

Agentenbasierte adaptive und dynamische Anfrageverarbeitung in virtuellen Datenbanksystemen

Marco Grawunder*

Zusammenfassung

Die Anfrageverarbeitung über integrierten, heterogenen und verteilten Datenquellen verlangt nach angepassten Vorgehensweisen und Architekturkonzepten. Dieser Beitrag stellt erste Ideen zu einem agentenbasierten Anfrageverarbeitungssystem vor, welches insbesondere auch in der Lage sein soll, in Fehlersituationen (z.B. Ausfall einer Quelle) reagieren zu können. Zu diesem Zweck werden Konzepte der adaptiven und dynamischen Anfrageverarbeitung eingesetzt. Die Adaptivität ist notwendig, damit initial bereits effizient ausführbare Anfragen erzeugt werden. Die Dynamik soll es ermöglichen auch zur Laufzeit einer Anfrage im Fehlerfall noch einzuschreiten, und so die Ausfallsicherheit einer Anfrageausführung zu erhöhen. Diese Arbeit diskutiert Möglichkeiten, zur Laufzeit einer Anfrage Datenquellen auszutauschen. Dies ist insbesondere in Situationen sinnvoll, in denen keine Interaktion mit dem Nutzer stattfindet und Anfragen eine lange Laufzeit besitzen (z.B. Publish/Subscribe-Szenario oder die Offline-Verarbeitung von Anfragen mobiler Endgeräte).

Schlüsselworte: *dynamische und adaptive Anfrageverarbeitung, Anfrageoptimierung, Agentenarchitektur, virtuelle Datenbanksysteme.*

1 Einleitung

Virtuelle Datenbanksysteme (VDBMS) [GHR98] stellen einen Mittelweg zwischen Föderierten Datenbanksystemen (FDBMS) [Con97] und Data Warehouse (DWH) [BG01] - Systemen dar. FDBMS finden vor allem dann Anwendung, wenn relativ wenige Quellen mit eher statischen Schemata integriert werden sollen. Vor der Ausführung einer Anfrage muss ein globales, integriertes Schema vorliegen, welches sich aus den externen Schemata aller zu integrierenden Datenquellen zusammensetzt. Einer der wesentlichen Aspekte bei FDBMS besteht also in der *Schemaintegration*.

In DWH-Systemen wird hingegen in einem, in der Regel manuellen, so genannten ETL-Prozess eine vollständige Offline-Integration der verschiedenen Datenquellen¹ vorgenommen. Der Begriff „ETL“ bezeichnet dabei die unterschiedlichen Phasen der Integration:

- Extraktion des relevanten Datenausschnitts aus den jeweiligen Quellen,
- Transformation des jeweiligen Datenausschnitts auf das Zielschema des DWH, zusätzlich können hier Datenbereinigungen (Data Cleansing [Hin01]) statt finden,

*Grawunder@informatik.uni-oldenburg.de, Carl-von-Ossietzky Universität Oldenburg, Fachbereich Informatik, Abteilung Informationssysteme.

¹Dies bedeutet insbesondere, dass die eigentlichen Daten von den ursprünglichen Quellen in das DWH transferiert werden, im Gegensatz zu FDBMS bei denen die Daten in der Quelle bleiben.

- Laden der transformierten Daten in das Warehouse.

Anfragen, die in der Regel analyseorientiert (OLAP) sind, werden auf den integrierten Daten vorgenommen. Einer der wesentlichen Aspekte bei DWH besteht also in der *Datenintegration*.

Die beiden grob umschriebenen Ansätze sind ideal, wenn die Menge der zu integrierenden Quellen konstant bleibt, die Hinzunahme oder das Entfernen von Quellen also die Ausnahmen darstellen und die Quellen disjunkte Informationen vorhalten. Schwierigkeiten treten bei überlappenden Quellen, d.h. Quellen mit teilweise redundanten Daten, auf. Die Entscheidung, welche Quellen welche Daten liefern sollen, wird statisch im voraus, quasi zur Entwurfszeit getroffen. Bei DWH-Systemen kann im ETL-Prozess entschieden werden, woraus die Daten extrahiert werden sollen. In FDBMS wird diese Entscheidung durch die Definition der Schematransformationsregeln getroffen. Die Möglichkeit, je nach Situation zu entscheiden, welche Daten verwendet werden sollen, ist in beiden Systemen nur sehr schwer zu realisieren.

In großen Firmen ergibt sich demnach häufig das Problem, dass sich große Datenbestände weder sinnvoll in einem FDBMS noch in einem DWH zusammenfassen lassen (zu viele Daten, zu viele (oft heterogene) Quellen, zu hohe Dynamik, zu hohe Redundanz).

Ein anderes Gebiet in dem FDBMS und DWH nicht gut geeignet sind, stellt das Gebiet der internetbasierten Datenbanken dar. Der Begriff „virtuelle Datenbanken“ beschreibt ursprünglich das Szenario, in dem viele verschiedene, im Internet bereits vorhandene Datenquellen, in einem System vereint werden, um Anfragen über der integrierten Sicht realisieren zu können. Erst zur Laufzeit einer Anfrage wird entschieden, welche der Datenquellen in der Anfrage tatsächlich verwendet werden (Datenintegration) und welche Schemata zu berücksichtigen sind (Schemaintegration). Unterschiedliche Quellen können dabei trotz unterschiedlichem Schema die gleichen Daten enthalten, d.h. die Quellen weisen oft starke *Redundanzen* auf. Da es sich um Quellen im Internet handelt, zeichnen sie sich in der Regel ebenso durch eine starke *Autonomie* aus, eine Kooperation mit dem Eigentümer einer Quelle findet in der Regel nicht statt und Veränderungen der Quelle müssen vom VDBMS selbstständig erkannt werden. Selten ist es möglich, Updates durchzuführen. Eine Eigenschaft, die ebenfalls durch das Internet begründet ist: Die Quellen, die eingebunden werden, weisen oft eine starke *Dynamik* auf, sowohl die Quellen betreffend (Schemaänderung), als auch die Anzahl der Quellen, die verfügbar sind. Da Quellen während der Ausführung einer Anfrage hinzukommen oder wegfallen können, ergibt sich eine neue Qualität bei der Verarbeitung von Anfragen. Die strikte Trennung zwischen der Übersetzungs-, Optimierungs- und Ausführungszeit einer Anfrage kann auf Grund dieser hohen Dynamik nicht mehr länger aufrecht erhalten werden, z.B. muss eine Festlegung der zu verwendenden Quellen zur Optimierungszeit auch zur Ausführungszeit noch verändert werden können. Es ist also notwendig zu einer dynamischen Anfrageverarbeitung zu kommen, die sich auch an neue Gegebenheiten anpassen kann (Adaptivität).

Wie aus den Beschreibungen der VDBMS deutlich wird, liegt hier ein höchst dynamischer und verteilter Ansatz vor, der durch entsprechende Architekturkonzepte angemessen unterstützt werden muss. Agenten [HS98] stellen dabei ein geeignetes Mittel dar, um diesen Anforderungen gerecht zu werden. Im Rahmen dieser Arbeit wird dabei ein Agent wie folgt definiert (vgl. [GK94]):

Ein Agent ist ein autonomes, langlebiges Software-Programm mit einer „gewissen Intelligenz“ (z.B. Lernfähigkeit) und der Fähigkeit zur nachrichtenbasierten Kommunikation mit anderen Agenten.

