rangle, \langle 1,1\rangle, \langle 0.2,0\rangle, \langle 1,0.4\rangle, \langle 0.5,0.7\rangle$.
Keine Beispiele für Konfidenzindizes sind dagegen:
$\langle 10,0\rangle, \langle -1,0\rangle, \langle 0,-1.2\rangle, \langle 1.5,0.5\rangle$,
da mindestens je ein Elemente nicht im Intervall [0, 1] liegt.
Beispiel: Die Information "Es ist ein Unfall auf der A5 passiert"hat ein Konfidenzintervall $\langle 0.3,0.7\rangle$, d.h. der negativen Aussage hat man den Wert 0.3 zugewiesen und der positiven Aussage den Wert 0.7.

Definition:(Infimum und Supremum)
Sei $a = \langle \alpha, \beta\rangle$ ein Konfidenzindex. Das Infimum (lower index) von a sei $l(a) = \alpha$ und das Supremum (upper index) von a sei $u(a) = \beta$.

Definition:(Gleicheit): Zwei Konfidenzindizes $a_1 = \langle \alpha_1, \beta_1\rangle$ und $a_2 = \langle \alpha_2, \beta_2\rangle$ sind gleich, genau dann wenn: $l(a_1) = l(a_2)$ und $u(a_1) = u(a_2)$.
Definition:(Partielle Ordnung) Jetzt definieren wir die partielle Ordnung $<, >, \leq$ *und* $\geq$.
Es gilt $a_1 < a_2$, wenn einer der folgenden Fälle eintritt: $l(a_1) \geq l(a_2)$ und $u(a_1) < u(a_2)$ oder $l(a_1) > l(a_2)$ und $u(a_1) \leq u(a_2)$. Es gilt $a_1 \leq a_2$ genau dann, wenn $a_1 < a_2$ oder $a_1 = a_2$. Außerdem gilt $a_1 > a_2$ genau dann, wenn $a_2 < a_1$ und $a_1 \geq a_2$ genau dann, wenn $a_2 \geq a_1$.

Basis-Operationen auf Konfidenzindizes

Jetzt werden wir die vier Operationen auf der Menge aller Konfidenzindizes vorstellen. Es gibt drei binäre Operationen und zwar Schnitt ($\cap$), Vereinigung($\cup$) und Differenz (-), sowie eine unäre Operation, die Negation ($\neg$). Seien $a_1 = \langle \alpha_1, \beta_1\rangle$ und $a_2 = \langle \alpha_2, \beta_2\rangle$ zwei Konfidenzindizes. Dann gilt: $(a_1 \cup a_2) = \langle \alpha_1 \wedge \alpha_2, \beta_1 \vee \beta_2\rangle$, $(a_1 \cap a_2) = \langle \alpha_1 \vee \alpha_2, \beta_1 \wedge \beta_2\rangle$ wobei $\wedge$ Minimum und $\vee$ Maximum bedeuten. $\neg a_1 = \langle \beta_1, \alpha_1\rangle$ und $(a_1 - a_2) = a_1 \cap (\neg a_2)$.

Wir benutzen die Notation $\xi(L)$ um die Menge aller Konfidenzindizes aus dem kompletten Verband L zu repräsentieren. Weiter benutzen wir nur ξ statt $\xi(L)$.

Beispiele für Basis-Operationen:
Seien $a_1 = \langle 0.3,0.7\rangle$, $a_2 = \langle 0.5,0.7\rangle$, $a_3 = \langle 0.4,0.6\rangle$, $a_4 = \langle 0.2,0.8\rangle \in \xi$.
Es gilt nun:

- $a_4 > a_1 > a_2$ und $a_1 > a_3$,
- $(a_1 \cup a_4) = \langle 0.3 \wedge 0.2, 0.7 \vee 0.8 \rangle = \langle 0.2, 0.8 \rangle = a_4$,
- $(a_1 \cap a_4) = \langle 0.3 \vee 0.2, 0.7 \wedge 0.8 \rangle = \langle 0.3, 0.7 \rangle = a_1$,
- $a_5 = a_2 \cup a_3 = \langle 0.4, 0.7 \rangle$ mit $a_5 > a_2$ und $a_5 > a_3$,
- $a_6 = a_2 \cap a_3 = \langle 0.5, 0.6 \rangle < a_2$ und $a_6 < a_3$.

7.2.2 Konfidenzindexmenge (Confidence Index Set, CISET)

Definition: Sei S eine Menge. Eine Konfidenzindexmenge oder CISET (sprich: siset) ist eine Abbildung F:S $\rightarrow \xi$. Das bedeutet, dass man F zwei Konfidenzgrade α und β zuweist, sodass für jedes Element $x \in S$, α den Konfidenzgrad des $x \in S^c$ (das Komplement von S) darstellt, und β den Konfidenzgrad des $x \in S$. Sei $S = \{u, v, x, y, z\}$. Wir definieren $F : S \rightarrow \xi$ wie folgt: $F(u) = \langle 0.4, 0.8 \rangle$, $F(v) = \langle 0, 1 \rangle$, $F(x) = \langle 1, 0 \rangle$, $F(y) = \langle 0, 0 \rangle$ und $F(z) = \langle 1, 1 \rangle$. Man beachte, dass es möglich ist, dass ein Element gleichzeitig S^c und S mit dem gleichen Konfidenzgrad angehört. Außerdem muss die Summe von Konfidenzgraden nicht gleich 1 sein. Vielmehr kann sie zwischen 0 (wie bei y) und 2 (wie bei z) liegen.

Man sagt, dass zwei CISET F und G von S gleich sind, wenn: $F(x) = G(x)$ mit x Element von S. Man schreibt dann $F = G$.

Definition: Seien F und G zwei CISET auf S, sodass $F(x) \leq G(x)$ für $x \in S$, dann ist F Teilmenge von G und G Obermenge von F ($F \subseteq G$). Wenn F eine Teilmenge von G ist und es existiert ein $x \in S$, sodass $F(x) < G(x)$, dann ist F eine echte Teilmenge von G und G eine echte Obermenge von F ($F \subset G$).

Basis-Operationen auf CISETs

Wir haben vorher vier Basis-Operationen für Konfidenzindizes gesehen. In diesem Abschnitt werden wir fünf weitere Operationen auf CISETs vorstellen: Vereinigung, Schnitt, Differenz, Produkt und Negation. Diese Operationen erlauben uns, neue CISETs aus gegebenen CISETs zu konstruieren.
Definition: Seien S eine Menge und F, G zwei CISETs auf S:

- Seien S eine Menge und F, G zwei CISETs auf S. Die *Vereinigung* von F und G ist eine Abbildung von $S \rightarrow \xi$, definiert als $(F \cup G)(x) = F(x) \cup G(x)$, $\forall$ x Element von S.
- Seien S eine Menge und F, G zwei CISETs auf S. Der *Schnitt* von F und G ist eine Abbildung von $S \rightarrow \xi$, definiert als $(F \cap G)(x) = F(x) \cap G(x)$, $\forall$ x Element von S. Zwei CISETs F und G sind disjunkt, wenn $F \cap G =$ leere Menge.
- Seien S eine Menge und F, G zwei CISETs auf S. Die *Differenz* von F und G ist eine Abbildung von $S \rightarrow \xi$, definiert als $(F - G)(x) = F(x) \setminus G(x)$, $\forall$ x Element von S.
- Seien S eine Menge und F, G zwei CISETs auf S. Das *kartesische Produkt* von F und G ist eine Abbildung von $S\text{x}S \rightarrow \xi$, definiert als $(F\text{x}G)(x, y) = F(x) \cap G(y)$, $\forall$ (x, y) Element von $S\text{x}S$
- Sei S eine Menge und sei F ein CISET auf S. Das *Komplement* von F ist eine Abbildung von $S \rightarrow \xi$, definiert als $(-F)(x) = -F(x)$.

Relationales Modell für CISETs

In diesem Abschnitt führen wir das CISET-relationale Modell für CISETs ein, eine Kollektion von Methoden und Techniken für die Organisation von Daten. Dabei steht der Begriff CISET Relation im Mittelpunkt.

Ein Modell ist eine Abbildung einer Menge von Objekten oder Ideen der realen Welt. Ein relationales Modell für CISETs ist Bestandteil einer Datenbank, die sich widersprechende Informationen speichert. Hier sind die Daten meistens in Form von Tabellen dargestellt, die man "CISET-relationen"nennt. Alle Informationen sind in einer oder mehreren Tabellen gespeichert. Formlos kann eine CISET-relationale Datenbasis als Ansammlung von CISET-relationen begriffen werden.

Beispiel: Daten zum Straßenverkehrszustand können in Form einer Tabelle mit dem Namen Störung organisiert werden.

STÖRUNG			
S-TYP	**S-ORT**	**CI-ORT**	**S-DAUER**
Stau	A5	$\langle 0.3, 0.7 \rangle$	2h
Unfall	A4	$\langle 0.4, 0.6 \rangle$	1h
Sperrung	B6	$\langle 0.5, 0.7 \rangle$	24h

Wenn wir die Tabelle Störung anschauen, erkennen wir Folgendes:

- Störung hat vier relationale Attribute: S-TYP, S-ORT, CI-ORT, und S-DAUER.
- Jede Zeile der Tabelle stellt ein Tupel dar.
- Die obige Tabelle hat drei Tupel mit $t_1 = \{$Stau, A5, $\langle 0.3, 0.7\rangle$, 2h$\}$, $t_2 = \{$Unfall, A4, $\langle 0.4, 0.6\rangle$, 1h$\}$ und $t_3 = \{$Sperrung, B6, $\langle 0.5, 0.7\rangle$, 24h$\}$.

Formalisierung der CISET-Relation

Wir fahren jetzt fort, den Begriff einer Tabelle zu formalisieren. Sei U eine Menge von CISET-relationalen Attributen. Für jedes Attribut $A \in U$, sei $DOM(A)$ der Definitionsbereich von A. Er bezeichnet die Menge aller möglichen Werte, die in dieser Spalte auftreten können. Sei $R = \{A_1, ..., A_n\}$ eine endliche Menge von CISET-relationalen Attributen, dann wird R "CISET-relationales Schema"genannt. Abhängig von der Komplexität von $DOM(A_i)$ können CISET-Relationen in verschiedene Kategorien klassifiziert werden (Typ 0, Typ 1, ...).

TYP	$DOM(A_i)$
0	Menge
1	CISET
2	Menge von Teilmengen eines CISET
3	CISET von Teilmengen eines CISET
...	...
2_j	Menge von Teilmengen eines Bereichs vom Typ $(2_{j-1}), j > 1$.
2_{j+1}	CISET von Teilmengen eines Bereichs vom Typ $(2_{j-1}), j > 1$.

Beispiel: Das Beispiel STÖRUNG ist vom Typ 0. Die CISET-Relation STÖRUNG hat drei CISET-Tupel und vier CISET-Attribute. Die Domäne jedes dieser Attribute ist folgende:
$DOM(S\text{–}TYP)$ = Menge aller Typen,
$DOM(S\text{–}ORT) = \{A5, A4, B6\}$,
$DOM(CI\text{–}ORT) = \xi$,
$DOM(S\text{–}DAUER)$= Menge aller Ziffern.
$DOM(S\text{–}TPY)$, $DOM(S\text{–}ORT)$, $DOM(CI\text{–}ORT)$
und $DOM(S\text{–}DAUER)$ erfüllen die Definition eines Typs 0 CISET Relation.

Wie in [Nai03] zu sehen ist, ist es in der Praxis am einfachsten, eine CISET-Relation vom Typ 0 zu implementieren, weshalb wir uns in dieser Arbeit nur auf Typ 0 CISETs konzentrieren.

Semantik für CISET-Relationen

Sei die folgende Tabelle eine CISET-Relation vom Typ 0:

STÖRUNG				
S-TYP	**S-ORT**	**CI-ORT**	**S-DAUER**	**CI**
Stau	A5	$\langle 0.3, 0.7\rangle$	2h	$\langle 0.1, 0.8\rangle$
Unfall	A4	$\langle 0.4, 0.6\rangle$	1h	$\langle 0.2, 0.9\rangle$
Sperrung	B6	$\langle 0.5, 0.7\rangle$	24h	$\langle 0.3, 0.7\rangle$

Die Semantik des ersten Tupels ist: *" Es gibt Stau auf der A5, der 2 Stunden anhalten kann, und der Ort des Staus hat einen Konfidenzindexwert von* $\langle 0.3, 0.7\rangle$*. Die Tatsache, dass es Stau auf der A5 gibt, der 2 Stunde dauern kann, hat einen Konfidenzindexwert von* $\langle 0.1, 0.8\rangle$*".* Man beachte, daß es einen essenziellen Unterschied zwischen der Interpretation der Attribute *CI–ORT* und *CI* gibt. Das Attribut *CI–ORT* ist ein gewöhnliches Attribut, das eine Information über den Stau enthält. Das Attribut CI hingegen trifft keinerlei Aussage über den Stau. Stattdessen enthält CI eine Information über die Gültigkeit des Tupels selbst, und weist dem Tupel ein Konfidenzindex zu. So unterscheidet sich CI von allen anderen Attributen, da das Attribut CI ein Tupel-Attribut darstellt, das das Tupel selbst qualifiziert.

Eigenschaften von CISET-Relationen

1. Die Reihenfolge der Attribute ist unwichtig.
   Zum Beispiel sind die folgenden zwei CISET-Relationen gleich:

STÖRUNG				
S-TYP	**S-ORT**	**CI-ORT**	**S-DAUER**	**CI**
Stau	A5	$\langle 0.3, 0.7\rangle$	2h	$\langle 0.1, 0.8\rangle$
Unfall	A4	$\langle 0.4, 0.6\rangle$	1h	$\langle 0.2, 0.9\rangle$
Sperrung	B6	$\langle 0.5, 0.7\rangle$	24h	$\langle 0.3, 0.7\rangle$

STÖRUNG 2				
S-TYP	**S-DAUER**	**CI-ORT**	**CI**	**C-ORT**
Stau	2h	$\langle 0.3, 0.7\rangle$	$\langle 0.1, 0.8\rangle$	A5
Unfall	1h	$\langle 0.4, 0.6\rangle$	$\langle 0.2, 0.9\rangle$	A4.
Sperrung	24h	$\langle 0.5, 0.7\rangle$	$\langle 0.3, 0.7\rangle$	B6

Es fällt auf, dass das erste Tupel in der Relation STÖRUNG sowie das erste Tupel in der STÖRUNG 2 CISET-Relation die gleiche Information beinhalten, sodass man die Aussage treffen kann, dass „Verschiedenen Quellen entsprechend ein Stau auf der A5 ist, der 2 Stunden dauern kann, zu 10% unsicher und zu 80% sicher. Außerdem hat der Ort des Staus einen Konfidenzwert von $\langle 0.3, 0.7\rangle$ ´´.

2. Die Reihenfolge der Tupel in einer CISET-Relation ist unwichtig:

STÖRUNG				
S-TYP	**S-ORT**	**CI-ORT**	**S-DAUER**	**CI**
Stau	A5	$\langle 0.3, 0.7\rangle$	2h	$\langle 0.1, 0.8\rangle$
Unfall	A4	$\langle 0.4, 0.6\rangle$	1h	$\langle 0.2, 0.9\rangle$
Sperrung	B6	$\langle 0.5, 0.7\rangle$	24h	$\langle 0.3, 0.7\rangle$

STÖRUNG 3				
S-TYP	**S-ORT**	**CI-ORT**	**S-DAUER**	**CI**
Unfall	A4	$\langle 0.4, 0.6\rangle$	1h	$\langle 0.2, 0.9\rangle$
Sperrung	B6	$\langle 0.5, 0.7\rangle$	24h	$\langle 0.3, 0.7\rangle$
Stau	A5	$\langle 0.3, 0.7\rangle$	2h	$\langle 0.1, 0.8\rangle$

Das Tupel 1 in der CISET-Relation STÖRUNG erscheint als Tupel 3 in der CISET-Relation STÖRUNG 3. Der Informationsgehalt dieser zwei Tupel ist gleich.

3. Es gibt keine identischen Tupel in einer Relation.

4. Eine CISET-Relation ist in 1Normal-form.

   Eine CISET-Relation ist eine Menge von Abbildungen. Jede Abbildung entspricht einem Tupel in der CISET-Relation. Nach der Definition einer CISET-Relation weist jede Abbildung jedem Attribut einen einzelnen Wert zu. Diese Eigenschaft bezeichnet man als *1Normal-form*. Eine CISET-Relation ist in erster Normal-form ($1NF$), wenn jeder Attributwert ein atomarer Wert ist, also insbesondere keine Liste oder Menge von Werten. Es fällt auf, dass ein Konfidenzindex, ähnlich wie bei komplexen Zahlen, als einzelner Wert behandelt wird.

