Konfliktbehandlung in einer Anfragesprache für Datenbankföderationen

Kai-Uwe Sattler Stefan Conrad
Institut für Technische und Betriebliche Informationssysteme
Universität Magdeburg
Postfach 4120, 39016 Magdeburg
{kus|conrad}@iti.cs.uni-magdeburg.de

Zusammenfassung

Ein Hauptproblem bei der Integration heterogener Datenbestände bilden Konflikte, die durch unterschiedliche Modellierung eines Sachverhaltes der Real-Welt, durch verschiedene Datenmodelle oder auch nur durch unterschiedliche Repräsentation der Real-Welt-Objekte entstehen. Die Konflikte müssen im Rahmen der Integration erkannt und bei der Definition der Abbildung zwischen globalen und lokalen Schemata aufgelöst werden. Da diese Abbildung die Grundlage für die Bearbeitung von Anfragen ist, ergibt sich eine enge Verzahnung von Konfliktbehandlung, Anfragetransformation und -ausführung. Vor diesem Hintergrund wird in diesem Beitrag eine Anfragesprache für Datenbankföderationen vorgestellt und die Behandlung der wichtigsten Konflikte mit den Mitteln dieser Sprache diskutiert.

1 Einführung

Die Integration heterogener Datenbestände stellt nach wie vor ein aktuelles Problem dar. So existieren einerseits die über Jahre gewachsenen Bestände in den Unternehmen und die Vielzahl von öffentlich zugänglichen Quellen wie z.B. im Internet. Andererseits führt gerade die damit verbundene Informationsflut zum Wunsch nach einer Integration und Verdichtung dieser Daten.

In den vergangenen Jahren wurde eine Vielzahl von Integrationsansätzen entwickelt. Stellvertretend seien hier Multidatenbanksysteme, Mediatoren und föderierte Datenbanksysteme genannt. Alle diese Ansätze basieren mehr oder weniger auf der Idee einer integrierten Sicht auf die verschiedenen Datenbestände, wobei diese Sichtweise für föderierte Datenbanken besonders ausgeprägt ist [SL90, Con97]. Die Ableitung dieser integrierten Sicht ist Gegenstand der Schemaintegration. Im Verlauf des Integrationsprozesses sind die Schemata der einzelnen Quellen zu analysieren, ein globales Schema zu definieren und die Abbildungen zwischen den lokalen Schemata und dem globalen Schema festzulegen. Mit Hilfe dieser Abbildungsinformationen können Anfragen auf globaler Ebene in Teilanfragen für die einzelnen Datenquellen (die oft auch als Komponentendatenbanken bezeichnet werden) zerlegt und übersetzt sowie die ermittelten Teilergebnisse auf globaler Ebene wieder zusammengesetzt werden [YM98].

Unabhängig von der „Richtung“ des Entwurfsprozesses (bottom-up, d.h. die Integration aller relevanten Daten bzw. top-down mit dem Ziel der Integration von Daten für ein konkretes Ziel [Has99]) sind dabei Konflikte zu lösen, die sich aus der Heterogenität der Bestände und der beteiligten Systeme ergeben. Beispiele hierfür sind unterschiedliche Bezeichnungen für den gleichen Sachverhalt, die Verwendung verschiedener Modellierungskonzepte zur Abbildung von Real-Welt-Objekten oder Konflikte, die durch Überlappung von Daten auftreten. Die Auflösung dieser Konflikte fließt ein in die Abbildung zwischen globalen und lokalen Schemata und ist damit wesentlicher Bestandteil der Anfragebearbeitung.

Der vorliegende Beitrag beschreibt die Anfragesprache FRAQL[1], eine SQL-Erweiterung zur Definition integrierter, objektrelationaler Schemata sowie zur Formulierung von Anfragen über diesen. Im Mittelpunkt stehen dabei die Mechanismen zur Behandlung von Integrationskonflikten. Nach einer Diskussion der verschiedenen Konfliktarten in Abschnitt 2 sowie verwandter Arbeiten (Abschnitt 3), werden wir in Abschnitt 4 zunächst die Anfragesprache und die zugrundeliegende Integrationsarchitektur vorstellen. Abschnitt 5 beschreibt schließlich die Konfliktbehandlung mit Hilfe von FRAQL. Abschnitt 6 beschließt den Beitrag mit einer Zusammenfassung und gibt einen Ausblick auf die weitere Arbeit.

2 Konflikte bei der Integration

Ein zentrales Problem bei der Integration von Datenbanken sind die vielfältigen Konflikte, die zwischen den Schemata der zu integrierenden Datenbanken auftreten können. Nur wenn diese Konflikte erkannt wurden und für sie Lösungen bereitgestellt werden, können Anfragen an Daten aus mehreren Datenbanken (oder allgemein Datenquellen) sinnvolle Ergebnisse liefern.

Viele Arten von potentiellen Schemakonflikten sind in den letzten Jahren ausgiebig untersucht worden. Eine einfache und für die Praxis gut verwendbare Klassifikation von Konfliktarten unterscheidet vier Arten von Konflikten (vgl. [SPD92]): semantische Konflikte, Beschreibungskonflikte, Heterogenitätskonflikte und strukturelle Konflikte. Im folgenden erläutern wir diese Konfliktarten und geben typische, häufig auftretende Beispiele an.

Semantische Konflikte. Auch wenn zwei unabhängig voneinander entstandene Schemata den gleichen Weltausschnitt (oder zumindestens gemeinsame Teile davon) beschreiben, so kann es aus verschiedenen Gründen vorkommen, daß bei einander entsprechenden Klassen von Objekten die zugehörigen Mengen von Real-Welt-Objekten nicht wirklich übereinstimmen. Für die Anwendungen, die auf dem einen Komponentensystem laufen, ist vielleicht nur eine Teilmenge der Objekte relevant, die in einem anderen Komponentensystem verwaltet werden.