Durch die Verwendung von Agenten kann insbesondere eine Komplexitätsreduktion des Integrations- und Verarbeitungsprozesses erreicht werden, da weniger globale Prozesse betrachtet werden müssen. Im Unterschied zu einer reinen Komponentenarchitektur hebt

die Verwendung von Agenten die Autonomie der verschiedenen Einheiten hervor, sie sind weitestgehend unabhängig von einer globalen Steuerungskomponente und können auch im Fehlerfall noch selbstständig versuchen, auftretende Probleme zu lösen. Das Agentenkonzept wurde gewählt, um die Autonomie der einzelnen Komponenten sowie eine lose Kopplung zwischen den Komponenten zu erreichen. Hinzu kommt die Fähigkeit der Agenten, Wissen zu erlangen, welches in zukünftigen Situationen helfen kann gestellte Aufgaben effizienter zu erfüllen.

Der Rest dieses Beitrags ist wie folgt aufgebaut. Im folgenden Abschnitt werden zunächst Ansätze beschrieben, die sich ebenfalls mit einer dynamischen und adaptiven Anfrageverarbeitung beschäftigen, bevor im Abschnitt 3 zum besseren Verständnis ein Anwendungsszenario definiert wird. Der in dieser Arbeit beschriebene Ansatz, die Quellen einer Anfrage erst zur Laufzeit festzulegen und gegebenenfalls auch auszutauschen, benötigt eine Beschreibung von Datenquellen. Erste Überlegungen wie eine solche Beschreibung aufgebaut sein kann, sind in Abschnitt 4 zu finden. Anschließend wird die Architektur (Abschn. 5) erläutert, die den eigentlichen Quellenwechsel (Abschn. 7) innerhalb des Anfrageverarbeitungsprozess (Abschn. 6) ermöglicht. Die Arbeit schließt mit einer kurzen Zusammenfassung (Abschn. 8).

2 Verwandte Arbeiten

Die Anfrageverarbeitung wird in der Regel in die drei Phasen Anfrageübersetzung, Anfrageoptimierung und Anfrageausführung unterteilt [Gra93]. Wie oben beschrieben, kann diese strikte Trennung zwischen Optimierung und Ausführung einer Anfrage nicht in allen Fällen aufrecht erhalten werden. Abbildung 1 versucht die verschiedenen Ansätze der Anfrageoptimierung in einer Übersicht zusammenzufassen. Wie zu sehen ist, kann die Optimierung in die beiden Phasen *statische Optimierung*, die im wesentlichen der klassischen Optimierung entspricht, und *dynamische Optimierung* unterteilt werden. Die statische Optimierung wendet dabei Restrukturierungsregeln an, um die Ausführung der Anfrage – in Form eines Operatorbaums – so effizient wie möglich zu gestalten. Dabei greift die Optimierung bei der Auswahl der Quellen und der Zugriffsmöglichkeiten (Baumzugriff, Hash-Zugriff, Full-Table-Scan, etc.) auf Metadaten zurück, die bei der Entscheidung helfen sollen, welche der möglichen Ausführungspläne der geeignetste ist. Da dieses Wissen in einer dynamischen Umgebung sehr schnell ungenau oder gar ungültig werden kann (z.B. kann sich die Erreichbarkeit einer Quelle auf Grund von hohen Auslastungen verringern), sind verschiedene Ansätze entwickelt worden, die unterschiedliche Teilaspekte des Anfrageverarbeitungsprozesses betrachten und als Ausgangspunkt für eine Anfrageoptimierung dienen können.

Für den Fall, dass die im Ausführungsplan berücksichtigten Datenquellen nicht schnell genug oder nur blockweise² antworten, kann eine als *Query Scrambling* bezeichnete Technik eingesetzt werden. Sie versucht die durch die Verzögerungen entstehenden Lücken durch die Ausführung anderer Operatoren des Ausführungsplans zu füllen. Dabei wird die Zusammensetzung des Operatorgraphen jedoch nicht verändert: es bleibt dieselbe Menge von Operatoren vorhanden, nur die Reihenfolge wird modifiziert. Bei einem Quellenausfall führt dieser Ansatz zu einem Abbruch der Verarbeitung [UFA98, FMR⁺99]. Er arbeitet damit auf der Plan-Ebene, die in der Abbildung 1 auch als „Query Scrambling Zone“ bezeichnet wird.

Auf dieselben Probleme reagiert eine andere Gruppe von Ansätzen, die man auf der Operator-Ebene ansiedeln kann und die jeweils spezielle Operatoren zum Datenzugriff einführen. Beispiele hierfür sind *Pipelined Hash Join* [IFF⁺99] und *XJoin* [UF99, UF00]

²Auch „bursty arrival“ genannt [AFT98]

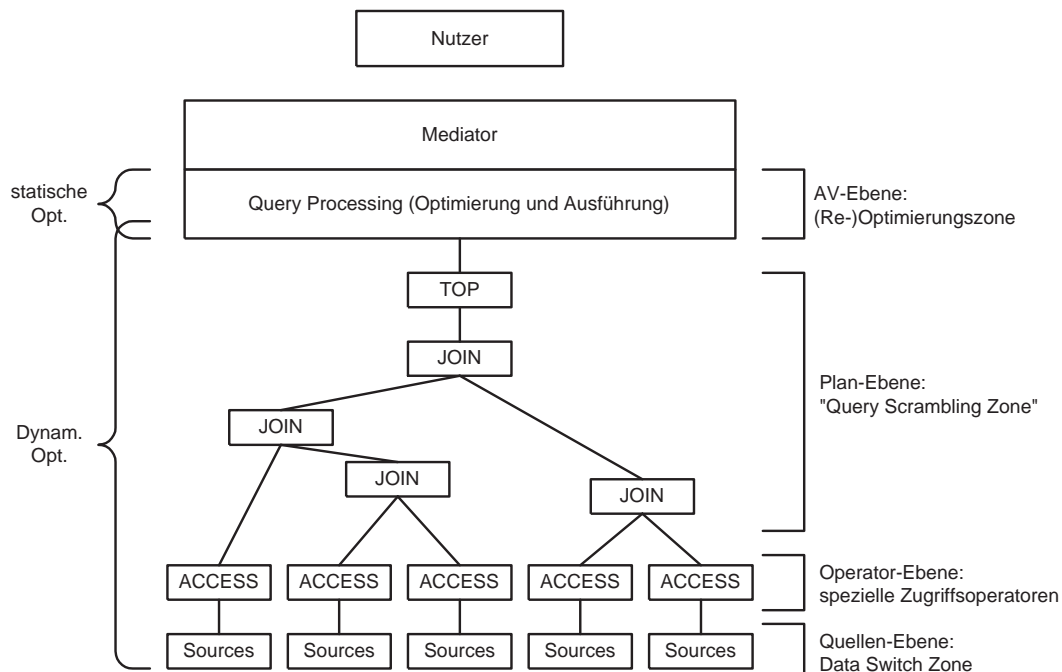


Abbildung 1: Ebenen der Anfrageoptimierung

für das effiziente Verarbeiten von Joins über verteilten Quellen und ein *XScan*-Operator [ILW00b], der auf „gestreamten“ XML-Daten ein effizientes Verarbeiten von XPath-Ausdrücken [CD99] erlaubt.