7.3 Anwendung des CISET-Ansatzes auf das Szenario

Folgendes Beispiel aus [Nai03] zeigt, wie man den CISET-Ansatz auf sich widersprechende Informationen anwenden kann:

Beispiel: *„Fabrik X in der Stadt Y produziert biologische Waffen"hat einen Konfidenzindex von* $\langle 0.3, 0.7\rangle$ *. Zu Beginn haben wir keine Information über die Fabrik X, folglich hat die Aussage "Fabrik X produziert biologische Waffen"den Konfidenzindex* $\langle 1, 0\rangle$*. Nach dem Eingang von Informationen aus einer ersten Quelle, werden diese analysiert und* β *erhält den Wert 0.6 für die Aussage oder anders ausgedrückt der Konfidenzindex wird in* $\langle 1, 0.6\rangle$ *umgewandelt. Eine zweite Quelle liefert nun Informationen, die der ersten Aussage widersprechen. Nach Untersuchung der zweiten Informationen wird* α *auf 0.3 gesetzt, womit sich der Konfidenzindex zu* $\langle 0.3, 0.6\rangle$ *abändert.*

Gehen wir auch ähnlich bei dem Beispiel unseres Szenarios vor, dann liefert uns die Aussage „Es ist ein Unfall auf der A5 passiert."einen Konfidenzindex $\langle 0.3, 0.8\rangle$. Am Anfang haben wir noch keinen Eintrag bezüglich des Verkehrzustands in der Datenbank. Nach Empfang der positiven Aussage der ersten Quelle (Die erste Quelle ist eine Person, die bei der Zentrale registriert ist, und bisher stets richtige Informationen gemeldet hat) wird die Zuverlässigkeit geprüft, und gemäß Vertrauensgrad der Quelle wird β der Wert 0.8 zugewiesen. In anderen Worten, der Konfidenzindex ist auf $\langle 1.0, 0.8\rangle$ geändert. Nach einiger Zeit trifft die Meldung einer zweiten Quelle ein, die der ersten Meldung widerspricht. Da nichts über diesen Melder bekannt ist, möchte man die Unfallmeldung nicht sofort streichen, da sie eventuell eine wichtige Information beinhalten kann. Es ist beispielsweise möglich, dass der Unbekannte bei der Meldung einen Aspekt falsch widergegeben hat, und so nur die Autobahnabkürzung nicht stimmt. Nach sorgfältiger Auswertung jener Information wird der Wert α auf 0.3 gesetzt. So ändert sich der Konfidenzindex zu $\langle 0.3, 0.8\rangle$. Die Information, dass es einen Unfall auf der A5 gibt, hat dann einen Konfidenzindex $\langle 0.3, 0.8\rangle$. So ist es mit dem CISET-Ansatz möglich, die beiden widersprüchlichen Aussagen in die Datenbank aufzunehmen. Jetzt bleibt der Eintrag in der Datenbank unverändert, bis eventuell weitere Informationen zu dem Unfall eintreffen. Wird noch eine Meldung eines zuverlässigen Melders gemeldet wird der β-Wert erhöht, ansonsten bleibt er unverändert. Dieses Verfahren wird auch tatsächlich in der Praxis angewendet, wie es z.B. in [Koo04] beschrieben wird. Da Konfidenzindizes je nach Vertrauensgrad gesetzt werden, sind es subjektive Daten.

Für die Initialisierung in [Nai03] wird davon ausgegangen, dass jeder Sachverhalt, über den noch keine Information vorliegt, als falsch anzunehmen ist. Deswegen wird stets mit $\langle 1.0, 0.0\rangle$ initialisiert. In der Realität ist ein Sachverhalt, über den keine Information vorliegt, zumindest für möglich zu halten, solange nichts dagegen spricht. Also hätte man auch auf $\langle 0.0, 0.0\rangle$ initialisieren können.

7.4 CISET-relationale Algebra

In diesem Abschnitt untersuchen wir die CISET-relationale Algebra. Diese sind nur auf kompatible relationale Schemata anwendbar.
Definition: Zwei Relationen $R = \{A_1, ...A_n\}$ und $S = \{B_1, ..., B_m\}$ sind kompatibel, wenn:

1. n = m, d.h. die Anzahl der Attribute ist gleich;

2. $DOM(A_i) = DOM(B_i)\ \forall\ i = 1, 2, .., n$

7.4.1 Vereinigung, Schnitt, Differenz

Seien R und S zwei Relationen. Relationale Operationen, wie Vereinigung und Schnitt sind genau dann Anwendbar, wenn R und S kompatibel sind. Im folgenden nehmem wir an, dass R und S kompatibel sind. Wie schon im Abschnitt „Relationales-Modell für CISETs"zu sehen ist, hat eine CISET-Relation, CISETs, die sich auf einzelne Attribute beziehen und CISETs, die sich auf das ganze Tupel beziehen.
Vereinigung: Wenn man zwei kompatible Relationen vereinigt, bekommt man eine neue Relation, die ebenfalls kompatibel ist. Man bekommt die Vereinigung von zwei CISET-Relationen R und S indem man alle Zeilen von R und S kombiniert.
Formal ergibt sich folgendes: $R = \{(A_i, b_{i,1}, ..., b_{i,m})\}$ mit $i \in I$ und $S = \{(A_j, b_{j,1}, ..., b_{j,m})\}$ mit $j \in J$ wobei A_i und A_j relationale Attribute sind und $\{b_{i,1}, ..., b_{i,m}, b_{j,1}, ..., b_{j,m}\}$ Elemente aus ξ sind.
$R \cup S = \{A_i, b_{i,1} \cup b_{j,1}, ..., b_{i,m} \cup b_{j,m} \mid i \in I, j \in J, A_i = A_j\} \cup \{(A_i, b_{i,1} \cup b_{j,1}, ..., b_{i,m} \cup b_{j,m}) i \in I, \neg\exists j \in J : A_i = A_j\} \cup \{(A_i, b_{i,1} \cup b_{j,1}, ..., b_{i,m} \cup b_{j,m}) j \in J, \neg\exists i \in I : A_i = A_j\}$
Beispiel: Wir benutzen folgende zwei CISET-Relationen, um die Vereinigung zu zeigen.

R			
A	**B**	**C**	**CI**
a_1	b_1	c_1	$\langle 0.3, 0.7\rangle$
a_1	b_1	c_2	$\langle 0.6, 0.2\rangle$
a_3	b_3	c_3	$\langle 0, 1\rangle$
a_4	b_4	c_4	$\langle 1, 1\rangle$

S			
A	**B**	**C**	**CI**
a_1	b_2	c_2	$\langle 0.2, 0.5\rangle$
a_2	b_2	c_1	$\langle 0.1, 0.8\rangle$
a_3	b_3	c_3	$\langle 0, 1\rangle$
a_4	b_4	c_4	$\langle 1, 1\rangle$

$R \cup S$			
A	**B**	**C**	**CI**
a_1	b_1	c_1	$\langle 0.3, 0.7\rangle$
a_1	b_1	c_2	$\langle 0.6, 0.2\rangle$
a_1	b_2	c_2	$\langle 0.2, 0.5\rangle$
a_2	b_2	c_1	$\langle 0.1, 0.8\rangle$
a_3	b_3	c_3	$\langle 0, 1\rangle$
a_4	b_4	c_4	$\langle 0, 1\rangle$

Schnitt: Den Schnitt von zwei CISET-Relationen bekommt man indem man alle Zeilen von R die identisch zu S sind nimmt. Formal ergibt sich folgendes: $R \cap S = \{A_i, b_{i,1} \cap b_{j,1}, ..., b_{i,m} \cap b_{j,m} \mid i \in I, j \in J, A_i = A_j\}$
Beispiel: Sei R und S aus obigen Beispiel:

$R \cap S$			
A	**B**	**C**	**CI**
a_3	b_1	c_3	$\langle 1, 1\rangle$
a_4	b_4	c_4	$\langle 1, 0\rangle$

Man erkennt, dass das Tupel $(a_4, b_4, c_4, \langle 1, 0\rangle)$ normalerweise nicht in $R \cap S$ vorkommt, da es den Konfidenzindex $\langle 1, 0\rangle$ (Contra-wert $= 1$, Pro-wert $= 0$) besitzt, also eigentlich als "falsch" oder "ungültig" verstanden werden kann.
Differenz: Differenz bei relationaler Algebra hat die gleiche Definition wie die Differenz des Basis-Operation, die wir schon gesehen haben.
$R - S = \{A_i, b_{i,1}, ..., b_{i,m} \mid i \in I, \neg\exists j \in J, A_i = A_j\} \cup \{A_i, b_{i,1} - b_{j,1}, ..., b_{i,m} - b_{j,m} \mid i \in I, j \in J : A_i = A_j\}$

7.4.2 Selektion und Projektion

Selektion: Selektionsoperationen erzeugen eine neue CISET-Relation, wenn man Tupel selektiert die gegebenen Prädikate berücksichtigen. Betrachten wir unsere vorige Relation R, für $a \in DOM(A), \sigma_{A=a}R$ ist eine Selektionsoperation, die alle Tupel von R enthalten, mit Attribut A gleich a. Wenn $DOM(CI) = \xi$ und $\langle \alpha, \beta \rangle$ ein Konfidenzindex ist, dann ist $\sigma_{(A=a) \wedge (CI.LL < \alpha) \wedge (CI.UL \geq \beta)}R$ eine Selektion auf R, die alle Tupel mit Attribut A gleich a enthält und dessen CI-Wert aus Elementen besteht, deren Infimum kleiner als α ist und Supremum größer oder gleich β ist.
Projektion: Projektion bei CISET-relationaler Algebra ist ähnlich wie bei der normalen Relatioanlen Algebra.

7.4.3 Produkt und Join

Produkt: Das Produkt von zwei CISET-Relationen erzeugt eine Liste von möglichen Tupelpaare. D.h., wenn wir eine CISET-Relation mit drei Zeilen und eine andere mit 4 Zeilen haben, besteht die CISET-Relation, die man nach dem Produkt bekommt, aus 12 Zeilen. Formal: $R \text{x} S = \{A_i, A_j, b_{i,1} \cap b_{j,1}, ..., b_{i,m} \cap b_{j,m} \mid i \in I, j \in J\}$
Joins: Bei Joins wird zuerst das Produkt berechnet, danach werden alle Tupel mit identischen Werte selektiert und anschließend werden identische Attribute durch Projektion eliminiert.

7.4.4 Division

Die Divisionsoperation ist ähnlich wie ein Ganzzahldivision. Seien zwei Relationen U und V mit $U = \{(a_{i,1}, ..., a_{i,n}, b_1, ..., b_k, c_{i,1}, ..., c_{i,m}) \mid i \in I\}$ und $T = \{(a_{j,1}, ..., a_{j,n}, c_{j,1}, ..., c_{j,m}) \mid j \in J\}$. Das Ergebnis der Division wird in drei Stufe berechnet. Erstens werden alle Tupel t_i aus U gesucht, für die ein Tupel t_j aus T existiert, so dass $\forall\ x \in \{1, ..., n\}$: $a_{i,x} = a_{j,x}$. Zusätzlich muss gelten, dass $c_{j,1} \leq c_{i,z}\ \forall\ z \in \{1, ..., m\}$. Zweitens wird eine Projektion auf alle Attribute der Relationen, die im ersten Schritt erhalten wurden und welche nicht in der Relation T enthalten sind, durchgeführt. Drittens werden alle Tupel, die in den Attributen $b_1, ..., b_k$ übereinstimmen, vereinigt.

7.5 Fazit

Der CISET-Ansatz ist ein komplexes Erweiterung des relationalen Modells, da die Vertrauensgrade subjektiv sind. Er ist aber ein gutes Modell insbesondere im Verkehrbereich, denn er berücksichtigt auch imperfekte und widersprüchliche Daten, die im Verkehrbereich öfters vorkommen. Bei Zugriff von Daten, zeigt sich die CISET-Ansatz unkompliziert, da relationale Algebra einfach zu ermitteln ist und Konfidenzindizes leicht zu interpretieren sind. Die Verwendung der Initialisierung ist allerdings etwas unklar.

Literaturverzeichnis

[Koo04] Erik Koop. *Datenbankunterstützung für imperfekte Daten im Verkehrsumfeld.* PhD thesis, IPD, Universität Karlsruhe (TH), 2004.

[Nai03] Premchand S. Nair. *Uncertainty in Multi-Source Databases.* Springer, 2003.

Eric Ndengang Foyet

8

Kopplung verteilter Datenbanksysteme

8.1 Einleitung

8.1.1 Problematik

Durch die rasche Entwicklung des World Wide Web hat sich die Zahl der Intra - /Internet-Anwendungen, die man als heterogene Daten integrieren muss, sehr schnell erhöht. Das Hauptproblem bei diesen Anwendungen ist der Datenaustausch. Nehmen wir zum Beispiel an, dass wir Datenbankanwendungen schreiben wollen, die Daten aus vielen verschiedenen Datenquellen miteinander abgleichen, so steht man vor dem Problem, dass Daten von einem Format in ein anderes übersetzt werden müssen, um in Abfragen verwendet werden zu können.
Die bisherige Art, „auf die Schnelle mal eben“ einen Übersetzer zu schreiben, führt zu schlecht dokumentierter und kaum wieder verwendbarer Software. Es wurde viele Datenmodelle und allgemeine Abfragesprachen entworfen, um das Kombinieren der Informationen von vielen unterschiedlichen Quellen zu unterstützen.

8.1.2 Wrapper/Mediator-basierte Architekturen

Eine Möglichkeit, unterschiedliche Informationsquellen zu integrieren, die die gleichen Entitäten der realen Welt darstellen, ist die Herstellung eines Mediators, der fähig ist, Fragen über diese Entitäten zu beantworten [Wie92].
Mediatoren verwenden rohe Quellen (passend angeschlossen durch Wrappers) und /oder andere Mediatoren, um auf die Anfragen zu antworten,

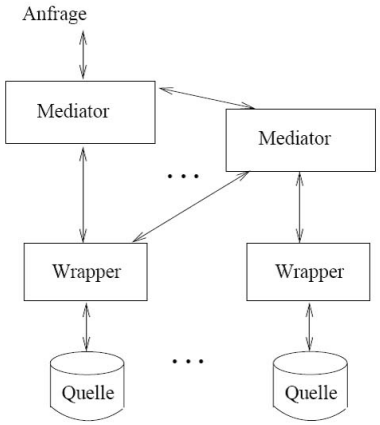
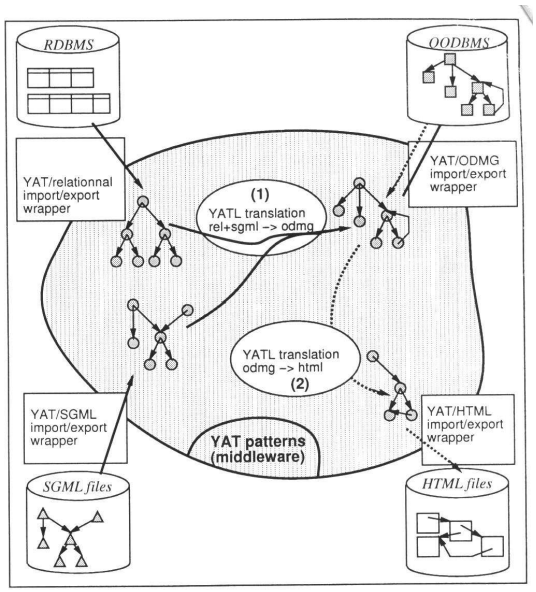


Abbildung 8.1: Ein Netzwerk von Mediatoren und Informationsquellen

genau so wie es in der Abbildung 8.1 dargestellt ist.

Ein Mediator ist die Ebene zwischen den Benutzern und den Datenquellen. Sie wird auch *Middleware* genannt. Ein Wrapper, auch **Translator** genannt bietet dem Mediator einen homogenen Zugriff zu den heterogenen Datenquellen.
Zwei Wrapper/Mediatorarchitekturen sind bekannt:

1. Bottom-Up Ansatz
   - Wrapper bilden den Inhalt der Datenquellen in ein allgemeines Datenmodell ab.
   - Mediator integriert dies zu einer globalen Datenbasis.
   - Benutzer stellt Anfragen an den Mediatoren, die auf der globalen Datenbasis ausgewertet werden.
2. Top-Down Ansatz
   - Mediator stellt eine allgemeine Anfragesprache zur Verfügung.
   - Benutzer stellt Anfragen an den Mediator, die von diesem aufgeteilt werden und die Teile an die Wrapper weitergegeben werden.
   - Wrapper stellen die Anfragen an die Teildatenbasen, und geben Ergebnisse zurück.
   - Mediator verknüpft diese Ergebnisse.




```
< ! DOCTYPE   brochure [
< ! ELEMENT   brochure  - -  (number, title,
                             model, desc, spplrs) >
< ! ELEMENT   number    - -  (#PCADATA) >
< ! ELEMENT   title     - -  (#PCADATA) >
< ! ELEMENT   model     - -  (#PCADATA) >
< ! ELEMENT   desc      - -  (#PCADATA) >
< ! ELEMENT   spplrs    - -  (sup)* >
< ! ELEMENT   supplier  - -  (name,address) >
< ! ELEMENT   name      - -  (#PCADATA) >
< ! ELEMENT   address   - -  (#PCADATA) > ] >
```

Abbildung 8.2: Abbildung 8.2a links: Das Szenario. Abb. 8.2b rechts: Die SGML-Broschüre.

