Konzeptbasierte Anfrageverarbeitung in Mediatorsystemen*

Kai-Uwe Sattler[1,2] Ingolf Geist[1] Rainer Habrecht[1] Eike Schallehn[1]

[1]Fakultät Informatik, Universität Magdeburg
[2]Institut für Informatik, Universität Halle
{kus|geist|habrecht|eike}@iti.cs.uni-magdeburg.de

Abstract: Ein Weg zur Überwindung der Heterogenität bei der Datenintegration in Mediatorsystemen ist die Nutzung semantischer Metadaten in Form eines Vokabulars oder einer Ontologie zur expliziten Modellierung des Hintergrundwissens. Dementsprechend muss die verwendete Anfragesprache diese Metaebene in die Anfrageformulierung und -auswertung einbeziehen. In diesem Beitrag wird hierzu ein Mediator für kulturhistorische Datenbanken vorgestellt, der auf einem konzeptbasierten Integrationsmodell basiert und die Anfragesprache CQuery – eine XQuery-Erweiterung – implementiert. Weiterhin werden ausgehend von der Semantikdefinition der Anfrageoperationen die Phasen des Rewriting, der Ausführung sowie die Nutzung eines Anfrage-Caches beschrieben.

1 Einführung

Der integrierte Zugriff auf heterogene Datenbestände stellt gerade im World Wide Web eine große Herausforderung dar. Probleme wie hohe Autonomie und Heterogenität der Quellen oder auch Skalierbarkeit und Evolutionsfähigkeit hinsichtlich einer großen Anzahl teilweise noch wechselnder Quellen treten hier im besonderen Maße zutage. Die Bandbreite möglicher Lösungsansätze reicht dabei von einfachen (Meta-)Suchmaschinen über materialisierende Ansätze bis hin zu Mediatorsystemen, die Anfragen auf einem globalen Schema in Teilanfragen zerlegen, an die Quellsysteme zur Bearbeitung weiterleiten und die Ergebnisse zusammenführen.

Mediatorsysteme lassen sich grob danach unterscheiden, wie die Korrespondenzen zwischen den lokalen Schemata der Quellen und dem globalen Mediatorschema gestaltet sind. Beim Global-as-View-Ansatz (GaV) wird das globale Schema als Sicht über den lokalen Schemata definiert. Dagegen wird beim Local-as-View-Ansatz (LaV) von einem globalen Schema ausgegangen, über das schließlich die lokalen Schemata als Sichten definiert werden, die somit nur Ausschnitte sowohl bezüglich des Schemas als auch der Extensionen beinhalten. Der GaV-Ansatz hat den Vorteil einer einfacheren Anfrageverarbeitung, beim LaV-Prinzip ist das Hinzufügen bzw. Entfernen von Quellen deutlich einfacher, da hier keine Korrespondenzen zwischen Quellen berücksichtigt werden müssen [Hal01].

*Diese Arbeit wurde teilweise gefördert von der DFG (FOR 345/1) sowie der Koordinierungsstelle für Kulturgutverluste.

Bei Mediatorsystemen der ersten Generation erfolgt die Integration im Wesentlichen auf struktureller Ebene. Die Daten aus den einzelnen Quellen werden dabei anhand struktureller Korrespondenzen wie die Zugehörigkeit zu gleich strukturierten Klassen oder die Existenz gemeinsamer Attribute kombiniert. Dies funktioniert am besten bei relativ homogenen Strukturen der beteiligten Quellen. In Szenarien mit eher disjunkten Domänen führt dies jedoch zu einer Vielzahl globaler Klassen, was wiederum detailliertes Hintergrundwissen zur Formulierung der daraus resultierenden komplexeren Anfragen erfordert.

Ein Ausweg ist die explizite Modellierung dieses Hintergrundwissen in Form semantischer Metadaten, d.h. als Vokabular, Begriffs- bzw. Konzepthierarchie oder gar Ontologie, und deren Nutzung zur Anfrageverarbeitung und Datenintegration. Ähnliche Bemühungen gibt es im Rahmen des Semantic Web, wo die (wissensbasierte) Verarbeitung von Web-Dokumenten durch das Hinzufügen einer semantischen Ebene mit Metadaten erleichtert werden soll. Erste Ergebnisse dieser Arbeiten sind Modelle und Sprachen für Vokabulare und Ontologien, wie z.B. RDF Schema (RDFS) und DAML+OIL, sowie damit verbundene Technologien. Da hierbei grundsätzliche sehr ähnliche Problemstellungen bestehen, liegt eine Verbindung von Semantic Web und Datenbank-/Mediatortechnologien nahe. Als besondere Anforderung aus Sicht der Datenintegration ergibt sich jedoch die Notwendigkeit, den Bezug zu den Quelldaten herzustellen, indem beschrieben wird, wie die Quellsysteme ein gegebenes Konzept der semantischen Ebene strukturell und inhaltlich realisieren. Diese Informationen müssen beim Registrieren einer Quelle bereitgestellt und als Teil der Anfragetransformation und -zerlegung ausgewertet werden.