Ein spezieller *Collector*-Operator [IFF⁺99] wird im *Tukwilla*-System verwendet. Dieser Operator spricht eine Reihe von äquivalenten Quellen zu Beginn der Verarbeitung gleichzeitig an und nur die Quelle, die über einen bestimmten Zeitraum am schnellsten antwortet, wird weiter verwendet. Wenn nach der Auswahl diese Quelle ausfällt, wird diese Anfrage abgebrochen und muss neu geplant werden. Es wird nicht die Vergangenheit der Verarbeitung berücksichtigt, also jedes mal erneut getestet welche Quellen am geeignetsten sind. Qualitätsüberlegungen (z.B. bezüglich Vertrauenswürdigkeit oder Aktualität) bleiben bei der Auswahl der Quellen unberücksichtigt. Zur Berücksichtigung der Vergangenheit wurde in [ZBR⁺00] ein Ansatz vorgeschlagen, der mit Hilfe einfacher Statistiken versucht, das Antwortverhalten von Quellen zu verschiedenen Zeitpunkten vorauszusagen und auf diese Weise immer eine zum aktuellen Zeitpunkt günstige Quelle zu verwenden (Lernen des Antwortzeitverhaltens unterschiedlicher Datenquellen).

Schließlich existieren noch spezielle architekturbasierte Ansätze wie das oben bereits genannte *Tukwilla* [IFF⁺99, ILW⁺00a], in dem insbesondere Wert auf ein Regelsystem gelegt worden ist, welches für verschiedene Laufzeitprobleme (Timeouts, Erreichen von Speichergrenzen, etc.) unterschiedliche Reaktionen (bis hin zum Neuplanen) erlaubt. In [BFMV00] wird für jede Datenquelle ein spezieller clientseitiger Puffer verwendet, in den unabhängig vom Bearbeitungsstand der Anfrage so viele Daten wie möglich von den jeweiligen Quellen geladen werden. Dieses einfache Prefetching kann jedoch erst nach der Übersetzung und Optimierung der Anfrage erfolgen. Den Ausfall einer Datenquelle kann auch dieser Ansatz nicht ohne weiteres verkraften.

Ein echtes Quellenwechseln zur Kompensation eines Quellenausfalls zur Laufzeit findet somit bei keinem der beschriebenen Ansätze statt. Der Bereich „Data Switch Zone“, bei dem auch ähnliche Quellen für die Beantwortung einer Anfrage herangezogen werden, wird nicht ausreichend berücksichtigt.

3 Anwendungsszenario: „Virtuelles Reisebüro“

Um die Ausführungen im Folgenden verständlicher zu gestalten, wird in diesem Abschnitt ein kleines Anwendungsszenario definiert. Das hier betrachtete Anwendungsszenario stellt ein „virtuelles Reisebüro“ für mobile Endgeräte (wie Palm oder WAP-Handy) dar. Dabei ergeben sich einige Besonderheiten, die bei der Verarbeitung von Anfragen berücksichtigt werden müssen:

- Onlinezeiten sind teuer, d.h. die Anfrage muss autonom und insbesondere auch im Fehlerfall ohne Nutzerinteraktion ablaufen können (soweit dies möglich und sinnvoll ist).
- Ein kleines, häufig nur zweifarbiges Display, d.h. die Anfrageformulierung und die Ergebnisrepräsentation müssen sehr einfach gehalten sein.
- Geringe Bandbreite, d.h. sowohl die Anfrage als auch das Ergebnis muss sehr kompakt sein, z.B. muss bei Suchmaschinenanfragen das Ranking sehr gut sein (relevante Treffer vorne).
- U.U. kann es sinnvoll sein, auch den aktuellen Ort des Nutzers in die Verarbeitung zu integrieren (localisation based services).

Vor allem die Berücksichtigung des ersten Punktes, die Ausführung der Anfrage weitgehend ohne Nutzerinteraktion, stellt dabei einen Schwerpunkt dieser Arbeit dar.

In dem virtuellen Reisebüro werden unterschiedliche Datenquellen (z.B. Touristik-Unternehmen, Städteführer, Fahrplanauskunft, Geodaten, Wetterdaten) in einem gemeinsamen System integriert. Da es für jede dieser Datenarten unterschiedliche Quellen geben kann (Touristik: TUI, Neckermann, ...³; Wetterdaten: Donnerwetter.de, Wetter.de, ...), muss initial eine Auswahl aus der jeweiligen Gruppe der Quellen stattfinden und an den Anfrageverarbeitungsprozess gebunden werden. Falls unerwartete Störungen beim Zugriff auf die gewählten Quellen auftreten, kann das Wechseln einer Quelle das Scheitern der Anfrage unter Umständen vermeiden. Eine Anfrage an das virtuelle Reisebüro könnte folgendermaßen ablaufen:

1. Ein mobiler Anwender sendet eine Anfrage über die Angebotserstellung für eine Reise auf die Kanarischen Inseln an das System (z.B. per SMS oder WAP). Nach dem Absenden der Anfrage ist der Nutzer nicht mehr online.
2. Aus einem Nutzerprofil könnte ermittelt, zu welcher Zeit der Anwender Urlaub hat, um zunächst einmal den Zeitraum für die Reise eingrenzen zu können. Alternativ kann natürlich die manuelle Eingabe des Reisezeitraums erfolgen.⁴
3. Danach wird vom System ein Ausführungsplan erstellt, der die Quellen enthält, die nach aktueller Einschätzung die Anfrage am besten beantworten. Eine gute Quelle kann dabei sowohl durch die Qualität, als auch durch das Antwortzeitverhalten definiert werden. Welchem Aspekt hier die höhere Priorität gegeben wird, muss im Nutzerprofil abgelegt werden.
4. Falls eine der Quellen ausfällt, sollte die Anfrage nicht abgebrochen werden, da der Nutzer zurzeit eventuell nicht erreichbar ist, sondern es muss versucht werden mit einer alternativen Quelle fortzufahren. Konkretes Beispiel: Der TUI-Server ist nicht erreichbar und muss durch den Neckermann-Server ersetzt werden.

³Hier soll kein Zugriff auf das START-System erfolgen, sondern die Information aus dem Web-Server des jeweiligen Unternehmens extrahiert werden.

⁴Das Nutzerprofil muss dabei nicht bei einem speziellen Anbieter liegen, sondern könnte lokal beim Nutzer (z.B. in seinem Handy gespeichert sein und dieses könnte die Daten dann in dem entsprechenden Formular ausfüllen.

5. Der Nutzer wird darüber informiert, dass das Ergebnis seiner Anfrage vorliegt.
6. Zusätzlich könnten noch Umrechnungen (Kosten der Reise in Euro und DM) und Literaturbestellungen (z.B. Reiseführer Fuerteventura) in das Ergebnis aufgenommen bzw. als Option angeboten werden.
7. Das nächstgelegene Reisebüro, an dem die Reise gebucht oder weitere Informationen eingeholt werden können, kann durch die Verwendung der aktuellen Ortsinformation ermittelt werden.