8.2 Semistrukturierte Datenmodelle

8.2.1 Das YAT Datenmodel

Das YAT System und Problemvorstellung

Das YAT-System (Yet another tree-based system) wurde am INRIA(Institut National de Recherche en Informatique et Automatique) in Frankreich als Werkzeug zur Spezifikation und Implementierung von Datenkonvertern für heterogene Datenquellen entwickelt [SJ97, Clu97, CDSS98].
Um das Problem zu veranschaulichen, betrachten wir das folgende Beispiel: Eine Autohändlerfirma möchte eine Intranet-Anwendung erstellen. Unter anderem speichert die Firma Informationen über seine Händler in einem relationalen Datenbanksystem und die Beschreibungen der verkauften Autos in SGML Dokumenten. Das Ziel ist es, alle Informationen in einer Objektdatenbank zu integrieren und eine HTML-Schnittstelle zur Verfügung zu stellen, damit die Angestellten sie auf dem Netz ansehen können.
Eine globale Ansicht der Anwendung, ist in Abbildung 8.2a zu sehen. Die Abbildung 8.2b stellt eine SGML-Broschüre dar, die Informationen über verkaufte Autos enthält.

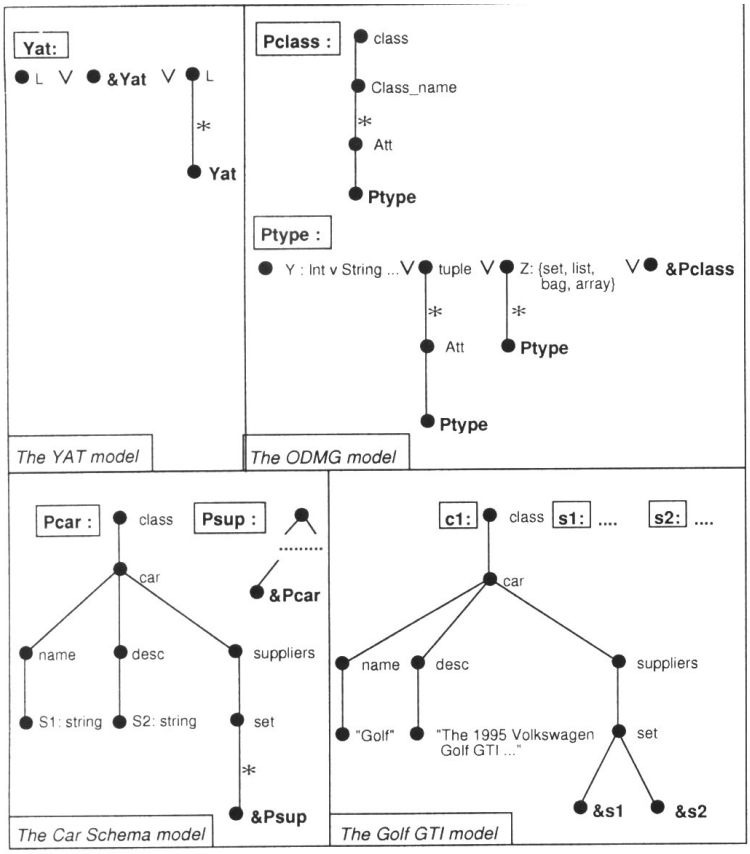


Abbildung 8.3: Darstellung ODMG-Daten mit YAT.

Modellierung

Abbildung 8.3 zeigt vier Modelle, die unterschiedliche Niveaus von Datenrepräsentation veranschaulichen. In der oberen linken Ecke der Abbildung, ist das Yat Modell zu sehen. Auf der rechten Seite steht eine erste Instanz von Yat, die ein ODMG Schema modelliert. Das Autoschemamodell auf der unteren linken Seite ist ein Fall beider vorhergehenden Modelle und stellt Daten dar, die zu dem spezifischen ODMG Schema passen. Schließlich stellt das Golfmodell die tatsächliche Datenbank dar und ist eine Instanz des Autoschemas.

Das YAT-Modell verwendet statt der Bezeichnung Schema den Ausdruck „***Pattern***“(Muster). Diese Muster erlauben die Realisierung der Modellierbarkeit aller Realweltdaten und bestehen aus einer Menge von geordneten Bäumen. Die Knoten eines Baumes sind mit Variablen oder Konstanten beschriftet. Es gibt eigentlich zwei Variablensorten:

- Datenvariablen wie Z.B L in YAT, class-name in ODMG, s1 in Car Schema in der oberen Abbildung.
- Mustervariablen wie Z.B **YAT**,**Pclass**,**Pcar**

Grundsätzlich ist die Domäne einer Datenvariable die Menge aller ihrer Konstanten und Variablennamen. Die Domäne der Yat Variablen L beinhaltet Z.B Konstanten wie class, car oder „Golf“ und Variablen wie class-name und att.
Die Domäne einer Mustervariable ist die Menge aller ihren Musterinstanzen. Interessant an dem YAT-Modell ist die Möglichkeit bereits definierte Muster zu instanzieren. Die Instanzierung geschieht durch:

- Die Benennung von Knoten eines Musters mit Namen von anderen Mustern
- Das Setzung einer Referenz auf einen Knoten eines Musters.

In Abbildung 8.3 ist zu sehen, dass **Pclass** eine Instanz von **Yat** ist und **psup** eine Instanz von **Pclass** ist.
Bei der Instanziierung eines Modells dient der *-Operator dazu, die Anzahl der erlaubten Ausgangskanten aus einem Knoten zu definieren. Dieser spezifiziert, dass die Kante entweder nicht oder beliebig oft auftreten. Wird der *-Operator an einer Kante weggelassen, zeigt dies, dass die Kante genau einmal auftreten muss.

Anfragesprache „YATL“

YATL ist die Anfragesprache, die in Yat verwendet wird. Sie ist eine regelbasierte Sprache bei der, jede Regel aus einem Kopf und einem Rumpf besteht. Der Kopf der Regel enthält Informationen, wie die gefundenen Daten neu strukturiert werden müssen. Der Rumpf enthält Muster und Prädikate.
Lassen wir uns dieses mit einem Beispiel veranschaulichen. In Abbildung 8.4a wird ein Yat Programm dargestellt, das für jede SGML-Broschüre ein Lieferantenobjekt erzeugt.

Der Rumpf besteht aus:

- Einem Muster, das die SGML-Broschüre darstellt.
- Einem einfachen Prädikat, das ältere Autos herausfiltert.
- 2 externen Funktionen, die die Stadt und die Postleitzahl aus einer Adresse extrahieren.

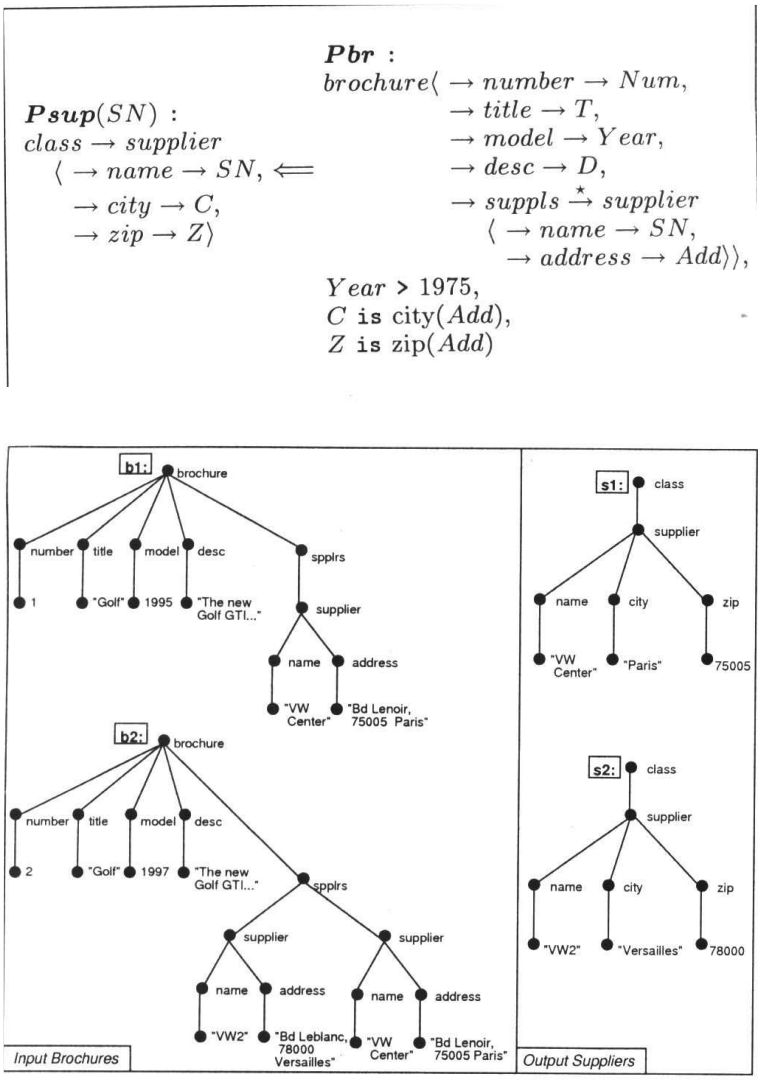


Abbildung 8.4: Abb 8.4a oben: Ein YAT Programm, das Lieferantenobjekte erzeugt. Abb 8.4b unten: Anwendung des Programms auf 2 SGML Broschüre.

Der Kopf besteht aus einem einzelnen Muster dessen Name mit einer Variable (SN) parametrisiert ist. Dabei ist SN der Name eines Lieferanten.
Parametrisierte Musternamen entsprechen explizite Skolemfunktionen. Skolemfunktionen dienen dazu, einen Objekt für jeden Lieferantenname zu erzeugen, und zwar entspricht die eindeutige ID des Objekts den Parameterwerten. In Abbildung 8.4b wird eine Anwendung dieses Programms auf 2 SGML-Broschüre dargestellt. Die beiden externen Funktionen haben für jedes Objekt jeweils die Stadt und die Postleitzahl extrahiert.
Das zweite Yat Programm aus Abbildung 5 erzeugt für jede Broschüre ein Autoobjekt. Interessant an diesem Programm ist die Benutzung einer Referenz. Der Rumpf ist derselbe wie der von oben. In dem Kopf referenziert jedes Auto seine Lieferanten. Die Erzeugung der Verbindungen von Autos zu Lieferanten wird durch die Nutzung von dem parametrisierten Musternamen **Psup**(SN) angefasst. Das Symbol & wird hier für die Referenz auf einen Lieferanten benutzt. Das Symbol {}wird für die Erzeugung eines Knoten mit mehreren Söhnen benutzt.

8.2.2 Das OEM Datenmodel

Das OEM-Datenmodell wurde ursprünglich für das TSIMMIS-Projekt entwickelt. TSIMMIS - The Stanford-IBM Manager of Multiple Information Sources - ist ein System zur Integration von Informationen [GMPQ$^+$95, Vas].

TSIMMIS: Konzept

In Tsimmis wird als Architektur der „Top Down Ansatz“angewendet, wobei folgendes Szenario läuft:

- Benutzer stellt Anfrage an einen Mediator
- Mediator gibt Anfrage an die Translatoren weiter
- Translatoren transformieren die Datenobjekte der Quellen in OEM und sind in der Lage, Anfragen aus den Quellen zu beantworten.
- Ergebnisse der Quellen werden in OEM an den Mediator zurückgegeben.
- Mediator integriert die Antworten der Quellen.

Abbildung 8.5 stellt die Architektur von Tsimmis dar.

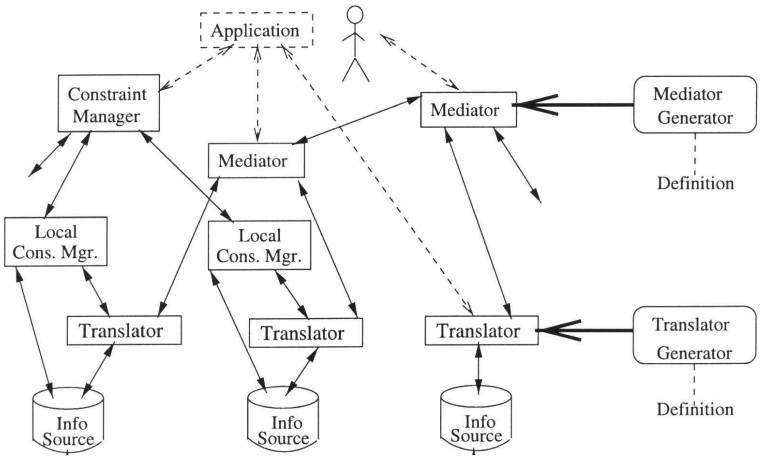


Abbildung 8.5: Tsimmisarchitektur.

Das Objekt Exchange Modell

OEM ist ein Datenmodell (ein sehr einfaches „selbstbeschreibendes“ Objektmodell) zur Kommunikation zwischen Translatoren und Mediatoren in TSIMMIS.
OEM kennt nur Objektidentitäten und Schachtelung als Konzept. Jedes Objekt besteht aus :

- *OID*:Ein Wert, der das Objekt eindeutig identifiziert.
- *Label*: Beschreibung des Objektinhalts. Damit kann das Objekt identifiziert werden und die Bedeutung des Objekts beschrieben werden.
- *Typ*: Datentyp des Werts. Verwendung findet hier entweder ein atomarer Typ oder ein Set-Typ.
- *Wert*: der Wert des Objekts.

Betrachten wir nun das folgende Beispiel, um OEM besser zu erklären. Abbildung 8.6 stellt eine Menge von OEM Objekten dar.

Auf der oberen Stufe befindet sich ein Root Objekt dessen Label Buchhandlung ist. Sein Wert ist eine Menge von Objekten vom Typ „set“. Unter der Objektmenge, die den Wert der Buchhandlung bildet, ist ein Objekt zu sehen, dessen Label Buch ist. Das Buchobjekt hat einen Wert dessen Typ auch „set“ ist. Jedoch anders als das Bibliothekobjekt, wird „set“ hier benutzt, um einen Satzaufbau zu simulieren.
Als Wert des Buchobjekts erwarten wir eine Menge von Subobjekten mit un-

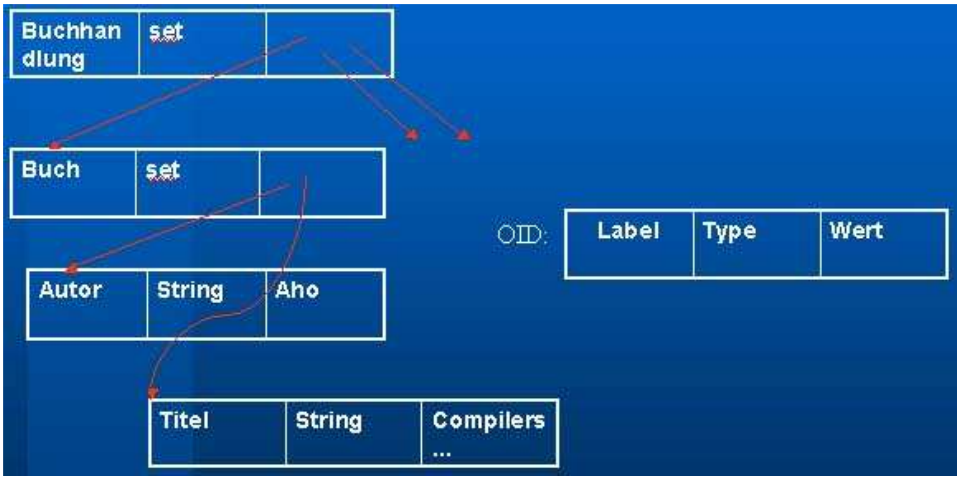


Abbildung 8.6: Versammlung von OEM Objekten.

terschiedlichen Labels. Man sieht doch 2 Subobjekte mit den Labels **Autor** und **Titel**.