Außer solchen Teilmengenbeziehungen können auch allgemeinere Überlappungen der zugehörigen Mengen auftreten, bei denen es eine nichtleere Schnittmenge von Objekten gibt oder in denen die jeweils lokal vorhandenen Objektmengen disjunkt zueinander sind, aber aus einer globalen Sicht zusammengehören. Im letzteren Fall könnte es sich z.B. um Klassen von Aufträgen handeln, die von verschiedenen Abteilungen bearbeitet werden. Auch wenn jeder Auftrag nur von genau einer Abteilung behandelt wird, kann es bei der Integration der abteilungslokalen Datenbanken sinnvoll sein, die verschiedenen (disjunkten) Auftragsmengen im integrierten Schema in einer Klasse zusammenzufassen.

Beschreibungskonflikte. Wenn wir in den zu integrierenden Schemata Objektklassen gefunden haben, die gleiche Objekte der realen Welt repräsentieren, so können sie sich doch in der Beschreibung der Eigenschaften dieser Objekte unterscheiden. Im einfachsten Fall sind einzelne Attribute nur in einer Komponentendatenbank vorhanden, weil diese Attribute für die Anwendungen der anderen Komponentendatenbank keine Rolle spielen. Auch können gleiche Eigenschaften durch *unterschiedlich viele Attribute* beschrieben sein, wie dies z.B. bei Adressen häufig passiert (Hausnummer und Postleitzahl als eigenständige Attribute oder als Teil von Straße und Ort).

Auch die Problematik *homonymer* und *synonymer Bezeichnungen* gehört zu den Beschreibungskonflikten. Dies kann sich auf Attributnamen genauso beziehen wie auf Klassen- und Beziehungsnamen. Darüber hinaus gibt es eine Reihe weiterer Beschreibungskonflikte, von denen wir hier nur einige exemplarisch erwähnen wollen. Unterschiedliche Wertebereiche und/oder Datentypen für gleiche Eigenschaf-

[1] FRAQL: **F**eder**a**tion **Q**uery **L**anguage

ten stellen *Wertebereichskonflikte* dar. Bei der Verwendung unterschiedlicher, aber ineinander umrechenbarer Maßeinheiten für eine Eigenschaft liegt ein *Skalierungskonflikt* vor. *Genauigkeitskonflikte* entstehen, wenn eine unterschiedliche numerische Genauigkeit, also z.B. unterschiedliche Anzahl von Nachkommastellen, verwendet wird. Da die abgespeicherten Werte nicht immer exakte Werte sind, können *Vagheitskonflikte* durch die Integration von vagen und exakten Daten auftreten. Ferner können noch Beschreibungskonflikte durch unterschiedliche Integritätsbedingungen oder Manipulationsoperationen für dieselben Eigenschaften gegeben sein.

Heterogenitätskonflikte. Viele Probleme bei der Integration entstehen daraus, daß für die zu integrierenden Schemata unterschiedliche Datenmodelle verwendet wurden. Unterschiedliche Datenmodelle bedingen, daß unterschiedlich viele Modellierungskonzepte zur Verfügung stehen. Ferner gibt es in der Regel semantische Unterschiede zwischen den Modellierungskonzepten. Diese Probleme müssen bei der Schematransformation in das globale Datenmodell beachtet werden. Je nachdem welche Datenmodelle als Quelle und Ziel der Transformation verwendet werden, sind entweder *semantische Verluste* zu vermeiden oder eine *semantische Anreicherung* durchzuführen. Die Heterogenität der Datenmodelle bedingt in der Regel auch *strukturelle Konflikte*, die durch die Schematransformation nicht beseitigt werden können.

Strukturelle Konflikte. Auch innerhalb eines Datenmodells kann oft derselbe Sachverhalt der realen Welt durch Verwendung unterschiedlicher Modellierungskonzepte in verschiedenen Formen beschrieben werden. Zum Beispiel haben wir in objektorientierten Datenmodellen die Wahl, eine bestimmte Eigenschaft einer Klasse von Objekten als wertbasiertes Attribut zu modellieren oder durch Referenz auf Objekte einer anderen Klasse darzustellen, wenn diese den entsprechenden Wert repräsentieren. Ebenso kann eine Eigenschaft einmal als Attribut und in anderen Fällen als Spezialisierung in Unterklassen modelliert werden.

Insbesondere Datenmodelle mit vielen Modellierungskonzepten erlauben viele strukturell unterschiedliche Modellierungen desselben Sachverhalts. Dies bedeutet aber nicht, daß dies in semantisch armen Datenmodellen nicht möglich ist. Ein schönes Beispiel hierfür stellt die Normalisierung relationaler Datenbankschemata dar. Obwohl das Relationenmodell nur wenige Modellierungskonzepte bietet, erlaubt es verschiedene zueinander semantisch äquivalente Modellierungen, die verschiedenen Normalformen genügen. Durch Normalisierung können wir also strukturell unterschiedliche relationale Schemata erzeugen. Und da bei bestehenden relationalen Datenbanken, die integriert werden sollen, Schemata in unterschiedlichen Normalformen vorliegen können (insbesondere auch nicht normalisierte Schemata), müssen solche strukturellen Konflikte auch bei semantisch armen Datenmodellen untersucht werden.

Eine besondere Art struktureller Konflikte sind *Meta-Konflikte*, bei denen Werte von Eigenschaften in einem Schema als konkrete Werte (z.B. eines Objektattributes) gespeichert werden, während dieser Wert in einem anderen Schema als Information im Schema (also in den Metadaten) enthalten ist.