Wie man an diesem Beispiel bereits erkennen kann, sind eine Reihe von Voraussetzungen notwendig, damit ein solches System funktionieren kann. Zunächst ist es sinnvoll, dass die Vorlieben und Eigenarten des Nutzers dem System bekannt sind, damit in Entscheidungssituationen (Auswahl der Quelle, Auswahl neuer Quellen, etc.) weiteres Wissen herangezogen werden kann. Die Modellierung von Nutzerinformationen wird jedoch im Rahmen dieser Arbeit nicht weiter betrachtet (vgl. hierzu z.B. [AS99]).

Ein anderer wesentlicher Punkt betrifft die Auswahl relevanter Quellen, sowohl zur Übersetzungs- und Optimierungszeit der Anfrage, als auch zur Laufzeit, falls ein Wechsel der Quelle notwendig wird. Zu diesem Zweck müssen Informationen über die jeweiligen Quellen in einem (u.U. verteilten) Repository abgelegt werden. Dieses Repository muss ein effizientes Speichern und Retrieval von Quellenbeschreibungen ermöglichen. Das Retrieval darf sich dabei nicht auf exakte Übereinstimmung beschränken, sondern muss auch die Möglichkeit bieten, einem Muster ähnliche Quellen zu finden. Erste Ansätze wie diese Quellenbeschreibungen modelliert werden können sind im folgenden Abschnitt beschrieben.

4 Quellenbeschreibungen

Damit entschieden werden kann, welche Datenquellen im Verarbeitungsprozess wann und wie berücksichtigt werden, muss für jede Quelle eine Beschreibung zur Verfügung gestellt werden. Diese Beschreibung ist entweder bereits in der Quelle enthalten (z.B. im Header einer HTML-Seite) oder wird von demjenigen zur Verfügung gestellt, der die entsprechende Quelle dem System bekannt macht (Integrator). Eine im System vorhandene Wissenskomponente übernimmt diese Beschreibung und ergänzt sie um eigene Aspekte bzw. modifiziert die eingetragenen Werte entsprechend der zwischenzeitlich gewonnenen Erkenntnisse.

Die Quellenbeschreibungen (innerhalb des Systems) bestehen im wesentlichen aus den drei Bereichen

1. Dateninformation,
2. Struktur- und Dienstinformation sowie
3. Qualitätsinformation,

die im Folgenden näher erläutert werden sollen.

4.1 Quellenbeschreibungen: Dateninformation

In diesem Bereich der Quellenbeschreibungen ist vermerkt, welche Art Information von einer Datenquelle bereitgestellt wird (also welche Daten die Quelle anbieten kann, unabhängig davon, in welchem Schema sie vorliegen). Zu diesem Zweck wird ein globales Klassifikationsschema definiert, in das die Quellen eingeordnet werden müssen. Eine Quelle kann sich auch auf mehrere Teilstrukturen dieses Schemas beziehen. Das Klassifikationsschema ist Teil des globalen Anwendungssystems (z.B. virtuelles Reisebüro). Beim Einfügen neuer

Quellen kann es unter Umständen notwendig werden, das Klassifikationsschema zu erweitern. Bei der Erstellung der Anwendung muss ein Basisklassifikationsschema bereitgestellt werden.

Das Klassifikationsschema kann mit Hilfe einer Ontologie-Beschreibungssprache, wie z.B. DAML/OIL⁵ erstellt werden.

4.2 Quellenbeschreibungen: Struktur- und Dienstinformationen

In diesem Teil der Quellenbeschreibungen ist vermerkt, wie die Objekte aussehen, die von einer Quelle exportiert werden und wie die Zugriffsmöglichkeiten sind, d.h. welche Funktionen aufgerufen werden können bzw. müssen, um die Daten aus dieser Quelle zu extrahieren.

Eine einfache Möglichkeit besteht darin, Eingabe- und Ausgabe-Muster zu definieren und damit eine relationale Abbildung⁶ der Quelle zu realisieren. Die Eingabe-Muster legen fest welche Eingaben eine Quelle verwenden kann. Auf diese Weise können Eingabefelder eines Formulars beschrieben werden. Bei Reisebestellungen könnten beispielsweise das Reisedatum und das Reiseziel als Elemente des Eingabe-Muster dienen. Entsprechende Ausgabe-Muster geben dann an, welche Werte von dem System generiert werden, beispielsweise: Termin, Dauer, Abflughafen, Belegung, Ausstattung/Verpflegung, Preise. Zusätzlich muss festgelegt werden, welche der Eingabe-Muster zu welchen Ausgabe-Muster führen können. Die einzelnen Muster dürfen sich nur aus bekannten Werten zusammensetzen, sie müssen im globalen Klassifikationsschema definiert worden sein, um die Behandlung von Homonymen und Synonymen zu vereinfachen.

Die Muster sollten eine Reihe von Anforderungen erfüllen. Sie sollten sehr leicht zu erstellen und zu warten sowie leicht übertragbar bzw. austauschbar sein. Sie sollten mächtig genug sein, um auch komplexere Strukturen beschreiben zu können (z.B. verschachtelte Strukturen) und müssen ein effizientes Retrieval erlauben. Wenn beispielsweise die Anfrage „Ich möchte die möglichen Abflughäfen für eine Reise auf die Kanarischen Inseln haben und kann als Eingabeparameter das Ziel (Kanarische Inseln) sowie das Reisedatum liefern.“ gestellt wird, sollte das oben beschriebene Muster sehr schnell gefunden werden.

Ein Mittel, diese Ein-/Ausgabe-Möglichkeiten einer Quelle zu beschreiben, könnte in der Verwendung von SOAP⁷ liegen. Man definiert in SOAP Funktionen, die als Eingabeparameter die jeweiligen Eingabe-Muster verwenden und als Ausgabe ein Objekt liefern, welches einem bestimmten Ausgabe-Muster entspricht. Wie die konkrete Umsetzung aussehen wird ist noch nicht klar und bedarf weiterer Untersuchungen.

4.3 Quellenbeschreibungen: Qualitätsinformation

In diesem Abschnitt der Quellenbeschreibungen werden Qualitätsinformationen über eine Quelle abgelegt. Vor allem diese Werte sind natürlich hohen Schwankungen unterlegen und müssen laufend angepasst werden. Im einzelnen kann man sich die folgenden Kriterien (nach [Hin01]) vorstellen:

- *Korrektheit*: Die Eigenschaften eines Datenobjektes stimmen mit denen der Realwelt überein.
- *Konsistenz*: Die Eigenschaften eines Datenobjektes sind logisch widerspruchsfrei.

⁵<http://www.daml.org/>

⁶im wesentlichen also eine Abbildung auf Tupelmengen

⁷<http://www.w3.org/TR/SOAP/>

- *Redundanzfreiheit*: Es gibt keine zwei Objekte, die dieselbe Entität der Realwelt beschreiben.
- *Vollständigkeit*: Alle im modellierten Realweltausschnitt vorkommenden Entitäten sind im System repräsentiert.

Der letzte Punkt kann dabei noch um den Aspekt *Abdeckung* ergänzt werden, der festlegt, wie viel Prozent modellierten Daten auch in dieser Quelle vorkommen. Bei keinem der Werte kann man exakte Zahlen angeben, sondern muss sich eher auf Schätzwerte verlassen, die allerdings in einem kontinuierlichen Prozess laufend angepasst werden müssen.