Vor diesem Hintergrund wird im Folgenden der YACOB-Mediator vorgestellt, der zur Formulierung und Auswertung von Anfragen über heterogenen Web-Quellen explizit modelliertes Domänenwissen in Form von Konzepten und deren Beziehungen nutzt. Dieses System wurde für die integrierte Suche in kulturthistorischen Datenbanken entwickelt, die Informationen über kriegsbedingt verlorengegangene Kulturgüter verwalten.

Der Beitrag dieses Ansatzes liegt in *(i)* der Definition eines auf RDFS basierenden Metamodells, das die Repräsentation von Konzepten und Beziehungen als begriffliche „Anker“ für die Integration mit den Informationen zur Abbildung auf die Quelldaten verbindet und dabei dem Local-as-View-Prinzip folgt, *(ii)* der Anfragesprache CQuery, welche die Formulierung von Anfragen sowohl über die Konzept- als auch die Instanzebene unterstützt und *(iii)* der Darstellung von Operationen und Strategien zur Transformation, Zerlegung und Ausführung globaler CQuery-Anfragen inklusive eines semantischen Caching.

2 Ein konzeptbasiertes Modell zur Datenintegration

Zur Repräsentation der zu integrierenden Daten und der ihnen zuzuordnenden semantischen Metadaten sind zwei Ebenen zu berücksichtigen: *(i)* die eigentliche Daten- oder Instanzebene, welche die in den Quellsystemen gespeicherten Daten umfasst sowie *(ii)* die Meta- oder Konzeptebene zur Beschreibung der Semantik der Daten und deren Beziehungen. Als Datenmodell für die Instanzebene (im Folgenden nur als „Datenmodell“ bezeichnet) wird in unserem Ansatz ein einfaches semistrukturiertes Modell auf Basis von

XML eingesetzt. Im Mediator und im Austausch mit den Quellsystemen bzw. deren Wrappern werden Daten somit konsequent in XML repräsentiert. Dementsprechend wird davon ausgegangen, dass ein Quellsystem Daten in XML exportiert und als Anfragen einfache XPath-Ausdrücke verarbeiten kann. Eine gegebenenfalls notwendige Transformation von Anfragen beispielsweise nach SQL wird dadurch vereinfacht, dass nur eine eingeschränkte Untermenge von XPath verwendet wird, die u.a. keine Funktionsaufrufe enthält. Für den Export kann eine Quelle prinzipiell XML-Daten mit einer beliebigen DTD liefern, da die notwendigen Transformationen automatisch durchgeführt werden können.

Das Modell für die Beschreibung der Konzeptebene (im Folgenden „Konzeptmodell“ genannt) basiert auf RDF Schema. Das Resource Description Framework (RDF) – vom W3C als Mechanismus zur Erzeugung und zum Austausch von Metadaten für Web-Dokumente entwickelt – ist ein einfaches graphbasiertes Modell, dessen Knoten Ressourcen oder Literalen und dessen Kanten Eigenschaften entsprechen. RDF Schema (RDFS) definiert darauf aufbauend weitergehende Primitive wie Klassen und Klassenbeziehungen. Obwohl eine grundsätzliche Verwandtschaft mit klassischen Datenbankmodellen besteht, weist RDFS dennoch einige Besonderheiten auf, wie z.B. dass Properties (Eigenschaften) unabhängig von Klassen definiert und nur über Domain Constraints bezüglich ihrer Anwendbarkeit auf bestimmte Klassen eingeschränkt werden.

Mit Modellierungsprimitiven wie *Class*, *Property* oder *subClassOf* ermöglicht RDFS die Beschreibung einfacher Vokabulare oder Ontologien. Ein Beispiel aus dem hier betrachteten Anwendungsbereich ist in Abb. 1 dargestellt. Ein Konzept im Mediatormodell entspricht somit einer Klasse in RDFS (genauer: ein Konzept wird als Subklasse einer speziellen, hier nicht dargestellten Klasse *Concept* definiert), Konzepteigenschaften werden als Properties abgebildet. Beziehungen zwischen Konzepten, die über die bereits in RDFS definierten Beziehungen wie *subClassOf* hinausgehen, werden ebenfalls als Property modelliert, wobei als Werte- und Bildbereich solcher Properties nur Klassen zugelassen sind.

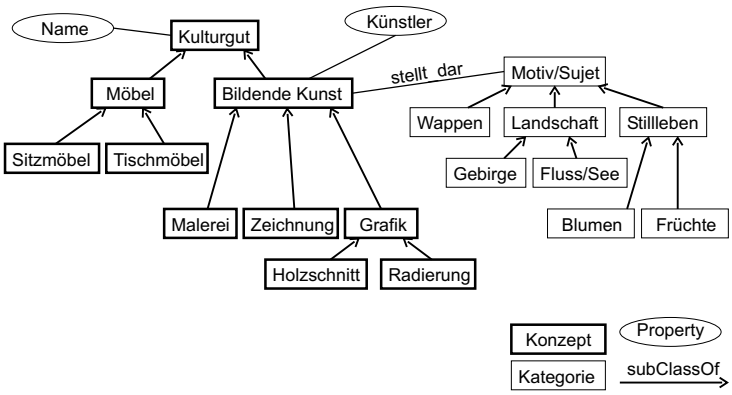

Abbildung 1: Konzepthierarchie

Die Konzeptebene übernimmt somit die Rolle eines globalen Schemas des Mediators, erlaubt aber darüber hinaus auch noch weitergehende semantische Modellierungen. So las-