Nicht immer läßt sich ein konkreter Konflikt genau einer dieser vier Klassen zuordnen. Dies liegt darin begründet, daß ein Konflikt durchaus mehrere verschiedene Ursachen und damit verschiedene Erscheinungsformen haben kann.

3 Verwandte Arbeiten

Bevor wir auf konkrete Konfliktlösungsstrategien in unserer Anfragesprache FRAQL eingehen, wollen wir in diesem Abschnitt kurz auf verwandte Ansätze hinweisen.

Dem allgemeineren Problem der Schemaintegration sind eine Reihe von Arbeiten gewidmet [BLN86]. Zu den bei der Integration auftretenden Konflikten wurden ebenfalls verschiedene Klassifi-

kationen vorgeschlagen, so u.a. in [KS91, SCG93, SPD92]. Konkrete Datenmodelle und Anfragesprachen, die die Integration heterogener Daten unterstützen, sind insbesondere Multidatenbanksprachen wie MSQL [GLRS93], SchemaSQL [LSS96] oder SQL/M [KGK+95]. Auf die Behandlung von Konflikten wird dabei aber nur zum Teil eingegangen, so u.a. mit Techniken zur Restrukturierung von Relationen in SchemaSQL. Eine ausführliche Diskussion struktureller Konflikte und Strategien zu deren Auflösung ist dagegen in [KCGS95] zu finden. In [HE99] wird die Homogenisierung relationaler Schemata durch semantische Anreicherung beschrieben.

Ein Ansatz, der die Herkunft der Daten als ein zusätzliches Tupelattribut einbezieht und damit die Interpretation der globalen Daten verbessern will, wird in [LCC99] vorgestellt. Noch weiter geht der Vorschlag semantischer Werte [SSR94], die die Interoperabilität von heterogenen Quellen durch die Repräsentation von Kontextinformationen ermöglichen. Die Komposition von komplexen Anfrageergebnissen (Objektfusion) aus verschiedenen Quellen wird in [PAGM96] behandelt.

Als Beispiele konkreter Systeme, die Anfragen auf heterogenen Datenbeständen unterstützen, seien an dieser Stelle noch föderierte Datenbanksysteme wie IRO-DB [GGF+96], Pegasus [ASD+91] oder (als kommerzielle Lösung) IBM DataJoiner [VZ98] und Systeme auf der Basis der Mediator-Architektur [Wie92] wie TSIMMIS [GPQ+97], Garlic [CHS+95] oder Information Manifold [LRO96] genannt.

4 FRAQL im Überblick

Ziel der Entwicklung von FRAQL ist die Untersuchung von Techniken der Anfragebearbeitung für große, lose gekoppelte Datenbankföderationen, wie sie z.B. über Internet-Quellen gebildet werden können. Eine besondere Herausforderung stellen in diesem Kontext dabei das Fehlen von statistischen Informationen über die Daten und die Zugriffspfade zur Optimierung, die Heterogenität, Überlappung und Redundanz der Daten sowie die Fähigkeiten zur Anfragebearbeitung und die nicht vorhersagbaren Antwortzeiten der Datenquellen dar [LRO96, IFF+99]. Im Mittelpunkt dieses Beitrags steht jedoch nur einer dieser Aspekte: die Behandlung von Integrationskonflikten durch die Heterogenität der Datenquellen.

FRAQL ist eine Anfragesprache für objektrelationale Datenbankföderationen, die als Erweiterung von SQL konzipiert ist. Diese Erweiterungen betreffen die Definition von Föderationen, die Einbeziehung von Metadaten in Anfragen, die Restrukturierung von Anfrageergebnissen sowie Mechanismen zur Konfliktbehandlung. FRAQL ist damit durchaus mit anderen Multidatenbanksprachen vergleichbar, zeichnet sich aber gegenüber Sprachen wie MSQL oder SchemaSQL durch die Erweiterbarkeit um benutzerdefinierte Funktionen und dynamisch einbeziehbare Datenquellen aus. Mit dieser Ausrichtung soll FRAQL eine Basis für weitergehende Integrations- und Fusionsaufgaben [SS99] bilden.

Unter einer Föderation verstehen wir im weiteren eine Menge von Datenbanken, die wiederum aus Relationen bestehen. Die Datenbanken können dabei durch vollwertige DBMS realisiert sein oder beim Einsatz geeigneter Adapter auch auf Web-Quellen basieren. In diesem Fall werden die Anfragemechanismen, die nicht von der Quelle unterstützt werden, durch den Adapter implementiert [SH99].

FRAQL liegt ein objektrelationales Datenmodell zugrunde: es wird die Definition von Objekttypen und davon abgeleiteten Objekttabellen unterstützt. Die Unterstützung objektrelationaler Konzepte soll einerseits die Integration von postrelationalen Datenquellen ermöglichen und andererseits mächtigere Modellierungskonzepte für die Definition des globalen Schemas bereitstellen.