Zusätzlich zu den datenbezogenen Qualitätsinformationen sind auch quellenbezogene Information wichtig. Die *Erreichbarkeit* legt fest zu welchen Zeitpunkten eine Quelle besonders gut oder besonders schlecht verfügbar ist. Wie granular die Parameter sind, könnte man von der Quelle abhängig machen. Beispielsweise werden die Server von Reiseveranstaltern zu Beginn der Ferienzeiten eine schlechtere Erreichbarkeit aufweisen als am Ende, hier rechnet man in Wochen, wohingegen Börseninformation wahrscheinlich eher Schwankungen im Stundenbereich haben werden (eventuell kurz vor Börsenschluss mehr Zugriffe?). Zusätzlich können noch Parameter wie *Aktualität* bzw. *Update-Frequenz* und *Zuverlässigkeit* bzw. *Vertrauenswürdigkeit* berücksichtigt werden.

5 Architektur

Eine Möglichkeit, wie die bisher beschriebenen Konzepte auf eine Anfrageverarbeitungsarchitektur umgesetzt werden können, veranschaulicht die Abbildung 2 zunächst für eine einzelne Quelle.

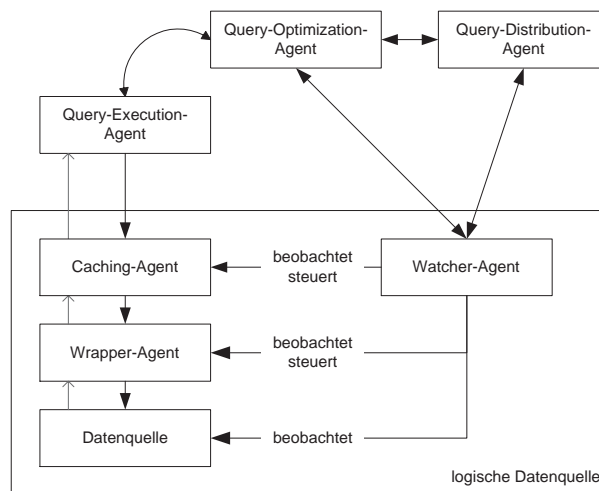


Abbildung 2: Architektur: Ein-Quellen Szenario

Wie zu erkennen ist, wird ein agentenbasierter Mediator/Wrapper-Ansatz verwendet. Die Kommunikation zwischen den Agenten wird über einen Nachrichtenaustausch realisiert.

Jede Datenquelle wird von einem Wrapper-Agenten gekapselt. Dieser Wrapper-Agent hat die Aufgabe, den Inhalt der Datenquellen über ein API zur Verfügung zu stellen. Bei Web-Seiten mit Suchformularen würde dies beispielsweise bedeuten, dass der Wrapper eine Methode anbietet, die eine bestimmte Anzahl von Parametern – die Eingabewerte des Formulars – erwartet und als Ergebnis eine bestimmte Struktur bzw. ein Objekt, mit den

Ergebnissen der Anfragen liefert. Auf diese Weise erreicht man eine Kapselung der Datenquelle von der Anwendung und es ist egal, ob es sich bei der Quelle um eine Web-Seite oder ein vollständiges Datenbanksystem handelt.

Oberhalb des Wrapper-Agenten kann ein Caching-Agent platziert werden, der die Ergebnisse des Wrapper-Agenten zwischenspeichert und der Anfrageverarbeitung (Query Execution Agent) zur Verfügung stellt. Da Wrapper- und Caching-Agenten dieselbe Schnittstelle besitzen, ist für die Anfrageverarbeitung nicht sichtbar, ob sie direkt auf den Wrapper oder über den Cache auf den Wrapper zugreift. Wrapper- und Caching-Agenten müssen für jede Datenquelle speziell erzeugt werden.⁸ Die Platzierung der Agenten kann dabei ein mögliches Optimierungsmittel darstellen.

Ein spezieller Watcher-Agent steuert und beobachtet die Ausführung. Er steuert den Cache, legt also fest, ob bestimmte Daten weiterhin vorgehalten, ob bestimmte Inhalte dauerhaft abgelegt oder bestimmte Inhalte gar nicht eingelagert werden sollen. Durch die Erkennung von wiederkehrenden Mustern im Cache kann ein spezielles Prefetching realisiert werden. Eine weitere Aufgabe des Watcher-Agenten liegt in der Beobachtung der Quellen, um Problemen bei der Anfrageausführung möglichst zuvor zu kommen. Zu diesem Zweck prüft der Watcher-Agent in regelmäßigen Abständen, ob sich die Inhalte bzw. Strukturen der Quelle verändert haben. In Anfragepausen oder festgelegten Intervallen werden Testanfragen an die Quelle geschickt und die Ergebnisse mit initial gewonnenen Werten verglichen. Bei Differenzen kann diese Quelle zur manuellen Überprüfung aus der Menge der zu verwendenden Quellen entfernt werden. Zusätzlich kann auf diese Weise das Antwortzeitverhalten der Quelle zu verschiedenen Zeitpunkten erlernt werden. Durch die Beobachtung von Quelle und Cache ist der Watcher auch in der Lage, die Caches so zu schalten, dass keine Weiterleitung der Anfrage an die Quelle erfolgt, falls diese nicht erreichbar ist und die Anfrage alleine anhand des aktuellen Cache-Inhaltes beantwortet werden kann.

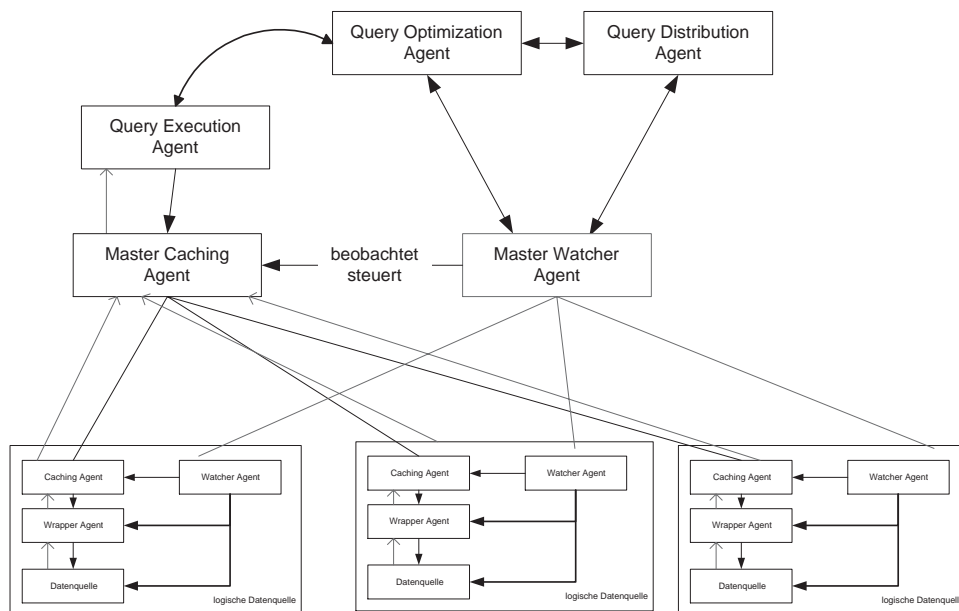


Abbildung 3: Architektur: Logisches Ein-Quellen Szenario

Bisher wurde beschrieben, wie eine einzelne Quelle in das System eingebunden werden kann. Für ein Datenintegrationssystem, das einen Quellenwechsel vollziehen soll, ist