Tsimmissprachen

1. The Mediator Specification Language (MSL)
   Mediatoren werden in MSL (eine regelbasierte Anfragesprache an OEM) programmiert. MSL-Regeln sind anhand dieser Formel beschrieben:

   $$Head(Vars) :- body(Vars, databases)$$

   - Rumpf: Pattern, das mit geeigneten Variablenbindungen erfüllt werden muss.
   - Der Kopf beschreibt die Struktur des OEM-Objekts, das generiert werden soll.

   In MSL werden Objekte verschiedener Quellen durch

   „<objekt>@source“

   identifiziert.

   Führen wir jetzt ein Beispiel durch. Sei die Anfrage gegeben: „Finde alle Bücher deren Autor Aho ist“. Dann sieht die entsprechende Regel dafür so aus:

$< BuchtitelX >: -$

$< Buchhandlung\{< Buch\{< Titel\ X >< Autor\ ''Aho'' >\} >\} > @S1$

S1 könnte hier entweder einen Mediator oder einen Translator sein. Der Head der Frage zeigt an, dass jeder Wert, den X bindet, im Ergebnis als Wert eines Objekts eingeschlossen ist, der Buchtitel beschriftet wird. Aus technischen Gründen werden diese Objekte Subobjekte eines Objekts mit dem Label *Antwort*, das durch die Frage produziert wird.

2. The Wrapper Specification Language (WSL)
   Wrapper werden in WSL programmiert. Jede WSL-Regel besteht aus

   - Einem „WSL-Template“
   - Einer Aktion in der Anfragesprache der zugrunde liegenden Datenbank.

   Bekommt der Wrapper eine MSL-Anfrage, die auf das WSL-Template passt, wird die Aktion mit der entsprechenden Variablenbindung ausgeführt.
   Als Beispiel möchten wir einen Wrapper für eine Quelle erstellen, der ein bibliographisches Suchsystem ist. Wir können Regeln wie folgt definieren:

   $< booksX >: -$

   $< library\{X :< book\{< title\ X >< author\ \$AU >\} >\} > @S1$

   - Der erzeugte Wrapper überprüft eine Frage und vergleicht sie mit diesem und anderen Mustern, die in der Wrapper-Spezifikation angegeben werden.
   - Die Frage
     $< booksB >: -$
     $< library\{B :< book\{< title\ X >< author\ ''Aho'' >\} >\} > @S1$
     würde eine “native” Frage zur Quelle S1 erzeugen, die Bücher anfordert, die von Aho geschrieben wurden.
   - Schließlich wird den String dann zur Quelle S1 geführt.

3. The LOREL Query Language
   In Tsimmis können Benutzer Anfragen in *MSL* oder *LOREL* stellen. LOREL (Lightweight Objekt Repository Language) ist eine auf OQL basierende Abfragesprache für das OEM Modell [AQM+97, MAG+97]. Die semistrukturierten Teile der Abfragen mit Lorel sind pfadorientiert. Es ist auch eine Abfragesprache für das LORE, Projekt in Stanford, das DBMS für das OEM Datenmodell aufbaut.
   Eine komplette Spezifikation von LOREL, einschließlich die formale

Syntax und Semantik werden in Quass, Rajaraman, Sagiv, Ullmanund in Widom explizit erklärt.
Lassen wir nun ein Beispiel ansehen. Lautet die Frage: „Find the books of which Aho is an Author“, so sieht der Code wie folgt aus:

Select library.book.title Where library.book.author =„Aho“

8.3 Gemeinsamkeiten und Unterschiede zwischen den Datenmodellen

OEM wurde für den Datenaustausch entwickelt und dient zur Kommunikation zwischen Wrappern und Mediatoren in Tsimmis. Damit konnten unstrukturierte Datenmengen einfach in eine gesamte Sicht der Daten integriert werden.
Der Einsatz von YAT wurde in Bezug auf den Datenaustausch zwischen verschiedensten Datenbanken, die beliebige, festgelegte Datenmodelle einsetzen, optimiert.
YAT ist ein System zum Erstellen von Konvertern. Tsimmis, ein System zur Integration von Informationen von verschiedenen Datenquellen.

8.4 Fazit

Folgende Themen wurden in diesem Paper behandelt:

- Das Wrapper /Mediator Konzept
- Das **YAT-Modell**, das aus einer Sammlung von Mustern besteht, die die ankommenden Daten und die Ergebnisse der Konvertierung beschreiben.
- **YATL**, die Sprache für die spezifikation der Datenumsetzung.
- Das Tsimmis Konzept und seine Architektur.
- **Das OEM Datenmodell**, ein objektorientiertes Model, das Objekt Label verwendet um Attribute -und Informationsklassen von Objekten darzustellen.
- Die Sprache **LOREL** für OEM Objekten
- **MSL**, eine für Mediatoren regelbasierte Abfragesprache, die OEM verwendet.
- **WSL**, eine zu MSL erweiterte Sprache, die die Beschreibung der Datenquelleninhalte erlaubt und Anfragenfähigkeiten erleichtert.

Literaturverzeichnis

[AQM+97] S. Abiteoul, D. Quass, J. McHugh, J. Widom, and J.L.Wiener. The LOREL Query Language for Semistructured Data. In *International Journal on Digital Libraries*, pages 68–88, 1997.

[Beh02] Jörg Behl. Analyse und Vergleich von semistrukturierten und objektorientierten Datenmodellen. In *Studienarbeit an der Universität Stuttgart am Institut für Parallele und verteilte Höchstleistungsrechner*, 2002.

[CDSS98] S. Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your Mediators need Data Conversion! In *Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, 1998.

[Clu97] S. Cluet. Modeling and Querying Semi-structured Data. In *SCIE*, pages 192–213, 1997.

[GMPQ+95] H. Garcia-Molina, Y. Papakonstantinou, Dallan Quass, Anand Rajaraman, Y. Sagiv, Jeffrey Ullman, V. Vassalos, and Jennifer Widom. The Tsimmis Approach to Mediation. In *The Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95), 27 - 29 June 1995, Hotel Carlton, Naharia, Israel*, 1995.

[Kal99] L. Kalinichenko. Integration of Heterogeneous Semistructured Data models in the Canonical One, 1999.

[MAG+97] J. McHugh, S. Abiteoul, R. Goldman, D. Quass, and J. Widom. LORE: A Database Management System for semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[SJ97] S.Cluet and J.Siméon. Data Integration based on data conversion and restructuring. In *Technical report ,Verso database group INRIA*, 1997.

[Vas] V. Vassalos. Wrapper specification and query processing in the tsimmis project.

[Wie92] G. Wiederhold. Mediators in the architekture of future information systems. In *in IEEE computer 25:3*, pages 38–49, 1992.

Teil V

Anwendungen

Christian Kuhn

9

OLAP in verteilten Data-Warehouse-Umgebungen

Beim Erfassen von Verkehrsdaten tritt eine sehr hohe Anzahl von Datensätzen auf. Dabei wird es unpraktikabel für die Analyse OnLine Transactional Processing Systeme (OLTP) zu verwenden, weil die Antwortzeiten zu hoch werden. Eine Lösung dieses Problems könnte der Einsatz von Data-Warehouse-Umgebungen mit OnLine Analytical Processing Systemen (OLAP) sein. In solchen OLAP-Systemen werden die Daten strukturiert und vorberechnet, so dass die Antwortzeiten für die Analysen optimiert werden können. In dem folgenden Szenario im verteilten Verkehrsbereich bietet es sich an, die Data-Warehouse-Umgebungen zu verteilen. Dabei ist es interessant zu betrachten, welche Möglichkeiten es gibt, diese verteilten Data-Warehouse-Systeme zentral abzufragen. In dieser Ausarbeitung werden zwei Ansätze dafür analysiert und bewertet.

9.1 Einleitung

9.1.1 Einführung in das Szenario

In dem Szenario wird angenommen, dass Daten anhand Mautstationen an Autobahnen an ein Data Warehouse pro Bundesland gesendet werden können. Eindeutig identifiert werden die Daten über die ID der Mautstation und die ID des Mauterfassungsgeräts im LKW. Außerdem kann noch das Datum und die Uhrzeit festgestellt werden. Dabei kann man sich noch vorstellen, dass weitere Daten über den LKW, wie z.B. geladene Güter und Gewicht (kann event. über Waage festgestellt werden, über die der LKW fährt) erfasst werden. Dabei müssen natürlich Richtlinien über die Behandlung von vertraulichen Daten beachtet werden.

Die Daten werden pro Mautstation erfasst und an eine zentrale Datenbank pro Bundesland gesendet. Zusätzlich wird davon ausgegangen, dass an jeder Autobahnauffahrt und -abfahrt auch wieder eine Mautstation steht, die mitbekommt, wenn der LKW die Autobahn verlässt bzw. wenn er auf die Autobahn auffährt. Dadurch kann auch eine Routenverfolgung pro LKW durchgeführt werden, weil die Daten jeder Mautstation an einer zentraler Stelle pro Bundesland gesammelt werden. Es kann dabei festgestellt werden, wie ein bestimmter LKW eine Strecke zwischen Berlin und Karlsruhe gefahren ist, da auch festgehalten wird, wann und ob das Bundesland verlassen wird und ein anderes Bundesland die Daten weiter sammelt (durch die ID des Mauterfassungsgeräts des LWKs kann wieder die Beziehung hergestellt werden). Dabei können die Daten dann zentral von z.B. der Spedition abgefragt werden mit dem Zweck die Strecke zu optimieren. Außerdem kann über die Erfassung des Gewichtes und die Anzahl der LKWs pro Zeiteinheit die Auslastung eines bestimmten Streckenabschnitts einer Autobahn festgestellt werden. Eine weitere Anwendung wäre eine Erfassung der Güter, die über die Autobahnen transportiert werden, für das Statistische Bundesamt, das die Daten momentan noch manuell ermittelt.

Es sind eine Reihe von weiteren Anwendungen denkbar, für die Daten von LKWs gesammt werden können, in naher Zukunft vielleicht auch von PKWs. Wie und wann die Daten aggregiert werden, ist nicht Gegenstand dieser Ausarbeitung. Die gesammelten Daten kommen fertig aggregiert im Data Warehouse des entsprechenden Bundeslands an. Auch die Behandlung von inkonsistenten oder widersprüchlichen Daten ist nicht Gegenstand dieser Ausarbeitung. Strukturell sieht das Szenario wie in Abbildung 9.1 aus.

9.1.2 Data-Warehouse und OLAP

Es gibt keine offizielle Definition von Data-Warehouse, die häufigste zitierte Definition von Bill Inmon [Inm96] lautet:

A data warehouse is a subject-oriented, integrated, time-varying, non-volatile collection of data in support of the management's decision-making process.

Anhand der Definition kann man also vier Eigenschaften eines Data-Warehouses rausfinden, die hier kurz skizziert werden (siehe [BG00]).

- Fachorientierung

  Der Zweck der Datenbasis liegt nicht mehr auf der Erfüllung einer Aufgabe z.B. eine Personaldatenverwaltung, sondern auf der Modellierung eines spezifischen Anwendungsziels.

- Integrierte Datenbasis

Abbildung 9.1: Szenario

Die Datenverarbeitung findet auf integrierten Daten aus mehreren Datenbanken statt

- Historische Daten

  Die Verarbeitung der Daten ist so angelegt, dass vor allem Vergleiche über die Zeit stattfinden. Es ist dazu unumgänglich, Daten über einen längeren Zeitraum zu halten.

- Nicht flüchtige Datenbasis

  Die Datenbasis ist als stabil zu betrachten. Daten, die einmal in das Data-Warehouse eingebracht wurden, werden kaum entfernt oder geändert.

Auch wenn die Definition von Inmon sehr häufig zitiert wird, hier noch eine etwas andere Definition [Kur98]:

Ein Data Warehouse repräsentiert eine, von den operativen Datenbanken getrennte Decision Support-Datenbank (Analyse-Datenbank), die primär zur

	Transaktional ausgerichtete Anwendungssysteme	**Analytisch orientierte Data-Warehouse Systeme**
Anwendertyp	Sachbearbeiter	Manager, Controller, Analysten
Anwenderzahl	sehr viele	wenige
Interaktionstyp	Lesen, Einfügen, Ändern	Lesen, Hinzufügen
Interaktionsdauer	kurz	periodisch
Anfragestruktur	einfach strukturiert	komplex
Anzahl gleichzeitiger Zugriffe	sehr viele	wenige

Tabelle 9.1: Abgrenzung aus Sicht der Anwendung

Unterstützung des Entscheidungsprozesses im Unternehmen genutzt wird. Ein Data Warehouse wird immer multidimensional modelliert und dient zur langfristigen Speicherung von historischen, bereinigten, validierten, synthetisierten, operativen, internen und externen Datenbeständen.

Im Folgenden wird die Abgrenzung eines Data-Warehouses gegenüber traditionellen, also transaktionsorientierten, Informationssystemen behandelt.

Abgrenzung aus Sicht der Anwendung

Ein wichtiges Kriterium zur Abgrenzung von Data-Warehouse-Systemen zu transaktionalen Anwendungssystemen ist der Anwendungskreis der jeweiligen Systeme. Ein transaktionales System hat kurze Lese- und Schreibtransaktionen, dabei werden meist einzelne Datensätze gelesen oder modifiziert, d.h. eingefügt, gelöscht oder geändert. Bei einem Data-Warehouse-System überwiegt bei der Analyse meist der Lesebetrieb mit Bezug auf eine Vielzahl einzelner Datensätze, die durch Anwendung statistischer Verfahren zu charakteristischen Aussagen verdichtet werden. Auch bei der Anzahl der potentiellen und tatsächlichen Nutzern gibt es Unterschiede zwischen den beiden Systemen. Während transaktional arbeitenden Systemen Nutzer im "Tausenderbereich" haben, sind klassische Data-Warehouse-Systeme bedingt durch die komplexen und aufwändig auszuwertenden Anfragen nur auf einen deutlich kleineren Nutzerkreis zugeschnitten, so dass auch die Gesamtzahl der Anwender pro Installation deutlich geringer ausfällt als bei klassischen transaktionalen Systemen. Die Tabelle 9.1 führt die Unterschied aus der Perspektive der Anwendung noch einmal auf.

	Transaktional ausgerichtete Anwendungssysteme	**Analytisch orientierte Data-Warehouse-Systeme**
Datenquellen	zentraler Datenbestand	mehrere unabhängige Datenquellen
Schemaentwurf	anfrageneutrale Datenmodellierung	analysebezogene Datenmodellierung
Eigenschaften des Datenbestands	originär, zeitaktuell, autonom, dynamisch	abgeleitet \ konsolidiert, historisch, integriert, stabil, teilweise aggregiert
Speicherform	zweidimensional	mehrdimensional
Aktualisierungszyklus	Echtzeit	periodisch
Datenvolumen	Megabyte - Gigabyte	Gigabyte - Terabyte
Typische Antwortzeiten	ms - s	s - min

Tabelle 9.2: Abgrenzung aus der Sicht der Datenhaltung

Abgrenzung aus der Sicht der Datenhaltung

Auch aus der Sicht der Datenhaltung gibt es Unterschiede zwischen transaktional ausgerichteten Anwendungssystemen und einem Data-Warehouse-System. Daten eines Data-Warehouse-Systems werden aus mehreren, u.U. heterogenen, Datenquellen extrahiert, bereinigt und in einer konsolidierten Datenbasis integriert. Dem gegenüber steht bei einem transaktionalen Anwendungssystem eine logisch zentralisierte Datenbasis mit originären und zeitaktuellen Daten im Vordergrund. Die Speicherform bei einem Data-Warehouse-System ist mehrdimensional, d.h. die Daten werden über die Dimensionen (Metadaten), die Hierarchien besitzen können, beschrieben und in Zellen gespeichert. Der benötigte Speicherplatz und die Antwortzeiten eines Data-Warehouse-Systems sind im Vergleich zu transaktional ausgerichteten Anwendungssystemen sehr hoch. Der Vergeich wird in der Tabelle 9.2 zusammenfassend angezeigt.

OLAP, ROLAP, MOLAP und HOLAP

Der Begriff Online Analytical Processing (OLAP) wurde 1993 von E.F. Codd [CCS93] wie folgt definiert:

OLAP sind zahlreiche spekulative "was-wenn" und/oder "warum" Datenmodell-Szenarien, welche innerhalb einer spezifischen historischen Basis und Perspektive ausgeführt werden. Dynamische Unternehmensanalysen sind notwendig, um Information aus Unternehmensdatenmodellen zu kreieren, zu manipulieren, anzuregen und herzustellen. Das beinhaltet die Fähigkeit,

neue oder unvorhergesehene Verbindungen zwischen Variablen zu erkennen, die Fähigkeit, die nötigen Parameter zu identifizieren um große Mengen an Daten zu handhaben, um eine unendliche Anzahl an Dimensionen zu kreieren und um kreuzdimensionale Bedingungen und Ausdrücke zu spezifizieren.