Hierbei definieren *Objekttypen* die Struktur von Objekten in Form von Attributen sowie deren Wertebereich und können in einer Spezialisierungshierarchie organisiert werden. *Objekttabellen* repräsentieren die globalen Relationen der Föderation, wobei zwischen Import- und Integrationsrelationen unterschieden wird. Eine *Importrelation* ist eine Projektion über eine lokale Relation einer Datenquelle. Bei der Definition einer solchen Relation wird die Herkunft (d.h. die Quelle) sowie falls notwendig eine Abbildung zwischen den Attributen der lokalen und der globalen Relation spezifiziert.

```
create type type_name [ under type_names ] (
  attribute_definitions
);

create table global_relation_name of type_name as
  import from datasource.local_relation_name
     [ mapping_definitions ];
```

Eine Quelle wird definiert, indem der dafür benötigte Datenbankadapter sowie die Verbindungsinformationen spezifiziert werden:

```
register source data_source at 'DSN=db;UID=user;PWD=password'
  using 'adaptor_name';
```

Integrationsrelationen sind dagegen Sichten über andere globale Relationen, die durch Operatoren wie Vereinigung, θ-Verbund und äußerer Verbund kombiniert werden. Zusätzlich lassen sich die SQL-Standardoperationen wie Selektion und Projektion in den Sichtdefinitionen verwenden.

```
create table global_relation_name of type_name as
  table_expression;
```

Weiterhin unterstützt FRAQL benutzerdefinierte Funktionen (*stored functions*), die in der Datenbank der Föderierungsebene (d.h. im Anfragesystem) abgelegt sind und im Rahmen von Anfragen aufgerufen werden können. Diese Funktionen werden in Java implementiert und im Anfragesystem registriert. Je zwei benutzerdefinierte Funktionen können als zueinander invers definiert werden. Diese Beziehung zwischen Funktionen wird im Rahmen der Anfragetransformation ausgenutzt.

Der Mechanismus zur Restrukturierung von Relationen ist an das mit SchemaSQL vorgeschlagene Prinzip angelehnt. Dabei können Variablen in einer Anfrage nicht nur als Tupelvariablen an eine Relation gebunden werden sondern auch an Metadaten, wie z.B. die Menge der Attribute einer Relation oder die Menge von Relationen eines Schemas. Im Gegensatz zu SchemaSQL basiert der Zugriff auf Metadaten in FRAQL jedoch nicht auf einer Spracherweiterung, sondern es wird der Schemakatalog genutzt. So umfaßt die globale Relation `catalog.columns` Informationen zu allen Attributen der globalen Relationen und die Relation `catalog.tables` die Beschreibung aller globalen Relationen. Natürlich lassen sich auch beliebige Nutzerrelationen mit Informationen über andere Relationen einbeziehen.

In Erweiterung zu Standard-SQL können Attribute einer Tupelvariablen in Anfragen dynamisch bestimmt werden, d.h. während in SQL-Anfragen die Namen von Attributen und Relation konstant sind, können diese in FRAQL aus dem aktuellen Wert anderer Tupelattribute ermittelt werden. Diese *Variablensubstitution* wird durch die Notation `$var` beschrieben und kann in einer Anfrage überall dort eingesetzt werden, wo Tupelattribute oder Relationen anzugeben sind. So bezeichnet z.B. der Ausdruck `tbl1.$(tbl2.col)` ein Attribut des aktuellen Tupels der Relation `tbl1`, dessen Bezeichner sich aus dem Wert von `tbl2.col` ergibt. In gleicher Weise kann die Relation im `from`-Teil einer Anfrage dynamisch bestimmt werden:

```
select t2.isbn, t2.titel
from catalog.tables t1, $(t1.name) t2
where t1.type_name = 'buch';
```

Mit dieser Technik ist eine flexible Transformation von Schemata durch Sichtenbildung möglich. Zu berücksichtigen ist aber, daß dies Einfluß auf die Optimierung von Anfragen hat. So sind statische Optimierungstechniken, die streng zwischen Aufstellen des Anfrageplans und der eigentlichen Ausführung trennen, nur bedingt geeignet.

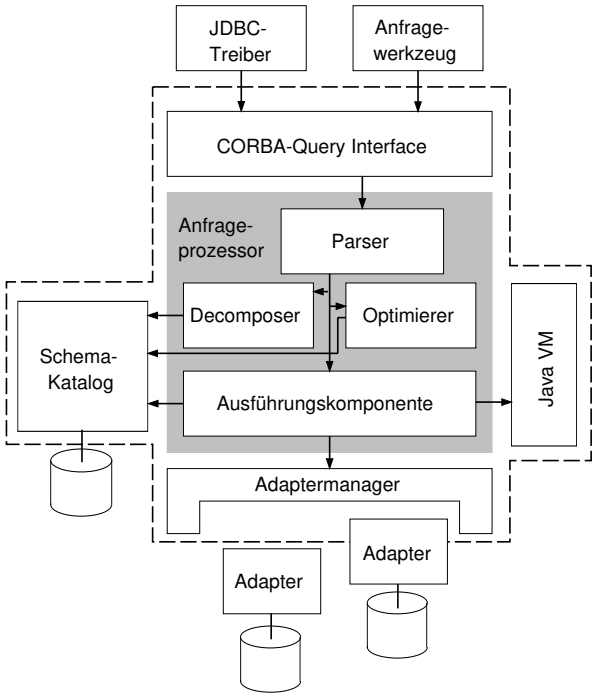

Abbildung 1: Architektur des FRAQL-Anfragesystems

FRAQL ist im Rahmen eines Anfragesystems für Datenbankföderationen implementiert. Die Architektur dieses Systems ist in Abb. 1 dargestellt. Die wesentlichen Komponenten sind der Anfrageprozessor bestehend aus Decomposer, Optimierer und Ausführungskomponente, die Java Virtual Machine zur Ausführung der benutzerdefinierten Funktionen, der Schemakatalog sowie die Adapterebene. Die Adapter implementieren eine einheitliche Zugriffsschnittstelle zu den Datenquellen und helfen damit bei der Überwindung der Systemheterogenität. Die Schnittstellen zu den Adaptern sowie zum Anfragesystem selbst sind mittels CORBA realisiert. Somit können Adapter dynamisch bei Bedarf nachgeladen werden. Auf der Basis der Anfrageschnittstelle sind ein JDBC-Treiber sowie ein interaktives Anfragewerkzeug implementiert.