sen sich nicht nur Schemainformationen in Form von Klassen bzw. Konzepten sondern auch Ausprägungen in Form von Werten darstellen. Hierzu werden Kategorien als eine spezielle Variante einer RDFS-Klasse eingeführt, die im Gegensatz zu Konzepten keine Extensionen aufweist. Eine Kategorie kann als Begriff angesehen werden, der in verschiedenen Quellen durch unterschiedliche Werte repräsentiert wird. Kategorien können unter Verwendung der *subClassOf*-Beziehung in Hierarchien organisiert werden.

Ein Beispiel für den Einsatz von Kategorien ist das Property *stellt_dar* zum Konzept „*Bildende Kunst*“ in Abb. 1. Der Wertebereich dieses Properties ist durch die Kategorie „*Motiv/Sujet*“ definiert, das wiederum verschiedene Unterkategorien besitzt. In einer Quelle könnte eine dieser Kategorien bspw. durch ein Literal „still life“ als Wert des das Property *stellt_dar* repräsentierende Element bzw. Attribut dargestellt werden, in einer zweiten Quelle durch eine beliebige andere Kodierung etc. Auf diese Weise lassen sich Informationen explizit machen, die anderenfalls in den Daten verborgen sind.

Formal lassen sich Konzept- und Datenmodell wie folgt beschreiben. Seien *URI* die Menge der Uniform Resource Identifier, *Name* eine Menge gültiger Bezeichner und $\mathcal{L}$ die Menge aller Literale. Die Menge der Klassen wird bezeichnet mit $\mathcal{T} = \mathit{URI} \times \mathit{Name}$ bestehend aus einer eindeutigen URI und einem Klassennamen. Klassen können weiter unterschieden werden in

- Konzepte (bezeichnet mit $\mathcal{C} \subset \mathcal{T}$) als Klassen zu denen Extensionen in den Quellsystemen existieren
- und Kategorien (bezeichnet mit $\mathcal{V} \subset \mathcal{T}, \mathcal{V} \cap \mathcal{C} = \emptyset$), die als abstrakte Eigenschaftwerte zur semantischen Gruppierung von Objekten verwendet werden können, jedoch keine Extension aufweisen.

Zu den Konzepten können Properties $\mathcal{P}$ definiert werden mit $\mathcal{P} = \mathit{Name} \times \mathcal{C} \times \{\mathcal{T} \cup \mathcal{L}\}$, die jeweils aus einem Bezeichner, einem Konzept, dem sie zugeordnet sind, und einer Klasse als Wertebereich bzw. einem Literal als Ausprägung bestehen. Ferner sei eine Spezialisierungsbeziehung **is_a** $\subseteq \mathcal{T} \times \mathcal{T}$ gegeben, wobei Spezialisierungshierarchien von Konzepten und Kategorien immer disjunkt sein müssen.

Das Datenmodell kann in Anlehnung an OEM [GMPQ$^+$97] beschrieben werden. Die Menge aller Objekte sei als $\mathcal{O} = \mathit{ID} \times \mathit{Name} \times \{\mathcal{L} \cup \mathit{ID} \cup \mathbb{P}\mathit{ID}\}$ definiert, wobei für ein Objekt $(\mathit{id}, \mathit{name}, v) \in \mathcal{O}$ *id* eine eindeutige Objekt-ID, *name* den Elementname und *v* den Wert des Objektes bezeichnen. *v* kann dabei ein atomarer Wert, eine Objekt-ID oder eine Menge von Objekt-IDs sein.

Die Extension **ext** : $\mathcal{C} \rightarrow \mathcal{O}$ eines Konzeptes $c = (\mathit{uri}, \mathit{name})$ umfasst eine Menge von Objekten, deren Elementname gleich dem Konzeptnamen ist und die Objekte, auf die im Rahmen der Menge *v* verwiesen wird, den zu dem Konzept definierten Properties entsprechen:

$$\mathbf{ext}(c) = \{o = (\mathit{id}, \mathit{name}, v) \mid \mathit{name} = c.\mathit{name} \wedge \forall (p, c, c') \in \mathcal{P} : \exists w \in v : w.\mathit{name} = p\}$$

Eine weitere wesentliche Komponente des Mediatormodells ist die Beschreibung der Abbildung auf die Schemata der Quellen, d.h. konkret die Angaben, wie eine Quelle Daten

zu einem gegebenen Konzept liefert und wie deren Struktur auf die durch das Konzept und dessen Eigenschaften definierte globale Struktur abgebildet wird. Hierbei werden *(i)* entsprechend der RDFS-Modellierung Konzepte und Eigenschaften getrennt betrachtet und *(ii)* das LaV-Prinzip verfolgt, d.h. ein Quellenschema wird bezüglich des globalen Schemas (hier des Konzeptschemas) definiert.