Die Auswertung erfolgt also nicht mehr statisch sondern dynamisch, indem die Dimensionen miteinander kombiniert werden. OLAP wird zumeist als Baustein (analytisches Werkzeug bzw. System) in einem übergeordneten Data-Warehouse-Konzept gesehen. Der Begriff "Data Warehouse" beschreibt dabei ausschließlich die Analysedatenbank. Diese Betrachtungsweise ist nicht unumstritten. Die Begriffe ROLAP (relationales OLAP), MOLAP (multidimensionales OLAP) und HOLAP (hyprides OLAP) werden auf die Architektur des Data Warehouses bezogen, auf die OLAP verwendet werden soll, was aber nicht ganz konsequent ist, weil OLAP nichts mit der Architektur im engeren Sinne zu tun hat. Im Zusammenhang von OLAP-Datenbanken spricht man auch von Würfeln, weil die Datenbanken mehrdimensional aufgebaut sind. Für mehr Informationen zu OLAP siehe [New], [olab] und [olaa].

ROLAP Der relationale Ansatz ist am weitesten verbreitet. Dies begründet sich darin, dass relationale Datenbanktechnologien sich bewährt haben. SQL wird zur Datentransformation und für OLAP-Abfragen verwendet. Die Architektur sieht so aus, dass der zentrale ROLAP-Server einerseits auf das Data Warehouse und andererseits auf die Metadatendefinitionen zugreift. Durch die zugrunde liegende RDBMS-Technologie sind ROLAP-Systeme in der Lage, sehr große Datenbestände (bis in den Terabytebereich) zu verwalten. Oft benötigte OLAP-Abfragen werden zum Teil vorab berechnet, um ein konstantes Antwortzeitverhalten des OLAP-Servers zu garantieren. Ein ROLAP wird meist mit Hilfe eines Star-Schemas modelliert, siehe Abbildung 9.2.

Star-Schema Das Star-Schema ist ein Ansatz zur relationalen Modellierung multidimensionaler Daten. In diesem Ansatz werden zwei Arten von Tabellen implementiert: Einerseits die Faktentabelle, welche Daten enthält, die aus den verschiedenen Vorgängen und Abfragen im transaktionalen System entstehen und sich über die Zeit ändern, andererseits die Dimensionstabellen, welche über die Zeit relativ statisch sind und die grundlegenden Datensätze von verschiedenen Objekten im System enthalten. Die Faktentabelle enthält die quantifizierenden, numerischen Merkmale und eine Anzahl an Fremdschlüsseln, die einen eigenen Primärschlüssel aus der Zusammensetzung der Fremdschlüssel bilden. Die Dimensionstabellen enthalten die qualifizierenden, meist textuelle Merkmale, sie bilden also die Beschreibung der Datensätze aus der Faktentabelle und sind meist hierarchisch aufgebaut.

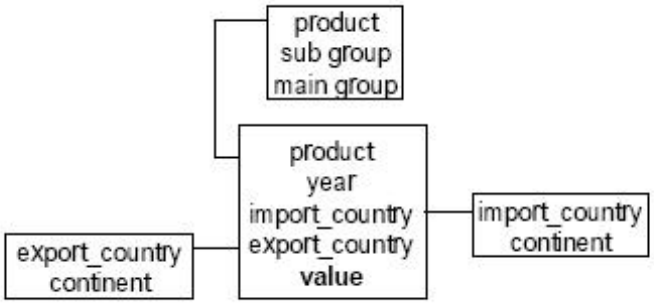


Abbildung 9.2: Beispiel für ein Star-Schema

Abbildung 9.2 zeigt ein Beispiel für ein Datenmodell nach dem Star-Schema.

MOLAP Multidimensionales OLAP verwendet eine (zumeist proprietäre) multidimensionale Datenbank zur Speicherung.

HOLAP Hybrides OLAP verwendet sowohl eine herkömmliche relationale Datenbank zur Speicherung der dünn besetzten historischen Data-Warehouse-Detailsdaten als auch eine multidimensionale Datenbank (MDDB) zur effizienten Speicherung der dicht besetzten Datenwürfel. Es werden beide Ansätze miteinander kombiniert um die jeweiligen Vorteile zu nutzen. Ein Teil der Daten wird in ROLAP gespeichert, ein anderer Teil in MOLAP.

9.1.3 Weiteres Vorgehen

Nachdem kurz erklärt wurde, was OLAP und Data Warehouse ist, wollen wir uns den eigentlichen Thema wieder zuwenden, nämlich der Implementierung eines verteilten Data-Warehouse-Systems im Verkehrssystem.

9.2 Skalla-Architektur-Ansatz

Der erste Ansatz, der für diesen Zweck betrachtet wird, benutzt ein ROLAP-System, das Skalla-Architektur genannt wird und von [ABJ+02] entwickelt wurde.

Eine Skalla-Architektur, wie in Abbildung 9.3, besteht aus mehreren verteilten Data-Warehouse-Systemen (*Skalla Sites*). Es gibt einen zentralen Koordinator (*Skalla Coordinator*), der alle OLAP-Anfragen annimmt, sie in die

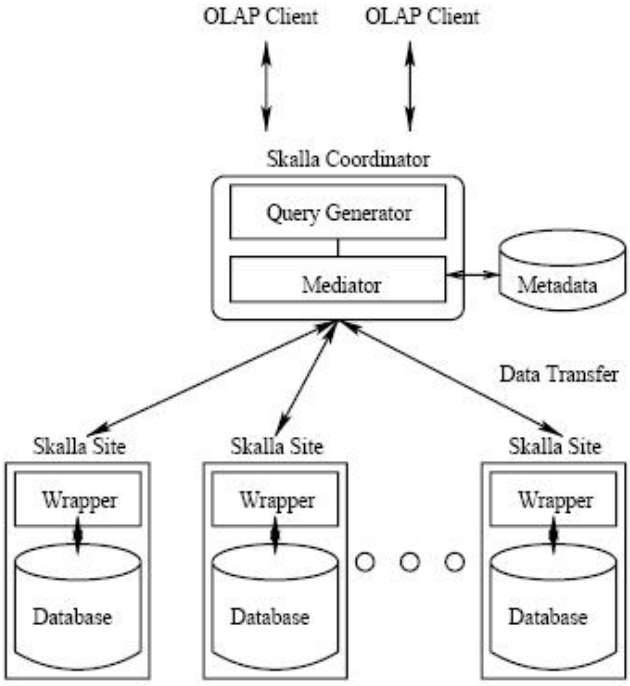


Abbildung 9.3: Skalla-Architektur

Abfragesprache GMDJ (Generalized MultiDimensional Join) umsetzt, an die verteilten Data-Warehouse-Systeme sendet, die Ergebnisse wieder einsammelt und an den anfragenden OLAP-Client zurück sendet. Der *Skalla Coordinator* besteht aus dem *Query Generator*, der die Aufgabe hat die Anfrage in GMDJ-Ausdrücke umzuwandeln und an den *Mediator* weiter zu schicken, der dann das Durchführen der Anfrage steuert. Der *Mediator* hat auch Zugriff auf die *Metadaten*. Die einzelnen *Skalla Sites* besitzen jeweils einen *Wrapper*, der dafür sorgt, dass die Daten in einem für den *Mediator* kompartiblen Format gesendet werden.

Dieses Modell passt hervorragend in unser Szenario, da wir auch von verteilten Data-Warehouse-Systemen ausgehen (in jedem Bundesland ein Data-Warehouse-System) und sie werden auch von einem Koordinator (zentraler Provider) abgefragt.

9.2.1 Beschreibung des Algorithmus

Wenn eine OLAP-Anfrage kommt, wird diese vom *Query Generator*, einem GMDJ Anfrageoptimierer, in GMDJ-Ausdrücken umgewandelt und an den *Mediator* weiter geschickt. Diese GMDJ-Ausdrücke sind jetzt optimiert

für verteiltes Rechnen. Als Erstes konstruiert der *Mediator* eine leere Ergebnisstruktur, in der die Ergebnisse später reingeschrieben werden sollen. Diese Ergebnisstruktur und die GMDJ-Anfrage wird an jede *Skalla Site* geschickt, dort lokal ausgeführt und dann wieder an den *Mediator* zurück gesendet. Dort werden die Ergebnisse synchronisiert, d.h. gleiche Zeilen zusammengefasst und aggregiert. Dieser Ablauf wird für jeden Operator in dem GMDJ-Ausdruck iterativ ausgeführt.

9.2.2 Definition eines GMDJ-Ausdrucks

Ein GMDJ-Befehl ist immer folgendermaßen aufgebaut: $MD(B_0, F_0, l_1, \theta_1)$. B_0 ist die Basisrelation (Basistabelle), F_0 die Detailrelation (abzufragende Tabelle), in einer Liste l_1 werden die Operationen angegeben und θ_1 bildet die Liste der Verknüpfungen und Bedingungen. Ein GMDJ-Ausdruck kann beliebig geschachtelt werden, dann wird erst der innere Operator ausgewertet und mit dem Ergebnis wird der äußere Operator ausgeführt usw.

9.2.3 Ablauf anhand eines Beispiels

Angenommen das verteilte Data-Warehouse hätte folgendes einfaches Schema:

```
Traffic(LorryID, LorryTyp, Date, Load, Weight)
```

und die einzelnen Skalla Sites wären mit Daten gefüllt, wie in den Tabellen 9.3 und 9.4 gezeigt.

LorryID	LorryTyp	Date	Load	Weight
4711	MAN	20040611	Oil	9
4712	MAN	20040611	Wood	11
4713	Daimler	20040611	Gas	8
4714	Daimler	20040611	Gas	10

Tabelle 9.3: Traffic auf Skalla Site 1

LorryID	LorryTyp	Date	Load	Weight
4812	MAN	20040611	Wood	9
4813	Daimler	20040611	Gas	10
4814	Daimler	20040611	Gas	8

Tabelle 9.4: Traffic auf Skalla Site 2

Wir wollen nun abfragen, wieviele Tonnen von welcher Ladung durch zwei Bundesländer (repräsentiert durch die Skalla Sites 1 und 2) von welchem

LKW-Typ gefahren wurden. Die Abfrage erfolgt mit dem Befehl

$$MD(MD(B_0, F_0, l_1, \theta_1) \rightarrow B_1, F_1, l_2, \theta_2)$$

Dabei ist B_0 $\pi_{LorryTyp,Load}(Traffic)$, F_0 ist die Traffic-Tabelle, l_1 ist ($cnt(*)$ $\rightarrow$ $cnt1$, $sum(Weight)$ $\rightarrow$ $sum1$), θ_1 ist ($F_0.LorryTyp = B_0.LorryTyp$ & $F_0.Load = B_0.Load$ & $F_0.Date = B_0.Date$), F_1 ist die Traffic-Tabelle, l_2 ist ($cnt(*) \rightarrow cnt2$) und θ_2 ist ($F_1.LorryTyp = B_1.LorryTyp$ & $F_1.Load = B_1.Load$ & $F_1.Date = B_1.Date$ & $F_1.Weight \geq sum1/cnt1$).

Aufgrund von diesem Befehl wird als Erstes eine Ergebnisstruktur angelegt: $X = \{LorryTyp, Load, cnt1, sum1, cnt2\}$. Dieses intiale Schema (X) und der Anfrageplan werden an die lokalen Seiten gesendet. Die Basistabellen (Gruppierungen der Anfrage) werden dann bei den Skalla Sites durch $\pi_{LorryTyp,Load}(Traffic)$ berechnet. Das Resultat für die Relation X_0 für die beiden Sites ist in den Tabellen 9.5 und 9.6 beschrieben.

LorryTyp	Load
MAN	Oil
MAN	Wood
Daimler	Gas

Tabelle 9.5: Ergebnisrelation X_0^{s1} der Skalla Site 1 im ersten Durchlauf

LorryTyp	Load
MAN	Wood
Daimler	Gas

Tabelle 9.6: Ergebnisrelation X_0^{s2} der Skalla Site 2 im ersten Durchlauf

Jede Skalla Site sendet seine Tabelle zum Mediator, der sie synchronisiert, um die Relation X_0 zu bekommen, die die Basiswerte (Gruppen) der Abfrage beinhaltet (Tabelle 9.7).

LorryTyp	Load
MAN	Oil
MAN	Wood
Daimler	Gas

Tabelle 9.7: Synchronisierte Ergebnisrelation X_0 des Mediators im ersten Durchlauf

Diese Relation wird dann wieder an die Skalla Sites gesendet. Die Skalla Sites benutzen diese Relation als Basis für ihre nächste Runde der Berechnung, bei der der erste (innere) GMDJ-Ausdruck verarbeitet wird. Der innere Ausdruck ist $MD(X_0, F_0, l_1, \theta_1)$. Dabei ist F_0 ist die Traffic-Tabelle,

l_1 ist $(cnt(*) \rightarrow cnt1, sum(Weight) \rightarrow sum1)$ und θ_1 ist $(F_0.LorryTyp = X_0.LorryTyp \ \& \ F_0.Load = X_0.Load \ \& \ F_0.Date = X_0.Date)$.

Das Ergebnis, das die beiden Skalla-Sites jeweils zurück liefern, steht in den zwei Tabellen 9.8 und 9.9.

LorryTyp	Load	cnt1	sum1
MAN	Oil	1	9
MAN	Wood	1	11
Daimler	Gas	2	18

Tabelle 9.8: Ergebnisrelation X_1^{s1} der Skalla Site 1 im zweiten Durchlauf

LorryTyp	Load	cnt1	sum1
MAN	Oil	0	0
MAN	Wood	1	9
Daimler	Gas	2	18

Tabelle 9.9: Ergebnisrelation X_1^{s2} der Skalla Site 2 im zweiten Durchlauf

Diese Zwischenergebnisse werden wieder zurück gesendet und vom Mediator synchronisiert (Tabelle 9.10).

LorryTyp	Load	cnt1	sum1
MAN	Oil	1	9
MAN	Wood	2	20
Daimler	Gas	4	36

Tabelle 9.10: Synchronisierte Ergebnisrelation X_1 des Mediators im zweiten Durchlauf

Dieses synchronisierte Ergebnis wird an die lokalen Seiten zurück geschickt und $cnt2$ wird auf jeder lokalen Seite berechnet, indem der Durchschnitt $(sum1/cnt1)$ der ersten Runde berechnet wird.

Der äußere Ausdruck ist $MD(X_1, F_1, l_2, \theta_2)$. Dabei ist F_1 ist die Traffic-Tabelle, l_2 ist $(cnt(*) \rightarrow cnt2)$ und θ_2 ist $(F_1.LorryTyp = X_1.LorryTyp \ \& \ F_1.Load = X_1.Load \ \& \ F_1.Date = X_1.Date \ \& \ F_1.Weight \geq sum1/cnt1)$.

Das Ergebnis, das die beiden Skalla-Sites jeweils zurück liefern, steht in den zwei Tabellen 9.11 und 9.12.

Schließlich werden die Zwischenergebnisse zum Mediator zurück gesendet und synchronisiert, um das endgültige Ergebnis zu berechnen (Tabelle 9.13).

Nur die Ergebnistabellen und Anfragen werden zwischen dem Mediator und den Sites gesendet. Das ist ein wichtiges Feature der Skalla-Verarbeitung, das

LorryTyp	Load	cnt1	sum1	cnt2
MAN	Oil	1	9	1
MAN	Wood	2	20	1
Daimler	Gas	4	36	1

Tabelle 9.11: Ergebnisrelation X_2^{s1} der Skalla Site 1 im dritten Durchlauf

LorryTyp	Load	cnt1	sum1	cnt2
MAN	Oil	1	9	0
MAN	Wood	2	20	0
Daimler	Gas	4	36	1

Tabelle 9.12: Ergebnisrelation X_2^{s2} der Skalla Site 2 im dritten Durchlauf

die effektive Minimierung der zu transferierenden Daten erlaubt. Ein GMDJ-Ausdruck mit m GMDJ-Operatoren braucht m+1 Runden zur Verarbeitung.

9.2.4 Optimierungen

Es gibt verschiedene Möglichkeiten diesen Ansatz zu optimieren.

1. Reduktion der Attribute

   Die Reduktion von Attributen kann die Datentransferkosten signifikant senken und damit die Gesamtkosten einer Abfrage verbessern.

   Es geht dabei darum, dass synchronisierte Attribute der Basisstruktur, die nicht länger für die verteilte Berechnung notwendig sind, herausgelassen werden, d.h. es werden weniger Daten übertragen.

2. Gruppenreduktion aufgrund der Verteilung

   Weil wir wissen, wo welche Daten stehen, muss die Ergebnisstruktur nicht überall hingeschickt werden. Wenn man z.B. die Bundesländer Rheinland-Pfalz und Baden-Württemberg abfragen will, muss man die Ergebnisstruktur nicht nach Hamburg schicken, sie kommt leer zurück und erzeugt nur unnötig Datenverkehr.