Der Prozeß der Anfragebearbeitung basiert auf dem in Abb. 2 dargestellten Ablauf [YM98]. Eine Anfrage wird vom Parser in einen Anfragebaum überführt und anhand der Definition der globalen Relationen in Teilanfragen zerlegt. Vor dieser Zerlegung ist noch ein Optimierungsschritt angeordnet, der aber in der gegenwärtigen Implementierung nur einige wenige einfache Heuristiken realisiert. Die Teilanfragen werden an die Adapter zur Bearbeitung weitergeleitet und die Ergebnisse entsprechend des Anfragebaumes zusammengesetzt.

5 Konfliktbehandlung in FRAQL

Ein integriertes Schema wird in FRAQL ausschließlich durch die globalen Relationen in Form von Sichten über lokalen Relationen definiert. Die Integration von Daten und damit verbunden die Behandlung von Konflikten sind demzufolge Teil der Anfragebearbeitung und hier insbesondere der Sichtauflösung, der Anfragetransformation sowie die Komposition der lokalen Ergebnisdaten. Die wesentlichen Ansatzpunkte zur Konfliktauflösung in FRAQL sind daher:

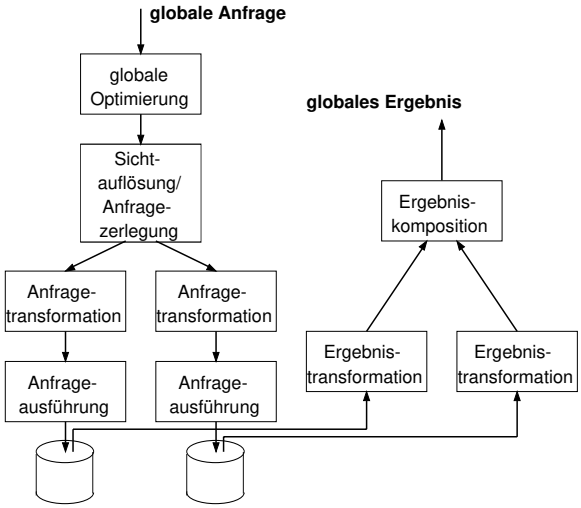

Abbildung 2: Prinzip der Anfragebearbeitung

- die Umbenennung von Attributen sowie die Anwendung benutzerdefinierter Funktionen,
- die Integrationsoperationen,
- die Verbindung von Daten und Metadaten.

Im folgenden wollen wir den Einsatz dieser Techniken zur Auflösung der in Abschnitt 2 diskutierten Konflikte vorstellen.

Die einfachste Form von Konflikten sind Beschreibungskonflikte, die immer dann auftreten, wenn gleiche Real-Welt-Objekte mit unterschiedlichen Eigenschaften modelliert werden. Konflikte dieser Art werden in FRAQL bei der Definition von Import-Relationen aufgelöst. Den Ausgangspunkt bilden dabei die globalen Objekttypen, die die gewünschten globalen Eigenschaften der Objekte festlegen. Für eine Import-Relation kann nun angegeben werden, wie die lokale Relation den Objekttyp erfüllt. Dabei gelten folgende Konventionen:

1. Alle Attribute der lokalen Relation, für die ein korrespondierendes Attribut (d.h. mit gleicher Bezeichnung und gleichem Typ) durch den globalen Objekttyp definiert ist, werden in die globale Relation übernommen und sind dort sichtbar.

2. Mit der Notation *global_name* `is` *local_name* wird ein mit *local_name* bezeichnetes Attribut unter dem Namen *global_name* in die globale Relation aufgenommen. Voraussetzung dafür ist die Kompatibilität der Attributtypen.

3. durch die Notation *global_name* `is` *func(local_name)* erfolgt unter Anwendung der benutzerdefinierten Funktion *func* eine Konvertierung des Wertes des lokalen Attributes. Die Funktion muß als Argument einen Wert entsprechend des Typs von *local_name* erwarten und ein Ergebnis vom Typ des globalen Attributs *global_name* liefern. Weiterhin muß zu dieser Funktion eine inverse Funktion definiert sein.

4. Alle weiteren Attribute der lokalen Relation werden ausgeblendet.

5. Alle Attribute der globalen Relation, für die keine Abbildung definiert wurde, werden mit `NULL` belegt.

Das folgende Beispiel illustriert die Anwendung dieser Konzepte. Zunächst wird ein Objekttyp `buch` definiert:

```
create type buch (
  isbn varchar(20),
  titel varchar(100),
  preis float
);
```

Weiter sei eine lokale Relation `book` in der Quelle `src` angenommen, die sich vom globalen Typ durch die Bezeichnung der Attribute (englisch statt deutsch) und die verwendete Währung für den Buchpreis (Euro statt DM) unterscheidet. Demzufolge wird eine einfache Konvertierungsfunktion `euro2dm` benötigt, die zuvor gemeinsam mit der inversen Funktion `dm2euro` registiert werden muß. Schließlich kann die globale Relation `de_buch` wie folgt definiert werden:

```
create table de_buch of buch as import from src.book (
  titel is title,
  preis is euro2dm (price)
);
```