Als Konsequenz werden eine Konzept-Mapping- und eine Property-Mapping-Vorschrift benötigt, die wiederum als RDFS-Klassen realisiert sind. Eine Instanz eines Konzept-Mappings legt fest, wie das zugeordnete Konzept von einem Quellsystem unterstützt wird. Hier können zwei Fälle unterschieden werden:

(a) ein Quellsystem liefert Instanzen, die alle durch das globale Konzept definierten Eigenschaften (Attribute) unterstützen,

(b) ein Quellsystem liefert nur partielle Informationen, d.h. die Instanzen müssen durch Daten (Attributwerte) aus anderen Quellen vervollständigt werden.

Für beide Fälle umfasst ein Konzept-Mapping *CM* die Komponenten Quellenname, lokaler Elementname sowie optional ein Filterprädikat: *CM = (Source, LName, FilterPredicate)*. Über den Quellenname kann die Quelle identifiziert werden, wenn Instanzen zu diesem Konzept angefragt werden. Der lokale Elementname bezeichnet das XML-Element, dass in der Quelle zur Repräsentation von Instanzen dieses Konzeptes verwendet wird. Mit dem Filterprädikat in Form eines XPath-Ausdrucks kann die Instanzmenge eingeschränkt werden. Dies ist beispielsweise notwendig, wenn im Quellsystem verschiedene Konzepte durch die gleiche Extension (d.h. mit dem gleichen XML-Elementnamen) jedoch mit unterschiedlichen Attributwerten (z.B. `typ = 'Bild'`) dargestellt werden. Die Existenz eines Konzept-Mappings gibt somit an, dass eine Quelle eine Teilmenge der Extension des zugeordneten Konzeptes liefert.

Eine Property-Mapping-Struktur *PM* definiert die Abbildung einer Konzepteigenschaft auf Elemente bzw. Attribute der Quelldaten. Hier umfasst die Mapping-Struktur die Komponenten Quellenname sowie den Pfad zum die Eigenschaft repräsentierenden Element: *PM = (Source, PathToElement)*.

Kategorien werden in den Quellen durch einfache Werte (Literale) repräsentiert. Ein Property, das als Wertebereich eine Kategorie besitzt, kann somit in der Quelle exakt die definierten Werte dieser Kategorie sowie der davon abgeleiteten Unterkategorien annehmen. Die entsprechende Value-Mapping-Struktur *VM* umfasst daher nur den Quellennamen und den die Kategorie repräsentierenden Wert in der Quelle: *VM = (Source, Literal)*.

Mit diesen Mapping-Instanzen werden nun die Elemente des Konzeptschemas „annotiert“. Zu jedem Konzept, das von einer Quelle unterstützt wird, sind somit ein Konzept-Mapping- und die zugehörigen Property-Mapping-Beschreibungen sowie eventuell notwendige Value-Mapping-Definitionen anzugeben. Da jedoch das Konzeptschema als wesentliche Beziehung die Spezialisierung bzw. Generalisierung (*subClassOf*) verwendet, muss nicht zu jedem Konzept in einer Spezialisierungshierarchie eine entsprechende Abbildung definiert werden. Vielmehr ist dies nur für die Konzepte erforderlich, die bezüglich einer Quelle Blätter der Hierarchie darstellen. In Abb. 2 ist diese Zuordnung anhand eines Beispiels illustriert.

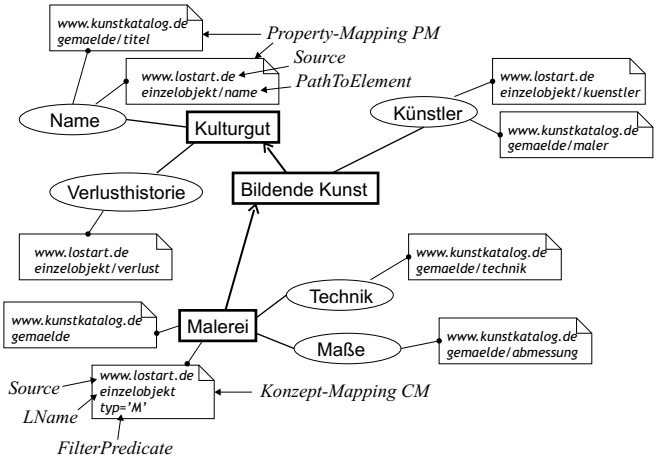

Abbildung 2: Abbildungsinformationen für die Konzepthierarchie

Das Beispiel zeigt auch, dass eine Quelle nicht notwendigerweise vollständige Daten im Sinne der Konzeptdefinition liefern muss. Werden von einer Quelle nicht alle in der Anfrage referenzierten Elemente unterstützt, so wird versucht, die fehlenden Informationen durch eine äußere Vereinigung aus einer anderen Quellen zu ergänzen. Die hierfür in Frage kommenden Quellen können anhand der vorhandenen Property-Mapping-Elemente identifiziert werden. Kann keine Quelle mit komplementären Informationen identifiziert werden, müssen im Anfrageergebnis die entsprechenden Elemente weggelassen werden (entspricht Null-Elementen).