⁸Die Erstellung sollte jedoch semi-automatisch erfolgen können. Die Wrapper sollten eine gewisse Robustheit gegenüber Veränderungen der Quelle besitzen, um den Aufwand bei ihrer Wartung zu minimieren.

dies natürlich nicht ausreichend. Um Mengen von Datenquellen zu kapseln, wird die Architektur wie in Abbildung 3 zu sehen ist um Master-Caching und Master-Watcher ergänzt. Diese Komponenten verhalten sich dabei nach oben, also in diesem Fall in Richtung der Anfrageverarbeitung wie normale Wrapper und Watcher, sind jedoch in der Lage mehrere Wrapper bzw. Watcher zu kapseln. Natürlich können Master-Agenten auch andere Master-Agenten als Quellen verwenden. Insbesondere die Anfrageverarbeitung darf keinen Unterschied zwischen den tatsächlich einer Datenquelle zugeordneten Agenten und den Master-Agenten erkennen.

Ein derartige Verwendung von Agenten erlauben es, auftretende Probleme so lange wie möglich lokal zu behandeln und führt damit zu einer Reduktion der Komplexität. Jeder Master-Agenten-Gruppe wird eine Menge von ähnlichen Quellen (vgl. Abschn. 7) zugeordnet. Wie diese Zuordnung stattfinden soll ist noch nicht klar. Eine Möglichkeit wäre die feste Zuordnung durch den Nutzer. Eine andere wäre die dynamische Zuordnung zur Laufzeit einer Anfrage anhand der Informationen, die in den Dateninformationen der Quellenbeschreibungen (vgl. Abschn. 4.1) vorliegen. Die Master-Agenten übernehmen das Ranking der Quellen und auch ein mögliches Quellenwechseln. Dabei kann es notwendig werden, zusätzliche Operatoren mit in den Anfragegraphen aufzunehmen, um die Veränderungen des Verarbeitungsprozesses durch ein anderes Schema der neuen Quelle berücksichtigen zu können.

Ein mögliches Quellenwechseln findet also auf der Ebene der Master-Agenten statt, im Gegensatz zu speziellen Operatoren der Anfrageverarbeitung wird hier also vor allem die „Data Switch Zone“ als Optimierungsgegenstand betrachtet (vgl. Abb. 1).

Aus den bisher beschriebenen Bausteinen, kann jetzt die gesamte Architektur aufgebaut werden (vgl. Abb. 4). Eine Masteragentengruppe repräsentiert dabei jeweils eigene Quellen in der Anfrageverarbeitung, die Quellen werden für die Abfrageverarbeitung vollständig gekapselt. Inwieweit sich die strikte Trennung der einzelnen Komponenten aufrecht erhalten lässt, wird ein detaillierter Entwurf zeigen. Gute Kandidaten für eine Zusammenlegung von Funktionalitäten wären sicherlich alle Agenten, die zusammen eine logische Datenquelle formen (also Wrapper-, Caching- und Watcher-Agenten).

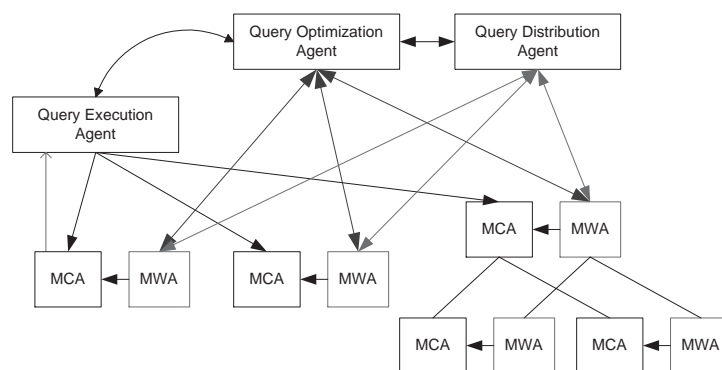


Abbildung 4: Architektur

6 Anfrageverarbeitung

Die konzeptionelle Sicht auf die Anfrageverarbeitung des Systems ist in groben Zügen in Abbildung 5 aufgezeigt. Eine (in eine Interndarstellung übersetzte) Anfrage wird von der Optimierungskomponente entgegen genommen. Mit der zusätzlichen Information aus der

Wissensbasis, die sowohl die Quellenbeschreibungen als auch die Nutzerprofile enthält sowie u.U. weiteren statistischen Informationen, wird die Anfrage optimiert und dann der Ausführungskomponente übergeben. Die Ausführungskomponente führt die Anfrage aus und wird dabei von einem Monitor bzw. einem Controller überwacht. Das Monitoring soll weitere Informationen über das Verhalten bestimmter Quellen gewinnen, also insbesondere die Erreichbarkeitsinformationen der Quellenbeschreibungen (Qualitätsaspekte, vgl. Abschn. 4.3) anpassen. Die Controlling-Komponente dient dazu, den Anfrageausführungsprozess als solchen zu überwachen und unter bestimmten Umständen einzuschreiten. Beispielsweise kann es notwendig werden, die Ausführung einer Anfrage zu unterbrechen und ein Neuplanen vornehmen zu lassen, falls ein Problem auftritt, welches sich nicht lokal (vgl. Abschn. 5) behandeln lässt.

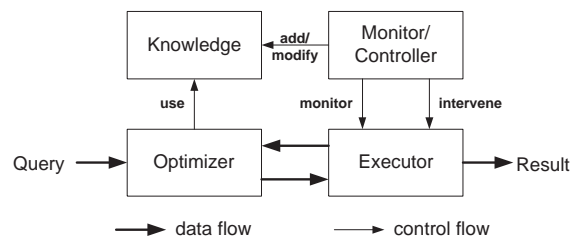


Abbildung 5: Konzeptionelle Sicht auf die Anfrageverarbeitung

Ein besonderes Augenmerk des hier beschriebenen Ansatzes liegt auf der Verwendung ähnlicher Datenquellen zur Optimierung des Anfrageverarbeitungsprozesses. Abbildung 6 beschreibt den Anfrageverarbeitungsprozess noch einmal mit dem Fokus auf dem Aspekt der Quellenwahl. Nachdem die Anfrage dem System übergeben worden ist, müssen die für diese Anfrage relevanten Quellen ermittelt werden. Dabei wird zunächst eine Liste der prinzipiell möglichen Quellen ermittelt, also ob es sich beispielsweise um Informationen über Reisen handelt oder ob es um Fahrplanauskünfte geht. Danach wird für jeden dieser Quellentypen eine Liste der im System bekannten Quellen aufgestellt und nach vom Nutzer zu definierenden Kriterien sortiert (Ranking). Beim Ranking werden in erster Linie die Informationen verwendet, die zu den Quellen und Nutzern in der Wissensbasis abgelegt sind. Nachdem das Ranking durchgeführt worden ist, kann der Anfragebaum erstellt werden, der an seinen Blättern den Zugriff auf die jeweiligen Quellen regelt. Das Laufzeitsystem überwacht die Ausführung.