Auf der Basis dieser Definitionen kann nun die globale Anfrage

```
select titel, isbn from de_buch where preis < 100.0;
```

in eine lokale Anfrage transformiert werden. Neben der Umbenennung der Attribute ist dabei auch der Ausdruck in der Selektionsbedingung zu transformieren. Da die benutzerdefinierte Funktion nur auf globaler Ebene verfügbar ist, muß die Selektion entweder vollständig dort ausgeführt werden oder es kann für konstante Ausdrücke eine Vorabberechnung unter Anwendung der inversen Funktion erfolgen. Voraussetzung hierfür ist jedoch, daß die Konvertierungsfunktion ordnungserhaltend und monoton ist. Andernfalls ist die Ersetzung nicht möglich und die Selektion muß auf globaler Ebene ausgeführt werden. Hier sind noch Erweiterungen denkbar, die eine exakte Spezifikation der Funktion ermöglichen. Für das obige Beispiel lautet die transformierte lokale Anfrage nach Auswertung des Ausdrucks `dm2euro (100)`:

```
select title, isbn from book where price < 51.13;
```

Semantische Konflikte treten bei der Integration immer dann auf, wenn sich die zu integrierenden Relationen überlappen. Wir können dabei zwischen horizontaler und vertikaler Überlappung unterscheiden: horizontale Überlappung besteht, wenn sich die Extensionen der Relationen überlappen, d.h. Tupel aus beiden Relationen einem Real-Welt-Objekt entsprechen. Vertikale Überlappung liegt dagegen vor, wenn die Relationen verschiedene Aspekte einer globalen Relation repräsentieren, aber dabei dennoch semantisch gleiche Attribute besitzen.

Zur Behandlung dieser Konflikte bietet FRAQL zunächst die bekannten Operationen Vereinigung, Verbund sowie äußerer Verbund, die bei der Definition der Integrationsrelationen angewendet werden können. Mit dem Vereinigungsoperator lassen sich horizontal überlappende Extensionen zusammenfassen, die Verbundoperationen helfen dagegen bei vertikaler Überlappung. In diesem Zusammenhang sind aber zwei grundsätzliche Problemstellungen zu lösen:

1. die Frage, wann zwei Tupel aus verschiedenen Relationen das gleiche Real-Welt-Objekt repräsentieren:
   Dies kann z.B. anhand der Objektidentität bzw. des Primärschlüssels oder durch wertmäßigen Vergleich der Attribute entschieden werden.

2. das Problem der Datenkonflikte, d.h. wie zu verfahren ist, wenn die zu integrierenden Tupel, die einem Real-Welt-Objekt entsprechen, unterschiedliche Attributwerte umfassen:
   Mögliche Auflösungsstrategien sind hier die Festlegung einer Vorzugsrelation, die den zu übernehmenden Attributwert bestimmt oder auch die Aggregation bzw. Mittelwertbildung.

Das Problem der Tupelidentität wird in FRAQL durch eine benutzerdefinierte Vergleichsbedingung bei den Vereinigungs- und Verbundoperationen gelöst. Damit wird bestimmt, welche Attribute für den Vergleich herangezogen werden, so daß Tupel, die das gleiche Real-Welt-Objekt repräsentieren, auch dann identifiziert werden, wenn sich einzelne Attributwerte unterscheiden (Abb. 3):

```
create table buecher of buch as
s1.buch union s2.buch
on s1.buch.isbn = s2.buch.isbn;
```

isbn	*autor*	*titel*
382578	Williams, T.	Otherland
326523	Gibson, W.	Idoru

(a) Relation *s1.buch*

isbn	*autor*	*titel*
382578	Tad Williams	Otherland
276830	St. Lem	Solaris

(b) Relation *s2.buch*

Abbildung 3: Tupelidentität bei Datenkonflikten

Zur Behandlung von Datenkonflikten werden in FRAQL benutzerdefinierte Auflösungsfunktionen eingesetzt. Eine solche Funktion wird immer dann aufgerufen, wenn bei der Integration der Relationen (durch Vereinigung oder Verbund) gleiche Tupel auftreten, wobei sich die Gleichheit hier wieder auf die Erfüllung der Vergleichs- bzw. Verbundbedingung bezieht. Die betroffenen Tupel werden der Funktion als Argumente übergeben; das Ergebnis des Funktionsaufrufs ist wieder ein Tupel, das schließlich in die globale Relation aufgenommen wird. Im Beispiel in Abb. 4 werden zwei Relationen `fb_buch` und `ub_buch` integriert, die die Bücher der Fakultäts- bzw. der Universitätsbibliothek beinhalten und deren Extensionen sich überlappen. Beide Relationen besitzen ein Attribut `bestand`, das in der globalen Relation als Summe beider Teilbestände abzubilden ist.

isbn	*bestand*
332456124	2
671078467	2

(a) Relation *fb_buch*

isbn	*bestand*
332456124	1
126784394	2

(b) Relation *ub_buch*

isbn	*bestand*
332456124	3
126784394	2
671078467	2

(c) Globale Relation *bib_buch*

Abbildung 4: Auflösung von Datenkonflikten

Die globale Relation `bib_buch` kann danach wie folgt definiert werden:

```
create table bib_buch of buch as
fb_buch union ub_buch
 on fb_buch.isbn = ub_buch.isbn
 reconciled by resolve;
```

isbn	*titel*	*typ*
38266051	Datenbanken	buch
01706012	Informatik-Spektrum	journal

(a) Relation *s1.publikation*

isbn	*titel*
38266051	Datenbanken
36155690	Query Processing

(b) Relation *s2.buch*

issn	*titel*
01706012	Informatik-Spektrum
00010782	CACM

(c) Relation *s2.journal*

Abbildung 5: Meta-Konflikte: Relation vs. Attribut

Die Auflösungsfunktion `resolve` wird zuvor als statische Java-Methode implementiert und als benutzerdefinierte Funktion registriert. Die Klasse `Tuple` ist dabei eine vordefinierte Klasse des FRAQL-Laufzeitsystems.

```
// Java
public static Tuple resolver (Tuple t1, Tuple t2) {
  Tuple result = new Tuple ();
  result.setInt (1, t1.getInt (1));
  result.setInt (2, t1.getInt (2) + t2.getInt (2));
  return result;
}

// FraQL
create function resolve (t1 tuple, t2 tuple)
returns tuple
external name 'Bibo.resolver';
```