3. Verteilungsunabhängige Gruppenreduktion

   Wenn nach der ersten (oder einer weiteren) Runde von einer Seite leere Attributwerte für eine Gruppe (Zeile) zurück an den Mediator gesendet werden, schickt der Mediator nicht mehr die gesamte synchronisierte Ergebnisstruktur an jede lokale Seite, sondern nur noch die Gruppen (Zeilen), für die vorher auch ein Wert an den Mediator geliefert wurde. Dadurch wird wieder Netzwerkverkehr gespart.

LorryTyp	Load	cnt1	sum1	cnt2
MAN	Oil	1	9	1
MAN	Wood	2	20	1
Daimler	Gas	4	36	2

Tabelle 9.13: Synchronisierte Ergebnisrelation X_2 des Mediators im dritten Durchlauf

4. Reduktion der Synchronisationen

   Dabei wird durch Umschreiben der GMDJ-Ausdrücke versucht, die Anzahl der Runden zu reduzieren oder Synchronisationen einzusparen Eine Synchronisation fällt beispielsweise dann weg, wenn nur eine Seite ein Ergebnis zurückliefert, was eine Synchronisation unnötig macht.

9.3 XML-Daten-Ansatz

Der zweite untersuchte Ansatz beschäftigt sich mit XML-Daten, die in einem relationalen OLAP-System vorliegen und wurde von [NNNT02] entwickelt. XML-Daten haben den Vorteil, dass auch heterogene Daten verarbeitet werden können, sie müssen vorher nur aus dem bestehenden Datenformat nach XML transformiert werden und können dann von der zu untersuchenden Architektur behandelt werden. Eine Datenänderung kann nur von dem datensammelnden Teil des Systems durchgeführt werden, der abfragende Benutzer hat keinen Einfluss auf die Daten.

Es wird weiterhin davon ausgegangen, dass es billiger ist, für jede Anfrage einen neuen Ergebniswürfel zu erstellen und die Daten erst auf Verlangen von den verteilten ROLAP-Systemen zu holen, weil kleinere Würfel, die ja nur das beinhalten, was der Benutzer (derjenige, der angefragt hat) auch wirklich sehen will, schneller die gewünschten Daten liefern können. Die Abfragesprache ist MDX (MultiDimensional Expressions) und ist ein Quasi-Standard von Microsoft. Nähere Beschreibung zur Sprache gibt es unter [mdx]. Die Architektur des zu untersuchenden Ansatzes beschreibt die Abbildung 9.4.

9.3.1 Beschreibung des Algorithmus

Eine OLAP-Anfrage wird folgendermaßen innerhalb der Architektur behandelt:

1. Das virtuelle "universelle" *Data-Warehouse-Schema* repräsentiert alle möglichen Analysedaten, die dem Benutzer gezeigt werden. Der Benutzer sendet eine Anfrage, indem er die MDX Abfragesprache benutzt.

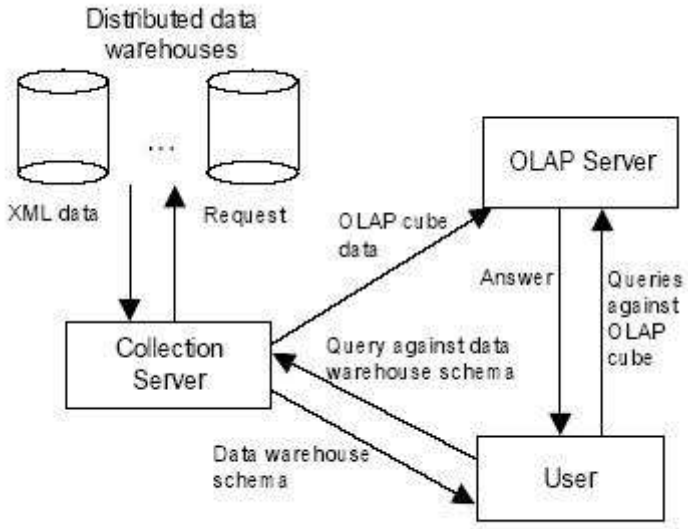


Abbildung 9.4: Systemarchitektur des XML-Ansatzes

2. Der *Collection Server* analysiert die Anfrage, um Kennzahlen, Dimensionen und Einschränkungen für den neuen Würfel herauszufinden

3. Der *Collection Server* sendet die Anfrage zu den verteilten *Data Warehouses*

4. Die *Data Warehouses* senden die verlangten Daten im XML-Format zurück und der *Collection Server* sammelt die Daten

5. Der *Collection Server* führt die verlangten Modifikationen an den Daten und aggregiert sie, wenn möglich und notwendig.

6. Der *Collection Server* sendet die Daten zum *OLAP-Server* um einen echten OLAP-Würfel zu konstruieren

7. Der Benutzer kann jetzt Anfragen an den OLAP-Würfel auf dem *OLAP-Server* mit den Tools stellen, die der *OLAP-Server* zur Verfügung stellt.

9.3.2 Ablauf anhand eines Beispiels

Die einzelnen Data-Warehouses liegen im relationalen OLAP vor, das Datenformat ist XML. Die Abbildung 9.5 zeigt einen Teil einer XML-Definition für ein OLAP-Schema. Das "*"-Symbol im Schema repräsentiert eine willkürliche Zeichenkette, die als Attributwert auftauchen kann. Das XML-Schema

```
<olap_cube name="*">
  <fact_table>
    <row Load          ="*"
         LorryTyp      ="*"
         Weight        ="*"
         Date          ="*"/>
  </fact_table>
  <Load>
    <row Load="*" sub_group="*"
         main_group="*"/>
  </Load>
  ..
</olap_cube>
```

Abbildung 9.5: Beispiel für Data-Warehouse-Schema in XML

enthält eine Faktentabelle, in der die Werte der Kennzahlen, die Kennzahlen und die Dimensionsschlüssel stehen und enthält noch verschiedene Dimensionstabellen, die die Dimensionshierarchien beinhalten. Ein Data-Warehouse-Schema der verteilten Data-Warehouse-Systemen könnte daher so aussehen, wie in Abbildung 9.5 gezeigt.

Die *fact_table* ist die Faktentabelle, in der der Wert der Kennzahl *Weight* und *Date* steht. Ausserdem stehen darin noch die Dimensionsschlüssel *LorryTyp* und *Load*, die auf die entsprechenden Dimensionstabellen referenzieren. In den hierarchisch aufgebauten Dimensionstabellen stehen dann weitere Informationen zu den Dimensionen, die ausgewertet werden können. Ein Beispiel für ein gefülltes Data-Warehouse-Schema wird in Abbildung 9.6 gezeigt. Dieser OLAP-Würfel kann jetzt, wie in Abbildung 9.7 gezeigt, per MDX abgefragt werden.

Diese einfache Abfrage analysiert das Gewicht in der *collection*-Datenbank für die Hauptproduktgruppe 0 ab, die entweder am 11.06.2004 oder 12.06.2004 von den LKW-Typen 2 oder 5 transportiert wurden.

So eine Abfrage liefert genügend Informationen, um ein neues OLAP-Würfelschema zu erstellen und zu füllen. In unserem Beispiel bekommen wir *LorryTyp*, *main_group* der Ladung und *Date* als Dimensionen und wir beschränken unsere Betrachtung auf die LKW-Typen 2 und 5, *main_group* der Ladung 0 und *Datum* 11.06.2004 und 12.06.2004. Diese Daten werden von jedem einzelnen Data Warehouse an den *Collection Server* geschickt, dieser nimmt die Daten an, aggregiert sie und schickt sie als OLAP-Schema weiter an den OLAP-Server. Dort wird der Würfel erstellt und der Benutzer kann diesen Würfel abfragen.

Die Verteilung der Datenbanken kann sowohl horizontal als auch vertikal er-

```
<olap_cube name="collection">
  <fact_table>
    <row Load ="Oil"  LorryTyp ="MAN"
         Weight ="8"  Date ="20040611" >
    <row Load ="Gas"  LorryTyp ="Daimler"
         Weight ="9"  Date ="20040611"/>
  </fact_table>
  <Load>
    <row Load="Oil"   sub_group="consistent"
         main_group="liquid"/>
    <row Load="Gas"   sub_group="fluid"
         main_group="liquid" />
  </Load>
  ..
</olap_cube>
```

Abbildung 9.6: Beispiel für ein gefülltes OLAP-Schema

```
SELECT {LorryTyp.company.[2],
        LorryTyp.company.[5]},
       Load.main_group.0.CHILDREN,
       {Date.[20040611], Date.[20040612]}
FROM collection
WHERE Weight
```

Abbildung 9.7: Beispiel für eine MDX-Abfrage

folgen. Horizontal bedeutet beispielsweise, dass die Faktentabelle auf einem Server, die Dimensionstabellen auf mehreren anderen Servern liegen. In einer zentralen XML-Datenbank liegen dann die Informationen vor, welche Datenbank auf welchem Server zu finden ist. Diese Art der Verteilung ist aber für unser Szenario nicht geeignet. Wir benötigen die vertikale Verteilung, das bedeutet, dass die ROLAP-Datenbanken vollständig auf verschiedenen Servern liegen (für jedes Bundesland ein eigenes ROLAP), die aber von der Struktur überall gleich aussehen, nur die gespeicherten Daten sind andere.

9.4 Zusammenfassung

Beide Ansätze sind im Prinzip ziemlich gleich, ein zentraler Mediator schickt die Anfrage an die verteilten Data-Warehouse-Umgebungen, sammelt die Ergebnisse und verarbeitet sie weiter. Aber es können trotzdem einige Unterschiede in der Tabelle 9.14 aufzulist werden.

Der größte Unterschied liegt bei der Abfragesprache. Während bei dem

Eigenschaft	**Skalla-Architektur**	**XML-Architektur**
Datenformat	jede Skalla-Seite hat einen Wrapper	XML
Dauer der Datensammlung	mehrere Runden	sofort alles
Abfragesprache	GMDJ	MDX
Ergebnis	Ergebnisstruktur	OLAP-Würfelschema
Optimierungsmöglichkeiten	mehrere (siehe Kapitel 9.2.4)	unbekannt

Tabelle 9.14: Vergleich von Skalla und XML

XML-Ansatz die Abfragesprache MDX verwendet wird, die aber eigentlich eine Abfragesprache für ein lokales Data Warehouse ist, benutzt der Skalla-Ansatz die Abfragesprache GMDJ, die speziell für verteilte Data-Warehouse-Systeme entwickelt wurde. Man kann beim GMDJ nicht genau abschätzen, wieviel Zeit die vielen Runden gegenüber dem XML-Ansatz kosten, der alle Daten auf einmal holt. Aber dafür existieren ja die aufgeführten Optimierungen. Beide Ansätze behaupten von sich, dass sie schnell auf Anfragen reagieren können (auch bei großen Datenmengen), aber ein direkter Geschwindigkeitsvergleich zwischen diesen beiden Ansätzen hat es noch nicht gegeben, außerdem konnte ich auch keine praktische Implementierung finden, anscheinend existieren nur die beiden theoretischen Papiere. Es wird bei dem XML-Daten-Ansatz auch keine Aussage darüber gemacht, wie lange der generierte OLAP-Würfel auf dem OLAP-Server gehalten wird. Das wirft nämlich zwei Probleme auf: zum Einen gibt das ein Speicherplatzproblem bei vielen gleichzeitigen Anfragen und zum Anderen ist der Inhalt des OLAP-Würfels irgendwann nicht mehr aktuell und der Würfel muss gelöscht werden. Über diesen Punkt wird aber auch keine Aussage getroffen. Bei beiden Ansätzen wird auch keine Aussage über Performanz und Skalierbarkeit gemacht, so dass eine endgültige Bewertung schwierig ist. Insgesamt scheint mir die Skalla-Architektur für unser Szenario, verteilte Data-Warehouse-Systeme im Verkehrsbereich, besonders aufgrund der Abfragesprache GMDJ, besser geeignet zu sein.

Literaturverzeichnis

[ABJ+02] Michael O. Akinde, Michael H. Böhlen, Theodore Johnson, Laks V.S. Lakshmanan, and Divesh Srivastava, editors. *Efficient OLAP query processing in distributed data warehouses*, 2002.

[BG00] Andreas Bauer and Holger Günzel. *Data-Warehouse-Systeme*. dpunkt.Verlag, Germany, 2000.

[CCS93] E.F. Codd, S.B. Codd, and C.T. Salley. Providing olap to user-analysts: An it mandate. Technical report, E. F. Codd and Associates, 1993. <http://www.arborsoft.com/essbase/ppr_pspdf.html>.

[Cla98] Nils Clausen. *OLAP - Multidimensionale Datenbanken*. Addison-Wesley, Germany, 1998.

[Inm96] W.H. Inmon. *Building the Data Warehouse*, volume Second Edition. John Wiley & Sons, New York, USA, 1996.

[Kur98] Andreas Kurz. Data warehousing im internet / intranet: Prototypische implementierung eines web-basierten executive information system für entscheidungsträger. Technical report, Institute of Software Technology, Technical University of Vienna, Vienna, Austria, 1998.

[Kur99] Andreas Kurz. *Data Warehousing - Enabling Technology*. MITP-Verlag GmbH, Bonn, Germany, 1999.

[mdx] http://www.microsoft.com. Informationen zu MDX.

[New] news://comp.databases.olap. Newsgroup für OALP.

[NNNT02] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, and Peter Thanisch, editors. *Constructing an OLAP Cube from Distributed XML Data*, 2002.

[olaa] http://www.olapcouncil.org. Das OLAP-Council.

[olab] http://www.olapreport.com. Der OLAP Report.

Christian Hennings

10

Integration verteilter Datenquellen in GIS-Datenbanken

10.1 Einleitung

Geographische Daten wurden über Jahre hinweg gesammelt und in verschiedenster Form archiviert. Die Datensammlungen unterscheiden sich dabei in mehrfacher Weise. Sie umfassen zum einen verschiedene Inhalte. So gehören topologische Karten und Straßenpläne genauso dazu wie Klimadaten oder eine Datenbank mit den touristischen Sehenswürdigkeiten einer Stadt. Ihre einzige Gemeinsamkeit ist die Zugehörigkeit zu einer bestimmten räumlichen Koordinate. Aus dieser inhaltlichen Vielfalt ergibt sich auch eine technische. Die Datenbestände sind historisch oft unabhängig von einander gewachsen und dementsprechend nicht in einem einheitlichen Format gespeichert. Sie können sich sowohl noch auf Papier als auch schon in elektronischer Form befinden. Aber selbst, wenn sie elektronisch gespeichert sind, so kann es große Unterschiede geben. So können die Daten zum Beispiel in einer simplen Sammlung von Textdateien oder auch in Relationalen oder XML-Datenbanken gespeichert sein. Entsprechend existieren dann für diese verschiedenartigen Quellen auch verschiedene Anfragesprachen. Hinzu kommt, dass selbst bei der Speicherung der Daten in technisch identischen Quellen wie SQL-Datenbanken, noch immer verschiedene Schemata verwendet werden können.

Durch die rasante Entwicklung des Internets sind nun immer mehr und umfangreichere solcher GIS-Quellen für Nutzer zugänglich geworden. Daraus ergibt sich ein neues Problem. Alle Datenquellen einzeln nach bestimmten Informationen zu befragen, kann aufgrund der Heterogenität sehr mühsam sein. Auch Informationen, die sich erst aus dem Zusammenhang mehrerer Quellen ergeben, sind so schwer zu gewinnen.

Dieser Aufsatz beschäftigt sich mit einem Geographic Information System (GIS), das sich mit Hilfe eines Mediatorenkonzeptes der Lösung dieses Problems annimmt. Es ist ein System, das mittels Wrappern auf die Datenbestände zugreift und sie so integriert, dass sie nach außen hin einem einheitlichen Schema entsprechen und mit einer einzigen Anfragesprache abrufbar sind. Dabei berücksichtigt dieses GIS die unterschiedliche Mächtigkeit verschiedener Quellen. Features, die von einer Quelle angeboten werden, werden wenn möglich immer genutzt, aber falls sie von einer Quelle nicht angeboten werden, so können sie von GIS erbracht werden. Auch ist es möglich Operatoren zu implementieren, deren Ausführung erst durch das Wissen um Zusammenhänge verschiedener Quellen ermöglicht wird. GIS verfolgt dabei die Umsetzung des Mediatorenkonzeptes. Im Gegensatz zu Data Warehouses importiert und speichert es dauerhaft keine Daten, sondern beläßt diese in ihrem Ursprung. Es beschränkt sich in diesem Punkt auf die Integration der einzelnen Ergebnisse.