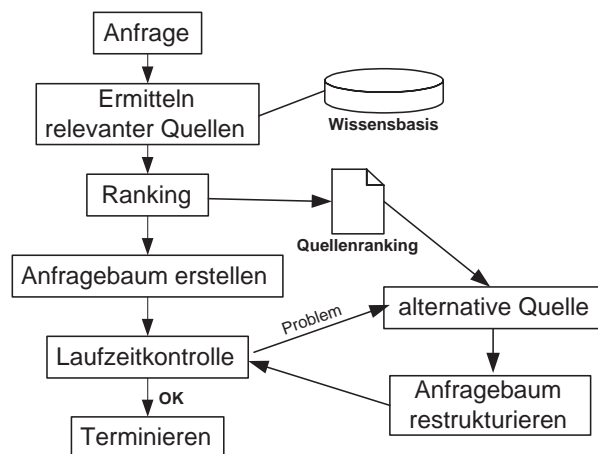


Abbildung 6: Ausführen einer Anfrage

Falls Probleme auftreten, die sich durch einen Quellenwechsel beheben lassen, wird das vorhandene Ranking dieses Quellentyps verwendet und aus dieser Liste die „nächstbeste“ Quelle ausgesucht. Diese Auswahl kann prinzipiell auf zwei unterschiedlichen Wegen erfolgen: Entweder wird einfach die nächste Quelle aus dem Ranking verwendet oder die Quelle, die die aktuelle Anfrage am wenigsten beeinflusst.⁹ Im zweiten Fall würde man eventuell den Nutzerwünschen bezüglich bestimmter Qualitätseigenschaften (Aktualität, Vertrauenswürdigkeit, etc.) nicht mehr gerecht werden, im ersten Fall könnte sich die Ausführung der Anfrage verzögern.

7 Quellenwechsel

Jede Laufzeitoptimierung ist teuer, sie kann in der Regel nie besser sein, als eine optimale statische Optimierung. Trotzdem kann es während der Ausführung einer Anfrage zu dem Problem kommen, dass eine für eine bestimmte Aufgabe ausgewählte Quelle nicht mehr für die Anfrageausführung bereit steht und hier ein Quellenwechsel die Alternative zu einem Abbruch der Verarbeitung darstellt.

Prinzipiell existieren zwei Zeitpunkte, zu denen erkannt werden kann, dass eine Quelle nicht (mehr) den Anforderungen entspricht, die die Anfrageverarbeitung bei der Wahl der Quelle angenommen hat. Der erste Zeitpunkt ist der für die Anfrageverarbeitung günstigste, nämlich die Initialisierung der Anfrage, also der erste Zugriff auf die Quelle. Wenn hier etwas fehlschlägt, kann relativ problemlos auf eine neue Quelle gewechselt werden (natürlich nur soweit eine vorhanden ist). Es sind noch keine Daten im Anfragebaum verarbeitet worden, die eventuell invalidiert werden müssen. Im Tukwilla-System wird genau diesem Aspekt dadurch Rechnung getragen, dass ein spezieller Collector-Operator eingeführt wurde, der mehrere Datenquellen ansprechen kann und nach einer gewissen Zeitspanne entscheidet, welche der Quellen am schnellsten Daten liefert und in der Anfrageverarbeitung berücksichtigt werden soll.

Der weitaus ungünstigere Fall tritt ein, wenn erst nach einiger Laufzeit erkannt wird, dass eine Quelle nicht mehr so reagiert, wie es erwartet worden ist. Das Problem liegt darin, dass zu diesem Zeitpunkt bereits Daten von der Quelle geliefert wurden, die in der Anfrageverarbeitung eventuell schon verwendet worden sind. Als Lösung kann entweder versucht werden die Verarbeitung mit der neuen Quelle an exakt derselben Stelle fortzuführen (dazu müsste die neue Quelle dieselben Daten liefern wie die alte) oder es müssten sämtliche, bereits verarbeiteten Daten annullieren werden. Welche Strategie vorzuziehen ist hängt stark davon ab, wie viele Daten bereits verarbeitet worden sind, welche alternativen Quellen zur Verfügung stehen und wie die Präferenzen des Nutzers sind: Möchte er nur irgendein Ergebnis haben oder sind die Vollständigkeit (soweit sie überhaupt erreichbar ist) und Korrektheit ein wesentliches Kriterium für die Akzeptanz des Ergebnisses. Zur Unterstützung der Entscheidungsfindung ist ein entsprechendes Kostenmodell notwendig, das die unterschiedlichen Parameter entsprechend den Nutzeranforderungen gewichten kann. Dieses Modell ist noch aufzustellen.

Ein wesentliches Kriterium bei der Berücksichtigung alternativer Quellen im Anfrageverarbeitungsprozess ist die Existenz gleichartiger Quellen und deren Einordnung in Ähnlichkeitsklassen. Im Prinzip können zusammengehörige Quellen, wie in Abbildung 7 dargestellt, nach deren Ähnlichkeit klassifiziert werden. *Datenähnlichkeit* bedeutet dabei, dass die Quellen zwar dieselben Daten enthalten, diese aber u.U. in verschiedenen Schemata vorliegen. Im Gegensatz dazu sagt die *Schemaähnlichkeit* aus, dass das Schema bestimmter Quellen einfacher in das Schema anderer Quellen transformiert werden kann als das

⁹also eine Quelle die der aktuellen Quelle ähnlich ist

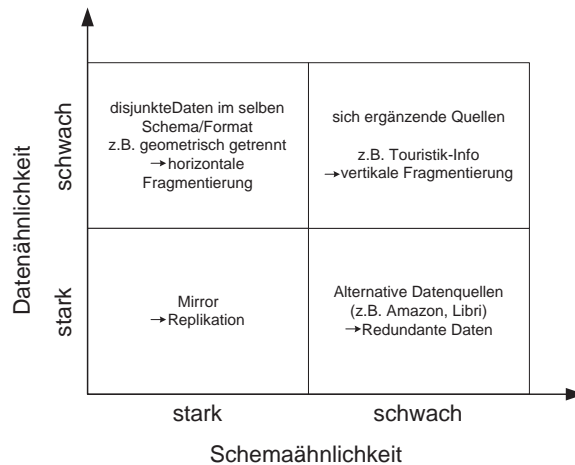


Abbildung 7: Ähnlichkeiten

anderer Quellen. Erst zusammen geben diese Ähnlichkeiten Hinweise auf die Auswahl einer passenden, alternativen Quelle. Je stärker die Ähnlichkeit zwischen zwei Quellen ist, desto einfacher ist ein Quellenwechsel zu realisieren. Das einfachste Beispiel stellt der linke untere Quadrant dar: Die stärkste Daten- und Schemaähnlichkeit liegt immer dann vor, wenn Daten repliziert werden. Im Internet trifft man häufig auf so genannte Mirror-Sites, deren Inhalte z.B. über Nacht auf andere Server transferiert werden, um so eine topologisch gesehen größere Verteilung zu erreichen. In verteilten Datenbanksystemen ist die Replikation ein wichtiges Hilfsmittel, um die lokale Verarbeitung an einem Knoten zu beschleunigen. Analog wird die horizontale Fragmentierung (Quadrant oben links), d.h. die disjunkte Verteilung von Daten nach einem bestimmten Schlüssel, eingesetzt. In diesem Fall lägen verschiedene Quellen mit demselben Schema aber unterschiedlichen Daten vor, die je nach Anwendung auch als Alternativen in einem Quellenwechsel einsetzbar wären.