Wenn Sachverhalte der realen Welt durch verschiedene Modellierungskonzepte abgebildet werden, dann führt dies zu strukturellen Konflikten. Je nach der Reichhaltigkeit der verfügbaren Konzepte des Datenmodells können dabei eine Vielzahl von Problemen auftreten. Wir wollen an dieser Stelle nur einen wichtigen Konfliktfall herausgreifen – die Meta-Konflikte. Weitere Konflikte und deren Auflösung speziell für das relationale Datenmodell werden u.a. in [KCGS95] beschrieben.

Metadaten-Konflikte bestehen, wenn ein Sachverhalt in einer Datenbank als Datum repräsentiert ist, während er in einer anderen Datenbank als Teil des Schemas modelliert wird. Die folgenden beiden Beispiele sollen dies verdeutlichen und die Auflösung dieser Konflikte in FRAQL zeigen.

Im ersten Beispiel (Abb. 5) werden einer Datenbank Bücher und Zeitschriften in einer Relation `publikation` gespeichert, wobei zur Unterscheidung ein Attribut `typ` dient, das die Werte `"buch"` und `"journal"` annehmen kann. In der zweiten Datenbank werden dafür zwei getrennte Relationen `buch` und `journal` verwendet. Für die Integration sollen nun die Relationen der zweiten Datenbank an die Struktur der Relation `publikation` angepaßt werden. Die einfachste Form ist ein Verbund beider Relationen, wobei das Typ-Attribut durch ein Literal belegt wird:

```
create table publikationen of publ_typ as
select isbn, titel, "buch" from s2.buch
  union
select issn, titel, "journal" from s2.journal;
```

isbn	*signatur*	*typ*
38934237	99a1802:1	freihand
38934237	99a1802:2	archiv

(a) Relation*s1.buch*

isbn	*freihand*	*archiv*
38934237	99a1802:1	99a1802:2
22660513	97a2342:1	96a2053:1

(b) Relation *s2.buch*

Abbildung 6: Meta-Konflikte: Attribut vs. Wert

Eine flexiblere Lösung ergibt sich, wenn die Namen der Relationen aus dem Schemakatalog ermittelt werden und die Möglichkeit der Variablensubstitution von FRAQL genutzt wird:

```
create table publikationen of publ_typ as
select t2.isbn, t2.titel, t1.name
from catalog.tables t1, $(t1.name) t2
where t1.schema = 's2';
```

Auf diese Weise muß die Definition der globalen Relation nicht geändert werden, wenn neue Publikationstypen (wie z.B. eine Relation `preprint`) eingefügt werden.

Im zweiten Beispiel (Abb. 6) werden in einer Relation `buch` einer Datenbank Freihand- und Archivexemplare von Büchern als separate Tupel gespeichert, wobei zur Unterscheidung wieder ein `typ`-Feld dient. In der zweiten Datenbank ist die Relation `buch` dagegen so definiert, daß diese Exemplare jeweils durch ein eigenes Attribut des Tupels repräsentiert sind.

Durch Zugriff auf die Attributnamen und die Substitution der Variablen kann die Relation der zweiten Datenbank entsprechend der ersten Relation umstrukturiert werden.

```
select b.isbn, b.$(ba.name), ba.name
from s2.buch b, catalog.attributes ba
where ba.table = 'buch' and ba.name <> 'isbn';
```

In Abschnitt 2 haben wir als weitere Konfliktart die Heterogenitätskonflikte genannt, die durch Verwendung unterschiedlicher Datenmodelle in den einzelnen Datenquellen auftreten. Konflikte dieser Art werden in FRAQL weitgehend in den Adaptern aufgelöst, die eine Abbildung der verschiedenen Modellierungskonstrukte vornehmen sowie Systemheterogenitäten, wie SQL-Dialekte oder verschiedene Datentypen, verbergen.

6 Zusammenfassung und Ausblick

Die Behandlung von Konflikten ist ein wichtiger Teilschritt bei der Integration heterogener Datenbestände. Dabei sind Unterschiede auf Schemaebene ebenso zu überwinden wie Probleme, die sich durch verschiedene Datenrepräsentationen eines Real-Welt-Objektes ergeben. Ausgehend von dieser Problemstellung wurden in diesem Beitrag Techniken diskutiert, die im Rahmen der Anfragesprache FRAQL bei der Lösung dieser Aufgaben helfen können. Die Konfliktbehandlung erfolgt auf globaler Ebene bei der Definition von Sichten auf die Relationen der einzelnen Datenquellen. Als wichtigste Mechanismen stehen dabei neben den von SQL bekannten Operationen auch Umbenennung, Werttransformationen sowie strukturelle Transformationen zur Verfügung. Mit diesen wenigen zusätzlichen Mitteln lassen sich bereits eine Vielzahl von Konflikten lösen.

FRAQL ist im Rahmen eines Anfragesystems für Datenbankföderationen in C++ implementiert und unterstützt gegenwärtig den im vorliegenden Beitrag eingeführten Sprachumfang. Adapter liegen bislang

für das Oracle-DBMS vor; die Anbindung (strukturierter) Web-Quellen ist ebenfalls möglich [SH99]. Neben dem bereits erwähnten JDBC-Treiber und dem Anfragetool ist auch eine graphische Entwurfsumgebung in Arbeit, die eine visuelle, datengestützte Definition des integrierten Schemas ermöglichen soll. Weitere Arbeiten auf diesem Gebiet sollen sich auf die dynamische Optimierung der Anfragebearbeitung durch die Verschachtelung von Anfrageplanung und -ausführung konzentrieren.