10.2 Allgemeines zum Mediatorkonzept von GIS

Den Kern bildet eine 3-Schichten-Client-Server-Architektur aus Datenquellen, Wrappern und dem Mediator. Die Wrapper regeln dabei die Kommunikation zwischen den Datenquellen und dem Mediator. Der Mediator nimmt die Anfragen entgegen, die von außen an das System gestellt werden, und kümmert sich um die Bearbeitung. Schließlich gibt er das Endergebnis zurück. Zusätzlich zu dieser Architektur werden noch 2 Ergänzungen benötigt. Zum einen sind dies die Inter-Shema Correspondence Assertions (ICA). Sie beinhalten Regeln über die Beziehungen zwischen den verschiedenen verwendeten Datenschemata. Zum anderen sind es die Operators. Sie umfassen Implementierungen von Operatoren, die das System anbietet oder benötigt, ohne dass sie von den Datenquellen angeboten werden.

10.2.1 Die 3-Schichten-Client-Server-Architektur

Die Basis der Architektur bilden die Datenquellen. Sie liefern den Inhalt, um dessen Verwendung es geht. Gegebenenfalls können sie auf diesen Daten auch noch bestimmte Operationen ausführen.

Jede dieser Quellen ist genau einem Wrapper zugeordnet. Sie sind ein entscheidender Faktor in der Kommunikation zwischen den Quellen und dem Mediator. Ihre Aufgabe besteht darin die Anfragen, die der Mediator versendet, in die Sprache zu übersetzen, die die Informationsquelle versteht. Anschließend müssen dann ihre Ausgaben entsprechend den externen Standards des Mediators transformiert werden(vgl. [KR99]). Der Wrapper ist bei GIS als Web Feature Server (WFS) umgesetzt, die mit dem Mediator

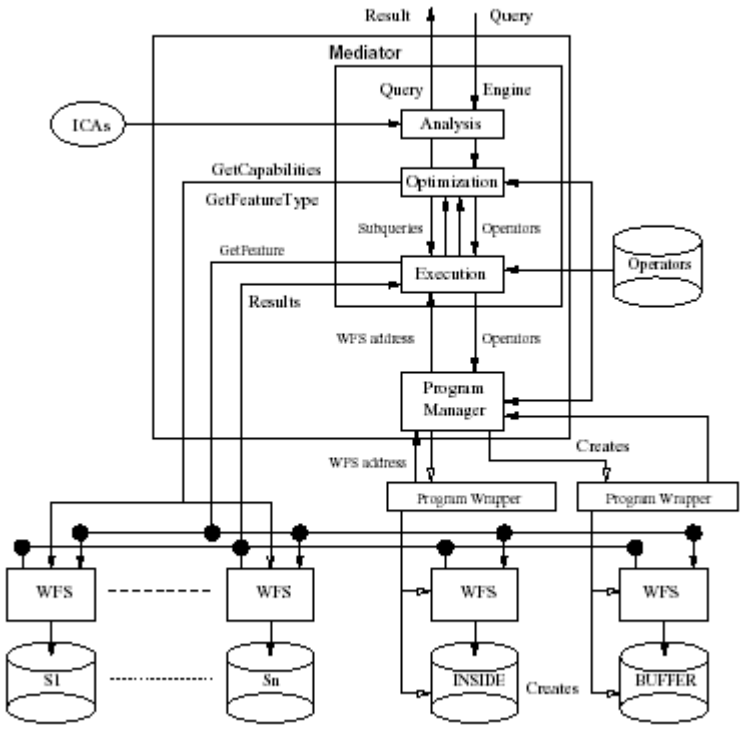


Abbildung 10.1: GIS Mediator Architektur [BEL02]

über HTTP kommunizieren. Ein WFS hat neben der Transformation von Anfragesprache und Ergebnissen noch weitere Aufgaben. Auf Anfrage des Mediators muss er mitteilen, welche Features er anbietet und welche Operatoren er darauf unterstützt. Zusätzlich muss er auch über die Struktur Auskunft geben können, wie diese Features genutzt werden.

Der Mediator selbst besteht aus drei Komponenten: einem Analyzer-, einem Optimizer- und einem Execution-Modul. Er führt zunächst eine Analyse der Anfragen durch, um unter anderem erste Konflikte zu beseitigen und die Anfrage zu optimieren. Anschließend splittet er sie in mehrere Subabfragen auf, die mittels WFS an die Datenquellen weitergeleitet werden. Nach Eingang aller Ergebnisse obliegt ihm noch die Aufgabe die Ergebnisse der Subabfragen zu einem Gesamtergebnis zusammenzusetzen.

10.2.2 Notwendige Ergänzungen

Da der Mediator verschiedenartige Quellen integrieren soll, muss er in der Lage sein, Konflikte zu lösen, die sich aus den Beziehungen zwischen verschiedenen verwendeten Datenschemata ergeben. So liegt zum Beispiel ein Aggregationskonflikt vor, wenn eine Quelle nur zwischen Straßen und Gebäuden unterscheidet, eine andere aber zusätzlich zwischen verschiedenen Gebäudearten wie Kirche oder Wohnhaus differenziert. Wenn in einer Anfrage nach Gebäuden gesucht wird, muss der Mediator wissen, dass die Kirchen der einen Quelle auch hierzu zu zählen sind. Das dafür notwendige Wissen wird in Form von Regeln bereitgestellt, den Inter-Shema Correspondence Assertions (ICA). Diese Regeln werden vom Mediator sowohl bei der Analyse und dem Aufsplitten einer Anfrage gebraucht als auch beim Zusammensetzen der Ergebnisse. Da diese Regeln ein formal definiertes System über Schemata verschiedener geographischer Informationsquellen darstellen und Relationen zwischen ihnen definieren, verfolgt GIS hier einen ontologiebasierten Ansatz.

Eine zweite Ergänzung betrifft das Ziel, auch solche Features dem User zugänglich zu machen, die von einzelnen oder gar allen Quellen gar nicht geliefert werden. Eventuell ergibt sich die Möglichkeit ein Feature anzubieten auch erst aus der Kombination mehrerer Quellen. So könnte zum Beispiel das Aufstellen einer Relation in den ICA's die Bereitstellung eines zusätzlichen Features erfordern. Diese Features müssen von Hand implementiert werden und werden gesammelt in den „Operators“ zur Verfügung gestellt.

10.3 Ablauf einer Anfrage an das GIS

Nach der Beschreibung der einzelnen Komponenten betrachte ich nun, was passiert, wenn von außen eine Anfrage an das GIS gestellt wird.

Zuerst wird sie vom Mediator analysiert. Dies geschieht im Analyzer-Modul. Es führt einige Optimierungen durch und löst mit Hilfe der ICA-Regeln bereits einige Konflikte. Ein gutes Beispiel wäre das bereits beschriebene Szenario, dass schließlich auch eine Kirche ein Gebäude ist und dies eventuell berücksichtigt werden muss.

Anschließend erstellt das Optimizer-Modul des Mediator einen Ablaufplan für die Anfrage. Hierfür erfragt es zunächst mittels WFS, welche Quelle welche Features unterstützt. Anschließend wird die Anfrage entsprechend dieser Informationen derart in Unterabfragen aufgeteilt, dass insgesamt die Fähigkeiten der Quellen möglichst gut ausgenutzt werden. Diese Unterabfragen wiederum werden in zwei Kategorien eingeteilt. Die eine umfasst jene,

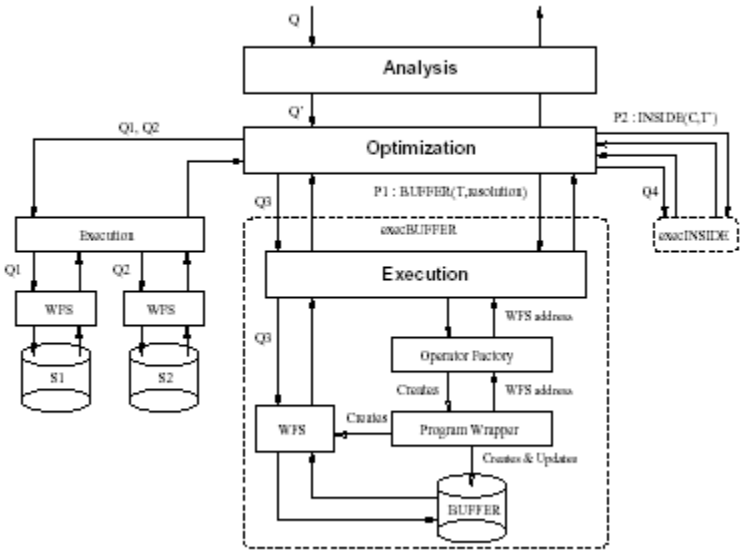


Abbildung 10.2: Ausführung einer Abfrage [BEL02]

die von einer Quelle allein und vollständig beantwortet werden können, die andere diejenigen, für deren Bearbeitung ein oder mehrere Operatoren aus dem zur Verfügung stehendem Pool benötigt werden.

Die Anfragen, die keinen weiteren Operator benötigen, werden dann vom Execution-Modul direkt an das entsprechende WFS weitergeleitet. Etwas mehr Aufwand muss für die zweite Gruppe betrieben werden. Wird ein Operator aus dem Pool benötigt, so sendet das Execution-Modul eine Anforderung und der „Program Manager“ führt die Implementierung für den jeweiligen Operator aus. Dann wird eine virtuelle Datenquelle erstellt, die die Ergebnisse der Ausführung enthält. Ebenso wird ein WFS bereitgestellt, über das auf die Daten zugegriffen werden kann.

Im letzten Schritt müssen dann nur noch die einzelnen Ergebnisse vom Optimization-Modul zu einem Gesamtergebnis zusammengefügt werden.

10.4 Berechnung des einheitlichen Endergebnisses

Bei der Berechnung des Gesamtergebnisses ist es von entscheidender Bedeutung, semantisch ähnliche Objekte zu filtern. Wenn zum Beispiel in den Unterergebnissen eine Kirche und ein Gebäude gefunden wurden, die beide an der exakt gleichen Position stehen, so kann man unter Berücksichtigung,

dass „Kirche“ eine Unterart von „Gebäude“ ist, von einer Korrespondenz ausgehen. Offensichtlich handelt es sich um ein und das selbe Objekt. Es ist dann natürlich nicht wünschenswert, die zweifache Ausführung in das Endergebnis zu übernehmen. Auch auf anderem Wege können sich semantische Ähnlichkeiten ergeben. So ist es durchaus wahrscheinlich, dass mit „Street“ und „Strasse“ das gleiche gemeint ist. Wenn sich ein Rinnstein und eine Straße in der Position überschneiden, so liegen nicht zwei verschiedene Objekte vor, sondern ein Objekt ist ein Teil des anderen.

10.4.1 Ontologiebasierter Ansatz

Zur Lösung des eben beschriebenen Problems verfolgt GIS einen ontologiebasierten Ansatz, der sich in den Regeln der ICA widerspiegelt. Die Idee begründet sich auf der Tatsache, dass unterschiedliche Datenquellen ähnliche Objekte der realen Welt oft in einer verschiedenen Semantik verwenden. Eine Ontologie sollte also aus einer Domänenontologie, einer Applikationsontologie für jede Quelle sowie Abstraktionsregeln darüber bestehen. Die Domänenontologie enthält Definitionen für Konzepte, die Objekten aus der Realen Welt entsprechen. Dies können zum Beispiel Straßen, Gebäude oder Wiesen sein. Eine Applikationsontologie beinhaltet hingegen, wie ein Konzept in einer Datenbasis umgesetzt wurde. Die Abstraktionsregeln schließlich definieren die Beziehungen zwischen Domänen- und Applikationsontologie.

10.4.2 Umsetzung in Prolog

Bei der Umsetzung dieses Ansatzes bedient sich GIS der Programmiersprache Prolog. Zunächst müssen sowohl die Objekte der Domäne als auch die der Applikationen taxonomiert und damit in eine hierarchische Beziehung gesetzt werden.

```
taxon[Subclass, class].
```

Da wie bereits erwähnt eine Kirche eine Unterart von Gebäuden ist, gilt:

```
taxon[Kirche, Gebaeude].
```

Dies muss für alle Objekte geschehen. Somit ist eine generelle Subklassenbeziehung durch folgende Prologzeilen rekursiv überprüfbar:

```
subClass[x,x].
subClass[x,z]:- (taxon[x,y] && subClass[y,z]).
```

Die erste Zeile stoppt lediglich die Rekursion, die zweite ermöglicht Subklassenbeziehungen beliebiger Tiefe.

Aufbauend auf diesen Taxonomien, werden nun Beziehungen zwischen Klas-

sen einzelner Ontologien hergestellt. Es gibt genau zwei mögliche Arten von Beziehungen zwischen einer Applikationsklasse und der Domänenklasse. Die erste ist eine Äquivalenzbeziehung und die zweite eine Aggregatbeziehung. Das bedeutet, dass mehrere Klassen der Domäne in einer Klasse von der Applikation zusammengefasst werden. Dies muss auch genau wie bei den Taxonomien für alle Klassen einmal angegeben werden.

```
refersToEquivalentClass [DomainClass, ApplClass].
refersToAggregateClass [DomainClass, ApplClass].
```

Mit Hilfe dieser beiden Beziehungen zwischen einer Domain- und einer Applikationsklasse, kann man nun eine Aussage über das Verhältnis von zwei Klassen aus verschiedenen Applikationen treffen. So sind zwei Applikationsklassen semantisch äquivalent, wenn sie beide zur selben Domainklasse äquivalent sind.

```
semanticEquivalentClass [ApplClass1, ApplClass2] :-
(refersToEquivalentClass [DomainClass, ApplClass1]
&&
refersToEquivalentClass [DomainClass, ApplClass2]).
```

Zwei Objektklassen von verschiedenen Applikationsontologien sind semantisch verwandt, wenn sie äquivalent zu Klassen in der Domänenontologie sind, die wiederum von einander Sub- oder Superklassen sind.

```
semanticRelatedClass [ApplClass1, ApplClass2] :-
(refersToEquivalentClass [DomainClass1, ApplClass1]
&&
refersToEquivalentClass [DomainClass2, ApplClass2]
&&
(subClass [DomainClass1, DomainClass2] ||
subClass [DomainClass2, DomainClass1])).
```

10.4.3 Praktisches Beispiel

Als Beispiel soll hier das Beispiel aus dem Papier „Ontology-Based Geographic Data Set Integration“ dienen. Die linke Grafik in Abbildung 10.3 zeigt die reale Welt wie sie in der Domänenontologie aussieht. Sie enthält unter anderem einige Gebäude, eine Straße und verschiedene Landstücke, die unterschiedlich verwendet werden. Die rechte Grafik liefert die zugehörige Klassenhierarchie.

Ganz offensichtlich sind „arableland“ und „grassland“ Unterklassen von Terrain („TRN“) und „conngt4“ eine Unterklasse von „ridingtracks“. Für die Taxonomierung gilt also:

```
taxon [arableland, TRN].
```

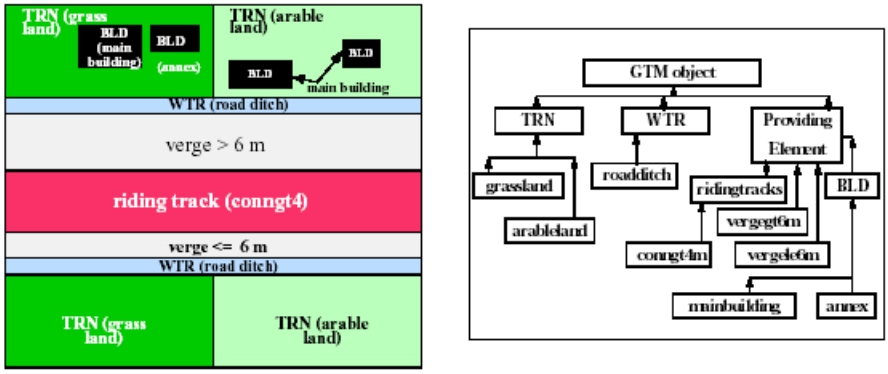


Abbildung 10.3: Die linke Grafik zeigt das Modell der realen Welt in der Domänenontologie, die rechte Grafik die zugehörige Klassenhierarchie [UvOMM99]

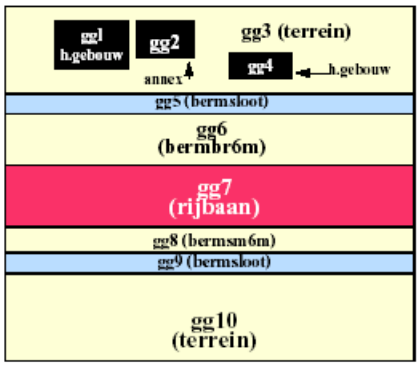


Abbildung 10.4: Abstraktion der realen Welt, wie sie von GBKN benutzt wird; gg1, gg2,... , gg10 sind Objektinstanzen [UvOMM99]

```
taxon [grassland, TRN].
taxon [conngt4, ridingtracks].
```

Für diese Region werden Daten von zwei Quellen angeboten. Abbildung 10.4 zeigt die Applikationsontologie, wie sie von GBKN genutzt wird, und Abbildung 10.5 die Applikationsontologie von TOP10vector.