Im rechten unteren Quadranten finden sich solche Quellen, die zwar prinzipiell dieselben Daten bereitstellen (z.B. Bücher, Fahrpläne oder Reiseangebote), aber die Daten in anderen Schemata anbieten. In der Regel wird es so sein, dass ähnliche Daten auch in ähnlichen Schemata vorliegen, und somit eine Transformation der Daten des einen Schemas in das Schema der anderen Quelle möglich ist. Schließlich befinden sich im oberen rechten Quadranten die Quellen, die zwar in irgendeiner Beziehung zueinander stehen (also beispielsweise ein bestimmtes Themengebiet behandeln), aber weder die gleichen Daten noch die gleichen Schemata besitzen, so dass die Quellen, die in diesem Quadranten zu finden sind, sich nicht ohne weiteres für einen Quellenwechsel verwenden lassen.¹⁰

Es bleibt die Frage unter welchen Umständen ein Quellenwechsel stattfinden kann und wie dieser dann konkret realisiert wird. Die Notwendigkeit eines Quellenwechsels ist bei einem temporären oder permanenten Ausfall einer Quelle offensichtlich. Allerdings kann auch der Fall auftreten, dass eine Quelle den Zugriff verweigert (z.B. weil zu viele Nutzer online sind) oder dass die Quelle gar keine relevanten Daten besitzt, weil z.B. die Annahme beim Quellenranking falsch war. In solchen Fällen muss zunächst eine alternative Quelle ausgewählt und in den Verarbeitungsprozess integriert werden. Wie diese Auswahl und die Integration aussehen wird, ist noch nicht klar und muss noch weiter untersucht werden.

¹⁰In diesem Quadranten lässt sich auch eine vertikale Fragmentierung einordnen, die die Daten anhand bestimmter Attribute auf unterschiedliche Quellen verteilt.

8 Zusammenfassung

In dieser Arbeit wurde ein Ansatz zur agentenbasierten, dynamischen Anfrageverarbeitung in virtuellen Datenbanksystemen vorgestellt. Der Fokus lag dabei vor allem auf Aspekten, die die Möglichkeit schaffen, alternative Quellen innerhalb des Anfrageprozesses zu verwenden. Die nächsten Aufgaben sind die genaue Spezifikation der Quellenbeschreibungen, die Erstellung eines Kostenmodells, welches die Auswahl und das Wechseln von Quellen berücksichtigt, die Entwicklung verschiedener Verfahren für ein Quellenwechseln (unter Berücksichtigung von Quellenähnlichkeiten) sowie eine prototypische Implementierung der Architektur.

Literatur

- [AFT98] Laurent Amsaleg, Michael J. Franklin und Anthony Tomic. Dynamic Query Operator Scheduling for Wide-Area Remote Access. *Journal of Distributed and Parallel Databases*, 6(3):1–33, 1998.
- [AS99] Giuseppe Amato und Umberto Straccia. User profile modeling and applications to digital libraries. In: Serge Abiteboul und Anne-Marie Vercoustre, Herausgeber, *European Conference on Digital Libraries (ECDL99)*, Lecture Notes of Computer Science (LNCS), S. 184–197, Paris, France, 1999. Springer.
- [BFMV00] Luc Bouganim, Francosie Fabret, C. Mohan und Patrik Valduriez. A Dynamic Query Processing Architecture for Data Integration Systems. *Bulletin of the Technical Committee on Data Engineering*, 23(2):42–48, 2000.
- [BG01] Andreas Bauer und Holger Günzel. *Data- Warehouse- Systeme. Architektur, Entwicklung, Anwendung*. dpunkt-Verlag, Heidelberg, 2001.
- [CD99] James Clark und Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, W3C, 16 November 1999 1999.
- [Con97] Stefan Conrad. *Föderierte Datenbanksysteme - Konzepte der Datenintegration*. Springer-Verlag, Berlin Heidelberg New York, 1997.
- [FMR⁺99] Michael J. Franklin, George A. Mihaila, Louiqa Raschid, Tolga Urhan, Maria Esther Vidal und Vladimir Zadorozhny. Searching and Querying Wide-Area Distributed Collections. In: *Russian National Conference on Digital Libraries*, St. Peterburg, 1999.
- [GHR98] Ashish Gupta, Venky Harinarayan und Anand Rajarman. Virtual Database Technology. In: *Proceedings of the Fourteenth International Conference on Data Engineering*, S. 297–301, Orlando, Florida, USA, 1998.
- [GK94] Michael R. Genesereth und Steven P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, July 1994 1994.
- [Gra93] Gotz Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.
- [Hin01] Holger Hinrichs. Datenqualitätsmanagement in Data Warehouse-Umgebungen. In: Andreas Heuer, Frank Leymann und Danny Priebe, Herausgeber, *Datenbanksysteme in Büro, Technik und Wissenschaft*, S. 187–206, Oldenburg, 2001. Springer.
- [HS98] Michael N. Huhns und Munindar P. Singh, Herausgeber. *Readings in Agents*. Kaufmann, 1998.

- [IFF⁺99] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy und Daniel S. Weld. An Adaptive Query Execution System for Data Integration. In: *Proceedings ACM SIGMOD International Conference on Management of Data*, S. 299–310, Philadelphia, Pennsylvania, USA, 1999. ACM Press.
- [ILW⁺00a] Zachary G. Ives, Alon Levy, Daniel S. Weld, Daniela Florescu und Marc Friedman. Adaptive Query Processing for Internet Applications. *IEEE Data Engineering Bulletin*, 23(2):19–26, 2000.
- [ILW00b] Zachary G. Ives, Alon Y. Levy und Daniel S. Weld. Efficient Evaluation of Regular Path Expressions on Streaming XML Data. Technischer Bericht UW-CSE-2000-05-02, University of Washington, 2000.
- [UF99] Tolga Urhan und Michael J. Franklin. XJoin: Getting Fast Answers From Slow and Bursty Networks. Technical Report CS-TR-3994, UNIACS-TR-99-13, University of Maryland, 1999.
- [UF00] Tolga Urhan und Michael J. Franklin. XJoin: A Reactively-Scheduled Pipelined Join Operator. *Bulletin of the Technical Committee on Data Engineering*, 23(2):27–33, 2000.
- [UFA98] Tolga Urhan, Michael J. Franklin und Laurent Amsaleg. Cost-based Query Scrambling for Initial Delays. In: Laura M. Haas und Ashutosh Tiwary, Herausgeber, *SIGMOD*, S. 130–141, Seattle, Washington, 1998. ACM.
- [ZBR⁺00] Vladimir Zadorozhny, Laura Bright, Louiqa Raschid, Tolga Urhan und Maria Esther Vidal. Web Query Optimizer. In: *International Conference on Data Engineering*, S. 661–663, San Diego, USA, 2000.