Literatur

[ASD+91] R. Ahmed, P. De Smedt, W. Du, W. Kent, M.A. Ketabchi, W. Litwin, A. Rafii und M.-C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19–27, December 1991.

[BLN86] C. Batini, M. Lenzerini und S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, Dezember 1986.

[CHS+95] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas II, J.H. Williams und E.L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In Omran A. Bukhres, M. Tamer Özsu und Ming-Chien Shan, Herausgeber, *Proc. RIDE-DOM '95, 5th Int. Workshop on Research Issues in Data Engineering - Distributed Object Management, Taipei, Taiwan*, S. 124–131. IEEE-CS, 1995.

[Con97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, Berlin/Heidelberg, 1997.

[GGF+96] G. Gardarin, S. Gannouni, B. Finance, P. Fankhauser, W. Klas, D. Pastre, R. Legoff und A. Ramfos. IRO-DB — A Distributed System Federating Object and Relational Databases. In O. A. Bukhres und A. K. Elmagarmid, Herausgeber, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, Kapitel 20, S. 684–712. Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[GLRS93] J. Grant, W. Litwin, N. Roussopoulos und T. Sellis. Query Languages for Relational Multidatabases. *VLDB Journal*, 2(2):153–171, 1993.

[GPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos und J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, März/April 1997.

[Has99] W. Hasselbring. Top-Down vs. Bottom-Up Engineering of Federated Information Systems. In S. Conrad, W. Hasselbring und G. Saake, Herausgeber, *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99, Kühlungsborn, Germany, May 5–7, 1999*, S. 131–138. infix-Verlag, Sankt Augustin, 1999.

[HE99] U. Hohenstein und A. Ebert. An Integrated Toolkit for Building Database Federations. In S. Conrad, W. Hasselbring und G. Saake, Herausgeber, *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99, Kühlungsborn, Germany, May 5–7, 1999*, S. 43–60. infix-Verlag, Sankt Augustin, 1999.

[IFF+99] Z.G. Ives, D. Florescu, M. Friedman, A. Levy und D.S. Weld. An Adaptive Query Execution System for Data Integration. In A. Delis, C. Faloutsos und S. Ghandeharizadeh, Herausgeber, *SIGMOD 1999, Proceedings of the ACM SIGMOD*, S. 299–310, 1999.

[KCGS95] W. Kim, I. Choi, S. Gala und M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, Herausgeber, *Modern Database Systems*, Kapitel 26, S. 521–550. ACM Press, New York, NJ, 1995.

[KGK+95] W. Kelley, S. Gala, W. Kim, T. Reyes und B. Graham. Schema Architecture of the UniSQL/M Multidatabase System. In W. Kim, Herausgeber, *Modern Database Systems*, Kapitel 30, S. 621–648. ACM Press, New York, NJ, 1995.

[KS91] W. Kim und J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, Dezember 1991.

[LCC99] E.-P. Lim, R.H.L. Chiang und Y. Cao. Tuple source relational model: A source-aware data model for multidatabases. *Data & Knowledge Engineering*, 29(1):83–114, January 1999.

[LRO96] A.Y. Levy, A. Rajaraman und J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan und N.L. Sarda, Herausgeber, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, S. 251–262. Morgan Kaufmann, 1996.

[LSS96] L.V.S. Lakshmanan, F. Sadri und I.N. Subramanian. SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. In T. M. Vijayaraman, A.P. Buchmann, C. Mohan und N.L. Sarda, Herausgeber, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, S. 239–250. Morgan Kaufmann, 1996.

[PAGM96] Y. Papakonstantinou, S. Abiteboul und H. Garcia-Molina. Object Fusion in Mediator Systems. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan und N.L. Sarda, Herausgeber, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, S. 413–424. Morgan Kaufmann, 1996.

[SCG93] F. Saltor, M. Castellanos und M. Garcia-Solaco. Overcoming Schematic Discreprancies in Interoperable Databases. In D. K. Hsiao, E. J. Neuhold und R. Sacks-Davis, Herausgeber, *Interoperable Database Systems, Proc. of the IFIP WG 2.6 Database Semantics Conf., DS-5, Lorne, Victoria, Australia, November, 1992*, S. 191–205, Amsterdam, 1993. North-Holland.

[SH99] K.-U. Sattler und M. Höding. Adapter Generation for Extraction and Querying Data from Web Sources. In *Proc. of 2nd ACM SIGMOD Workshop WebDB'99*, 1999.

[SL90] A. P. Sheth und J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SPD92] S. Spaccapietra, C. Parent und Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, Juli 1992.

[SS99] K.-U. Sattler und G. Saake. Supporting Information Fusion with Federated Database Technologies. In S. Conrad, W. Hasselbring und G. Saake, Herausgeber, *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99, Kühlungsborn, Germany, May 5–7, 1999*, S. 179–184. infix-Verlag, Sankt Augustin, 1999.

[SSR94] E. Sciore, M. Siegel und A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254–290, Juni 1994.

[VZ98] S. Venkataraman und T. Zhang. Heterogeneous Database Query Optimization in DB2 Universal DataJoiner. In A. Gupta, O. Shmueli und J. Widom, Herausgeber, *VLDB'98, Proceedings of 24rd Int. Conf. on Very Large Data Bases, 1998, New York City, New York, USA*, S. 685–689. Morgan Kaufmann, 1998.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, März 1992.

[YM98] C. T. Yu und W. Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.