GBKN unterscheidet offensichtlich nicht zwischen arableland und grassland. Hier werden also die Objektklassen „arableland“ und „grassland“ aus der Domainontologie zu der Klasse „terrein“ in der Applikationsontologie auf-aggregiert.

```
refersToAggregateClass[grassland, terrein].
refersToAggregatClass[arableland, terrein].
```

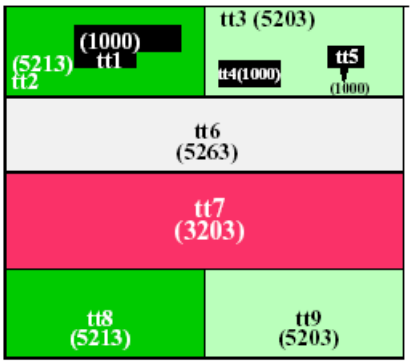


Abbildung 10.5: Abstraktion der realen Welt, wie sie von TOP10vector benutzt wird; tt1, tt2,... , tt10 sind Objektinstanzen [UvOMM99]

Die Klasse „ridingtracks“ scheint hingegen äquivalent umgesetzt zu sein. Sie heißt in der Applikation „rijbaan“. Somit ergibt sich für den Ausdruck in Prolog:

```
refersToEquivalentClass [ridingtracks, rijbaan].
```

10.4.4 Auffinden von korrespondierenden Objekten

Der eben vorgestellte ontologiebasierte Ansatz dient dazu, semantische Ähnlichkeit zwischen Objekten zu überprüfen. Das ist aber nicht ausreichend, um korrespondierende Objekte aus zwei unterschiedlichen Datenquellen zu finden. Entscheidend ist bei den geographischen Objekten, um die es sich hier handelt, die räumliche Lage. Dazu werden die Zwischenergebnisse, die die einzelnen Quellen auf eine Anfrage hin geliefert haben, anhand ihrer Koordinaten übereinander gelegt. Dadurch entsteht eine neue Einteilung in verschiedene Oberflächen (Abbildung 10.6).

So entstand aus gg1 und tt1 f1, aus gg2 und tt1 f3 und aus gg3 und tt1 schließlich f2. Jede dieser neuen Oberflächen ist exakt einer Objektinstanz in jeder der Quellen zugeordnet. Wenn sich ein Objekt also mit mehreren Objekten aus einer anderen Quelle überschneidet, dann entstehen entsprechend auch mehrere Oberflächen.

Nun wird eine Tabelle erstellt, in der jede der Oberflächen genau einen Eintrag bekommt. Ihm werden die beiden zugeordneten Objektinstanzen sowie deren zugehörige Klassen hinzugefügt. Schließlich kommt dann noch ein weiterer Eintrag hinzu, der die semantische Übereinstimmung der Klassen aufgrund der in Prolog eingegeben Daten angibt.

Für die Oberflächen f1, f2 und f3 bedeutet dies folgendes: f1 ist den Objekt-

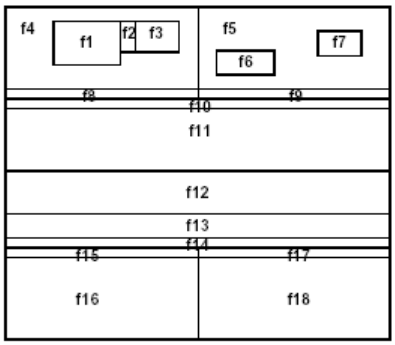


Abbildung 10.6: Kombination von GBKN (Abbildung 10.4) und TOP10vector (Abbildung 10.5) [UvOMM99]

instanzen gg1 und tt1 zugeordnet. f2 ist den Objektinstanzen gg3 und tt1 zugeordnet und f3 den Objektinstanzen gg2 und tt1. gg1 ist eine Instanz der Klasse „hoofdgebouw“, gg2 der Klasse „bijgebouw“ und gg3 der Klasse „terrein“. tt1 gehört zu der Klasse „1000“. „Hoofdgebouw“ und „bijgebouw“ sind letztlich Klassen, die eine Art von Gebäude dartsellen. Gleiches gilt für „1000“. Sie sind äquivalent zu Klassen in der Domänenontologie, die wiederum von einander Sub- oder Superklassen sind. Deshalb wird bei f1 und f3 als semantische Übereinstimmung „related“ also verwandt angegeben. Anders verhält es sich bei f2. gg3 ist eine Landschaftsoberfläche. Diese ist mit einem Gebäude „incompatible“.

In einem letzten Schritt werden jetzt die Objektinstanzen in Listen gespeichert. In die erste kommen Objekte, für die eine semantische und räumliche Übereinstimmung gefunden wurde.

```
{
   { {gg1,gg2},{tt1} },
   { {gg10},{tt8,tt9} },
   ...
}
```

In der zweiten Liste werden die Objekte gespeichert, für die keine korrespondierenden Objekte gefunden wurden.

```
{
   { {gg5},{} } ,
   ...
}
```

Welche Konsequenzen aus diesen Listen folgen, ist nicht genau festgelegt. Durchaus denkbar, dass der User hier selber angeben kann, ob und nach welchem System zusammengefasst werden soll.

Face-id	GBKN-oid	GBKN-class	TOP-oid	TOP-class	Semantic similarity
f1	gg1	hoofdgebouw	tt1	1000	related
f2	gg3	terrein	tt1	1000	incompatible
f3	gg2	bijgebouw	tt1	1000	related
f4	gg3	terrein	tt2	5213	relevant
f5	gg3	terrein	tt3	5203	relevant
f6	gg4	hoofdgebouw	tt4	1000	related
f7	gg3	terrein	tt5	1000	incompatible
f8	gg5	bermsloot	tt2	5213	incompatible
f9	gg5	bermsloot	tt3	5203	incompatible
f10	gg5	bermsloot	tt6	5263	incompatible
f11	gg6	bermbr6m	tt6	5263	relevant
f12	gg7	rijbaan	tt7	3203	relevant
f13	gg8	bermsm6m	tt7	3203	relevant
f14	gg9	bermsloot	tt7	3203	incompatible
f15	gg9	bermsloot	tt8	5213	incompatible
f16	gg10	terrein	tt8	5213	relevant
f17	gg9	bermsloot	tt9	5203	incompatible
f18	gg10	terrein	tt9	5203	relevant

Tabelle 10.1: Die Oberflächen aus dem Beispiel mir der semantischen Ähnlichkeit ihrer Klassen, Quelle: [UvOMM99]

Literaturverzeichnis

[BEL02] O. Boucelma, M. Essid, and Z. Lacroix. A wfs-based mediation system for gis interoperability. In *Proc. Conf. GIS'02, November 8-9, 2002, McLean, Virginia, USA*, pages 23–28, 2002.

[KR99] B. König-Ries. *Ein Verfahren zur semi-automatischen Generierung von Mediatorspezifikationen, infix-Verlag, St. Augustin, Deutschland.* PhD thesis, Universität Karlsruhe, 1999.

[UvOMM99] H. Uitermark, P. v. Oosterom, N. Mars, and M. Molenaar. Ontology-based geographic data set integration. In *Proc. International Workshop on Spatio-Temporal Database Management STDBM'99 (M. H. Böhlen, C. S. Jensen, and M. O. Scholl, eds), pp. 60-78. Edingburgh, Scotland, UK, September, 10-11, 1999. Lecture Notes in Computer Science, Vol. 1678. Springer, Berlin.*, pages 1–19, 1999.

Gesamt-Literaturverzeichnis

[ABJ+02] Michael O. Akinde, Michael H. Böhlen, Theodore Johnson, Laks V.S. Lakshmanan, and Divesh Srivastava, editors. *Efficient OLAP query processing in distributed data warehouses*, 2002.

[AQM+97] S. Abiteoul, D. Quass, J. McHugh, J. Widom, and J.L.Wiener. The LOREL Query Language for Semistructured Data. In *International Journal on Digital Libraries*, pages 68–88, 1997.

[B103] Forschungsgruppe Nexus B1. B1 – Modellierung und Verwaltung des Umgebungsmodells. http://www.nexus.uni-stuttgart.de/de/forschung/dokumente/docs-teilprojekte/poster-teilprojekt-b1.pdf, Dezember 2003.

[Beh02] Jörg Behl. Analyse und Vergleich von semistrukturierten und objektorientierten Datenmodellen. In *Studienarbeit an der Universität Stuttgart am Institut für Parallele und verteilte Höchstleistungsrechner*, 2002.

[BEL02] O. Boucelma, M. Essid, and Z. Lacroix. A wfs-based mediation system for gis interoperability. In *Proc. Conf. GIS'02, November 8-9, 2002, McLean, Virginia, USA*, pages 23–28, 2002.

[BG00] Andreas Bauer and Holger Günzel. *Data-Warehouse-Systeme*. dpunkt.Verlag, Germany, 2000.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 322–331. ACM Press, 1990.

[BM72] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica*, 1(3):173–189, 1972.

[Bri04] R. Brinkman. Efficient tree search in encrypted data. Technical report, Univeristy of Twente, Enschede, the Netherlands, http://www.ub.utwente.nl/webdocs/ctit/1/000000f3.pdf, 2004.

[CCS93] E.F. Codd, S.B. Codd, and C.T. Salley. Providing olap to user-analysts: An it mandate. Technical report, E. F. Codd and Associates, 1993. <http://www.arborsoft.com/essbase/ppr_pspdf.html>.

[CDSS98] S. Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your Mediators need Data Conversion! In *Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, 1998.

[CFK+01] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[Chr03] Björn Christmann. *Verarbeitung von Sensordaten - Seminar Data Warehousing im Verkehrsbereich in SS 2003*. IPD, Universität Karlsruhe (TH), 2003.

[Cla98] Nils Clausen. *OLAP - Multidimensionale Datenbanken*. Addison-Wesley, Germany, 1998.

[Clu97] S. Cluet. Modeling and Querying Semi-structured Data. In *SCIE*, pages 192–213, 1997.

[DCAEA01] Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Storage and retrieval of moving objects. pages 173–184. Springer Verlag, 2001.

[FGNS00] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proceedings of ACM SIGMOD Int. Conf. On Management of Data*, pages 319–330, Dallas, Texas, 2000.

[FKNT02] Ian Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. http://www.globus.org/research/papers/ogsa.pdf, June 2002.

[GMPQ+95] H. Garcia-Molina, Y. Papakonstantinou, Dallan Quass, Anand Rajaraman, Y. Sagiv, Jeffrey Ullman, V. Vassalos, and Jennifer Widom. The Tsimmis Approach to Mediation. In *The Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95), 27 - 29 June 1995, Hotel Carlton, Naharia, Israel*, 1995.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[Inm96] W.H. Inmon. *Building the Data Warehouse*, volume Second Edition. John Wiley & Sons, New York, USA, 1996.

[Kal99] L. Kalinichenko. Integration of Heterogeneous Semistructured Data models in the Canonical One, 1999.

[Koo04] Erik Koop. *Datenbankunterstützung für imperfekte Daten im Verkehrsumfeld*. PhD thesis, IPD, Universität Karlsruhe (TH), 2004.

[KR99] B. König-Ries. *Ein Verfahren zur semi-automatischen Generierung von Mediatorspezifikationen, infix-Verlag, St. Augustin, Deutschland*. PhD thesis, Universität Karlsruhe, 1999.

[Kub00a] J. Kubiatowicz. Distributed object location in a dynamic network. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/papers/pdf/tocs03.pdf, 2000.

[Kub00b] J. Kubiatowicz. Oceanstore: An architecture for global-scale persistent storage. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/talks/ASPLOS-OceanStore.pdf, 2000.

[Kub01] J. Kubiatowicz. Maintenance-free global data storage. Technical report, Univeristy of Califorina, Berkley, http://oceanstore.cs.berkeley.edu/publications/papers/pdf/ieeeic.pdf, 2001.

[Kub03] J. Kubiatowicz. Oceanstore status and directions. Technical report, Univeristy of California, Berkley, http://oceanstore.cs.berkeley.edu/publications/talks/tahoe-2003-01/kubitron(underscore)overview.pdf, 2003.

[Kur98] Andreas Kurz. Data warehousing im internet / intranet: Prototypische implementierung eines web-basierten executive information system für entscheidungsträger. Technical report, Institute of Software Technology, Technical University of Vienna, Vienna, Austria, 1998.

[Kur99] Andreas Kurz. *Data Warehousing - Enabling Technology*. MITP-Verlag GmbH, Bonn, Germany, 1999.

[MAG+97] J. McHugh, S. Abiteoul, R. Goldman, D. Quass, and J. Widom. LORE: A Database Management System for semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[mdx] http://www.microsoft.com. Informationen zu MDX.

[MSJC02] Samuel Madden, Robert Szewczyk, Michael J.Franklin, and David Culler. *Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks*. 2002.

[Nai03] Premchand S. Nair. *Uncertainty in Multi-Source Databases*. Springer, 2003.

[New] news://comp.databases.olap. Newsgroup für OALP.

[NGS+01] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A model-based, open architecture for mobile, spatially aware applications. In C.S. Jensen, M. Schneider, B. Seeger, and V.J. Tsotras, editors, *Proc. Of the 7th Intl. Symposium on Advances in Spatial and Temporal Databases (SSTD 2001)*, page 117, Redondo Beach, CA USA, July 2001.

[NNNT02] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, and Peter Thanisch, editors. *Constructing an OLAP Cube from Distributed XML Data*, 2002.

[Obr] Philipp Obreiter. Bewegliche objekte in datenbanken. pages 79–99.

[Obr04] P. Obreiter. *Bewegliche Objekte in Datenbanken*, chapter 6, pages 79–99. Springer Verlag, Heidelberg, 2004.

[olaa] http://www.olapcouncil.org. Das OLAP-Council.

[olab] http://www.olapreport.com. Der OLAP Report.

[OVI02] OVID. Stärkung der SelbstOrganisationsfähigkeit im Verkehr durch I+K-gestützte Dienste, Universität Karlsruhe (TH), 2002. http://www.ovid.uni-karlsruhe.de.

[PJ01] Dieter Pfoser and Christian S. Jensen. Querying the trajectories of on-line mobile objects. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 66–73. ACM Press, 2001.

[PW03] Raymond Pon and Wesley W.Chu. *Challenges and Issues in Querying Sensor Networks*. 2003.

[Sam90] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[SJ97] S.Cluet and J.Siméon. Data Integration based on data conversion and restructuring. In *Technical report ,Verso database group INRIA*, 1997.

[SJLL02] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing of moving objects for location-based services. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 463. IEEE Computer Society, 2002.

[SWCD97] A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.

[TPS03] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *Proceedings of the 19th VLDB Conference*, pages 790–801, 2003.

[TWZC02] Goce Trajcevski, Ouri Wolfson, Fengli Zhang, and Sam Chamberlain. The geometry of uncertainty in moving objects databases, 2002.

[UvOMM99] H. Uitermark, P. v. Oosterom, N. Mars, and M. Molenaar. Ontology-based geographic data set integration. In *Proc. International Workshop on Spatio-Temporal Database Management STDBM'99 (M. H. Böhlen, C. S. Jensen, and M. O. Scholl, eds), pp. 60-78. Edingburgh, Scotland, UK, September, 10-11, 1999. Lecture Notes in Computer Science, Vol. 1678. Springer, Berlin.*, pages 1–19, 1999.

[Vas] V. Vassalos. Wrapper specification and query processing in the tsimmis project.

[VB02] Steffen Volz and Jan-Martin Bofinger. Integration of spatial data within a generic platform for location-based applications. In *Proc. Of the Joint Intl. Symposium on Geospatial Theory, Processing and Apllications*, Ottawa, Canada, January 2002.

[VGH+02] Steffen Volz, Matthias Großmann, Nicola Hönle, Daniela Nicklas, and Thomas Schwarz. Integration mehrfach repräsentierter Straßenverkehrsdaten für eine föderierte Navigation. *it+ti*, 44(5):260–267, Oktober 2002.

[Wie92] G. Wiederhold. Mediators in the architekture of future information systems. In *in IEEE computer 25:3*, pages 38–49, 1992.

[WXCJ98a] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. *Statistical and Scientific Database Management*, pages 111–122, 1998.

[WXCJ98b] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. pages 111–122, Juli 1998.

[YG03] Yong Yao and Johannes Gehrke. *Query Processing for Sensor Networks.* 2003.

Abbildungsverzeichnis

Tabellenverzeichnis