Durch die Spezifikation der Mapping-Elemente für jede Quelle lassen sich sowohl Beschreibungskonflikte (Namenskonflikte von Elementen), strukturelle Konflikte als auch Datenkonflikte auflösen. Letztere jedoch nur für Kategorien, da hierbei eine Abbildung zwischen Werten stattfindet. Die hierfür notwendigen Transformationen werden bei der Anfrageübersetzung über die Mapping-Informationen und bei der Verarbeitung der Ergebnisse durch quellenspezifische (kompilierte) XSLT-Regeln durchgeführt, die bei der Registrierung einer Quelle automatisch generiert werden.

3 Die Anfragesprache CQuery

Das im vorangegangenen Abschnitt beschriebene Modell zur Datenintegration hat für die Planung und Ausführung von Anfragen zwei wesentliche Konsequenzen:

- Es werden Operationen benötigt, die sowohl auf Konzept- als auch auf Datenebene anwendbar sind und darüber hinaus einen Übergang zwischen beiden Ebenen ermöglichen.

- Eine globale Anfrage ist unter Verwendung der Abbildungsinformationen so zu transformieren und zu zerlegen, dass die zu den gewünschten Konzepten relevanten Quellen einbezogen und die einzelnen Teilanfragen autonom von den jeweiligen Quellen beantwortet werden können.

Zur Formulierung von Anfragen wird in dem hier vorgestellten Mediatorsystem eine CQuery genannte Variante von XQuery eingesetzt. Die Besonderheiten von CQuery sind im Wesentlichen semantischer Natur, syntaktisch folgt CQuery der FLWR-Notation von XQuery. Eine CQuery-Anfrage umfasst folgende Komponenten: die Auswahl von Konzepten anhand bestimmter Eigenschaften oder durch Operationen wie Traversierung von Beziehungen und Mengenoperationen, die Filterung der Daten als Instanzen der Konzepte sowie die Verknüpfung bzw. Projektion der Ergebnisse.

Der Aufbau einer typischen CQuery-Anfrage ist wie folgt:

Q_1 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c)
WHERE $e/kuenstler = 'van Gogh'
RETURN
  <painting>
    <title>$e/name</title>
    <artist>$e/kuenstler</artist>
  </painting>
```

Diese Anfrage liefert ein XML-Dokument mit Informationen über Gemälde in Form von `painting`-Elementen, die wiederum aus dem Titel und dem Künstlernamen bestehen.

In CQuery-Anfragen dient die **`FOR`**-Klausel zur Auswahl. Das Pseudoelement `concept` wird dabei als Dokumentbaum aus allen definierten Konzepten interpretiert, wobei die Konzepteigenschaften ebenfalls als Element angesehen werden. Auf diese Weise lassen sich die Standardmechanismen von XQuery auch für die Anfrageteile auf Konzeptebene nutzen. Spezielle Sprachkonstrukte oder -erweiterungen, wie sie etwa für RDF-Anfragesprachen vorgeschlagen wurden, sind somit nicht notwendig. Neben Selektionen bezüglich Konzepteigenschaften lassen sich auch Mengenoperationen wie **`UNION`**, **`EXCEPT`** und **`INTERSECT`** zwischen Konzeptmengen sowie Traversierung von Konzeptbeziehungen nutzen. Beziehungen werden dabei ebenfalls als Elemente aufgefasst, so dass für das Konzeptschema aus Abb. 1 der folgende Ausdruck

```
concept[name='Bildende Kunst']/!subClassOf
```

die direkt vom Konzept „Bildende Kunst“ abgeleiteten Konzepte liefert. Das dem Beziehungsnamen vorangestellte „!“-Zeichen gibt dabei an, dass die zu `subClassOf` inverse Beziehung (die hier nicht explizit benannt ist) betrachtet werden soll. Ein dem Beziehungsnamen folgendes „+“-Zeichen erzwingt dagegen die Bestimmung der transitiven Hülle bezüglich dieser Beziehung. So würde

```
concept[name='Bildende Kunst']/!subClassOf+
```

alle direkt oder indirekt vom Konzept „Bildende Kunst“ abgeleiteten Konzepte liefern. Zu beachten ist jedoch, dass speziell für die `subClassOf`-Beziehung die explizite Angabe

von „+“ nicht erforderlich ist, da bei der Bestimmung der relevanten Quellen die Transitivität dieser Beziehung berücksichtigt wird. Dies bedeutet, dass für die obige CQuery-Anfrage nicht nur die Quellen ausgewählt werden, die exakt das Konzept „Bildende Kunst“ unterstützen, sondern alle Quellen, die ein von diesem Konzept abgeleitetes Konzept repräsentieren. Diese implizite Ermittlung der transitiven Hülle eines Konzeptes erfolgt grundsätzlich nach einer eventuell angegebenen Pfadtraversierung aber vor einer Anwendung von Mengenoperation. Somit liefert der folgende Ausdruck alle Konzepte, die Kulturgüter – jedoch keine Möbel – sind:

```
concept[name='Kulturgut'] EXCEPT concept[name='Möbel']
```

In der **LET**-Klausel wird der Übergang von der Konzept- zur Instanzebene vollzogen. Hierzu dient die vordefinierte Funktion `extension()`, die zu einem Konzept die Extension, d.h. die Menge aller Instanzen der zugeordneten Quellen, liefert. Da die Konzepte in der **FOR**-Klausel einzeln einer Variablen zugewiesen werden, wird diese Funktion für jedes Konzept ausgewertet. Auf das Ergebnis – die Instanzmenge, die wiederum an eine Variablen gebunden werden kann – kann schließlich die Filterbedingung der **WHERE**-Klausel angewendet werden. Werden der Variablen in der **FOR**-Klausel nacheinander mehrere Konzepte zugewiesen, so werden die einzelnen mit `extension()` bestimmten Instanzmengen vereinigt – diese Operation wird im weiteren als extensionaler Union bezeichnet.

Während die Operationen der **FOR**-Klausel ausschließlich auf den Metadaten, d.h. im Mediator, ausgeführt werden, sind die Auswertung der `extension()`-Funktion sowie der Filterbedingung in der **WHERE**-Klausel mit Zugriffen auf die Quellsysteme verbunden. Die `extension()`-Funktion initiiert die Ausführung der Anfrage im Quellsystem (siehe unten), wobei – sofern möglich – die Filterbedingung ebenfalls als Teil dieser Quellenanfrage übernommen wird. Verbundoperationen werden formuliert, indem in der **LET**-Klausel mehrere verschiedene Variablen für Instanzmengen verwendet werden und im **WHERE**-Teil eine entsprechende Bedingung spezifiziert wird.

Im **LET**-Teil einer Anfrage lässt sich nicht nur die Extension zu einem Konzept bestimmen. Auch Anfragen über Eigenschaften sind möglich, indem das Pseudoelement `property` als Kindelement eines Konzeptes genutzt wird, das die Menge aller zu diesem Konzept definierten Eigenschaften repräsentiert. Werden die ausgewählten Eigenschaften an eine Variable gebunden, so kann diese auch zur Selektion auf Instanzebene genutzt werden, was wiederum einer disjunktiv verknüpften Anfrage über alle definierten Eigenschaften entspricht:

Q_2 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c), $p := $c/properties
WHERE $e/$p = 'Blumen'
  ...
```

Anfrage Q_2 liefert daher alle Gemälde, bei denen der Begriff „Blumen“ als Wert eines Properties auftaucht. Auf diese Weise sind Anfragen mit Schemaoperationen möglich, d.h. Operationen über Metadaten – in diesem Fall Informationen über Properties. Solche Operationen haben sich gerade in Anfragesprachen für heterogene Datenbanken als sehr nützlich zur Überwindung struktureller Heterogenitäten erwiesen.

In ähnlicher Form kann auf die einem Property zugeordneten Kategorien zugegriffen werden, indem im **LET**-Teil eine Variable an einen Pfadausdruck mit einem entsprechenden Property gebunden wird. Diese Variable verweist danach auf eine Menge von Kategorien und kann im **WHERE**-Teil anstelle eines Wertes eingesetzt werden:

Q_3 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c),
    $k := $c/stellt_dar[name='Stillleben']
WHERE $e/stellt_dar = $k
  ...
```

Mit dieser Anfrage wird zunächst der Variablen `$k` die Menge der Kategorien von Stillleben zugewiesen, indem die Menge aller Kategorien zur Beziehung `stellt_dar` bestimmt und diese dann auf die „Stillleben“ reduziert wird. Dies schließt auch alle von der konkreten Kategorie „Stillleben“ abgeleiteten Kategorien ein. In die Bedingung im **WHERE**-Teil wird dann der für die jeweilige Quelle lokal verwendet Begriff für „Stillleben“ bzw. im Fall von mehreren Kategorien eine disjunktive Verknüpfung über alle übersetzten Begriffe eingesetzt. Das Ergebnis der Anfrage Q_3 umfasst somit alle Gemälde, die ein Stillleben darstellen.

Die **RETURN**-Klausel hat die gleiche Bedeutung wie in XQuery: Hier kann das Anfrageergebnis entsprechend einer vorgegebenen XML-Dokumentstruktur ausgegeben werden, wobei die Ergebniselemente (d.h. sowohl die Instanzen als auch die Konzepte) über die eingeführten Variablen referenziert werden.

4 Anfragebearbeitung

Die Semantik der Anfragesprache lässt sich auf einfache Weise mit Hilfe von Algebraoperationen beschreiben. Ausgehend von den Definitionen aus Abschnitt 2 können zwei Gruppen von Anfrageoperationen unterschieden werden, die im Wesentlichen den bekannten Operationen der Relationenalgebra entsprechen. Operationen der Konzeptebene sind die Selektion Σ_{Cond}, die bekannten Mengenoperationen $\cup$, $\cap$, $-$ sowie die Pfadtraversierung $\Phi_P(c)$, die zu jedem Konzept einer Menge C alle über die Beziehung p referenzierten Klassen liefert:

$$\Phi_p(C) = \{c' \mid \exists c \in C : (p, c, c') \in \mathcal{P}\}$$

Die Traversierung einer inversen Beziehung kann entsprechend formuliert werden:

$$\Phi_{\overline{p}}(C) = \{c' \mid \exists c \in C : (p, c', c) \in \mathcal{P}\}$$

Weiterhin wird für die Berechnung der transitiven Hülle eines Konzeptes (bzw. einer Konzeptmenge) bezüglich einer Beziehung p eine eigene Operation Φ_p^+ eingeführt:

$$\Phi_p^+(C) = \{c' \mid \exists c_s \in C : (p, c_s, c') \in \mathcal{P} \vee \exists c_i \in \Phi_p^+(\{c_s\}) : (p, c_i, c') \in \mathcal{P}\}$$

Die Operationen der Instanzebene sind Selektion (σ), Projektion (π) sowie das kartesische Produkt ($\times$). Eine in CQuery formulierte Anfrage kann nun wie folgt in einen Ausdruck aus den obigen Operationen übersetzt werden.

1. Die Klausel **FOR** `$c` **IN** `concept[`*Cond*`]` wird in einen Ausdruck der Form $\Phi^{+}_{\overline{\textbf{is\_a}}}(\Sigma_{Cond}(\mathcal{C}))$ überführt. Hierbei steht $\overline{\textbf{is\_a}}$ für die inverse Beziehung zu c_2 **is_a** c_1.

2. Traversierungen der Form `concept[`*Cond*`]/`*prop*$_1$`/.../`*prop*$_n$ werden in $\Phi^{+}_{\textbf{is\_a}}(\Phi_{prop_n}(\ldots\Phi_{prop_1}(\Sigma_{Cond}(\mathcal{C}))))$ übersetzt.

3. Die Anwendung von Mengenoperationen wie `concept[`*Cond*$_1$`]` **UNION** `concept[`*Cond*$_2$`]` liefert folgenden Ausdruck:

$$\Phi^{+}_{\textbf{is\_a}}(\Sigma_{Cond_1}(\mathcal{C})) \cup \Phi^{+}_{\textbf{is\_a}}(\Sigma_{Cond_2}(\mathcal{C}))$$

4. Ein Ausdruck **LET** `$e := extension($c)` **WHERE** *Cond* wird transformiert, indem die Übersetzung *CExpr* des an die Variablen `$c` gebundenen Ausdrucks in folgenden Ausdruck eingesetzt wird, wobei $\biguplus$ für die äußere Vereinigung steht:

$$\biguplus_{c \in CExpr} \sigma_{Cond}(\textbf{ext}(c))$$

   Die Verwendung der äußeren Vereinigung als Integrationsoperation erlaubt die Kombination von Datenquellen mit teilweiser Überlappung bzw. mit einer partiellen Union-Kompatibilität der Schemata, wobei angenommen wird, dass die in allen Relationen vorhandenen Attribute (bzw. Elemente) Schlüsseleigenschaften aufweisen. Tritt ein Tupel mit seinem Schlüsselwert in allen Eingangsrelationen auf, so wird es nur einmal in die Ergebnismenge aufgenommen. Anderenfalls werden die fehlenden Attribute bzw. Elemente weggelassen, was dem Auffüllen mit Nullwerten im relationalen Fall entspricht.

5. Werden in der **LET**-Klausel mehrere Variablen für Instanzmengen bzw. Property- oder Kategoriemengen eingeführt, so wird das kartesische Produkt über diesen Mengen gebildet.

6. Die **RETURN**-Klausel wird in eine entsprechende Projektion π umgesetzt, die neben XML-Tags als Literale auch Pfadausdrücke zur Projektion von Attributen umfasst.

Für eine Anfrage

Q_5 :
```
FOR $c IN concept[name='Malerei']
LET $e := extension($c)
WHERE $e/kuenstler = 'van Gogh'
RETURN
  <painting>
    <title>$e/title</title>
    <artist>$e/kuensler</artist>
  </painting>
```

ergibt sich unter Anwendung der obigen Regeln folgender Algebraausdruck, wobei *Proj* für den Projektionsausdruck aus Q_5 steht:

$$\biguplus_{c \in \Phi^{+}_{\textbf{is\_a}}(\Sigma_{\text{name='Malerei'}}(\mathcal{C}))} \pi_{Proj}(\sigma_{\text{kuenstler='van Gogh'}}(\textbf{ext}(c)))$$

Die Übersetzung einer CQuery-Anfrage in die interne Algebraform bildet auch gleichzeitig die Vorbereitung für die eigentliche Anfragebearbeitung, deren Ablauf in Abb. 3 dargestellt ist.

Gegeben:
Anfrageausdruck der Form $\biguplus_{c \in CExpr} IExpr(c)$
Ergebnis $R := \{\}$

Berechne Konzeptmenge $C :=$ *CExpr*
forall $c \in C$ **do**
/ Anfrage im Cache suchen → Ergebnis ist R_c */*
$R_c := \text{cache-lookup}(IExpr(c))$
if $R_c \neq \{\}$ **then**
/ Cache-Eintrag gefunden */*
$R := R \uplus R_c$
else
/ Quellenanfrage ableiten */*
Bestimme alle in *IExpr* referenzierten Properties p_i und Kategorien k_j
forall CM_s zu c **do**
$s :=$*CM*$_s(c)$*.Source*
/ nur nicht-redundante Konzepte anfragen */*
if $c \in \text{cmin}(C, s)$ **then**
/ zu jeder unterstützenden Quelle eine eigene Anfrage konstruieren */*
Transformiere *IExpr(c)* entsprechend $CM_s(c)$, $PM_s(p_i)$ und $VM_s(k_j)$ in eine Quellenanfrage Q
Führe Q an Quelle $CM_s(c)$*.Source* aus; Ergebnis ist R_Q
$R := R \uplus R_Q$
fi
od
fi
od

Abbildung 3: Ablauf der Anfragebearbeitung

Dieser Algorithmus verarbeitet nur elementare Anfrageausdrücke über Konzeptmengen und deren Extensionen. Anfragen mit kartesischen Produkt bzw. Verbund werden in elementare Teilausdrücke zerlegt, die dann in der dargestellten Weise verarbeitet werden können. Dabei wird auch versucht, durch einfache Heuristiken eine Vereinfachung der Anfragen vorzunehmen. Hierzu zählen neben den bekannten algebraischen Optimierungsregeln („Herunterdrücken“ von Selektionen) insbesondere die Ersetzung von Zugriffen auf Schemaelemente (wie in Anfrage Q_2) durch disjunktiv verknüpfte Bedingungen sowie die Ersetzung von Kategorievariablen durch die entsprechenden Wertemengen (Anfrage Q_3).

Der erste Schritt ist die Auswertung der Operationen auf Konzeptebene. Zu jedem der dadurch ermittelten Konzepte wird zunächst versucht, den extensionsbezogenen Teil *IExpr* der Anfrage aus dem Cache zu beantworten (siehe hierzu Abschnitt 5). Schlägt dies

fehl, werden alle Konzept-Mappings und damit alle dieses Konzept unterstützende Quellen bestimmt. Da zwischen den Extensionen verschiedener Konzepte einer Quelle Überlappungen bzw. Redundanzen existieren können, werden nur nicht-redundante Konzepte berücksichtigt. Diese werden mit Hilfe der Funktion $\mathrm{cmin}(C, s)$ ermittelt, die zu einer gegebenen Quelle s die minimale Teilmenge der Konzeptmenge C mit Hilfe folgender Heuristiken bestimmt.

Eliminierung Betrachtet man Konzepthierarchien als Klassenhierarchien mit extensionalen Beziehungen, so könnte bei einer Beziehung c_2 **is_a** c_1 zwischen zwei Konzepten c_1 und c_2, die von der selben Quelle unterstützt werden, die Anfrage zur Bestimmung der Extension von c_2 entfallen, da **ext**$(c_2) \subseteq$ **ext**(c_1) gilt. Da in dem hier betrachteten Szenario Konzepthierarchien nicht als Ergebnis einer Schemaintegration definiert werden, gilt diese Annahme jedoch nicht zwingend, sondern nur dann, wenn beide Konzepte in der Quelle durch die gleiche Klasse, d.h. repräsentiert durch das gleiche lokale Element im Konzept-Mapping *CM*(c), unterstützt werden:

$$c_2 \textbf{ is\_a } c_1 \wedge \mathit{CM}(c_1).\mathit{LName} = \mathit{CM}(c_2).\mathit{LName} \Rightarrow \textbf{ext}(c_2) \subseteq \textbf{ext}(c_1)$$

Zusammenfassen Teilanfragen zu Konzepten in parallelen Zweigen einer Konzepthierarchie, die sich auf die gleiche Quelle beziehen, können zusammengefasst werden, wenn sie auch die gleiche Klasse betreffen, d.h. wenn $\mathit{CM}(c_1).\mathit{LName} = \mathit{CM}(c_2).\mathit{LName}$. In diesem Fall können die gegebenenfalls spezifizierten Filterbedingungen der Konzept-Mappings disjunktiv verknüpft werden.

Im nächsten Schritt werden die verbliebenen Teilanfragen anhand der Mappings in Quellenanfragen übersetzt. Das Konzept-Mapping liefert dabei den Bezeichner des lokalen Elementes, die Property-Mappings zu den dem Konzept zugeordneten Properties die Bezeichner der Attribute und die Value-Mappings die Übersetzung von Begriffen. Eine Übersetzung des Anfrageausdrucks $\sigma_{P\theta v}(\textbf{ext}(c))$ wird demnach unter Verwendung der Mapping-Informationen *CM*(c) und *PM*(p) in einen XPath-Ausdruck der Form

$$/\langle \mathit{CM}(\mathrm{c}).\mathit{LName}\rangle[\langle \mathit{PM}(\mathrm{p}).\mathit{PathToElement}\rangle\ \theta\ \mathrm{v}]$$

überführt. Quellenanfragen sind demzufolge nur einfache Selektionen über lokalen Extensionen, die auch von Nicht-DBMS-Quellsystemen beantwortet werden können. Darüber hinausgehende Operationen wie Vereinigung oder Verbund werden ausschließlich vom Mediator ausgeführt. Ein Delegieren dieser Operationen an die Quellsysteme würde die Berücksichtigung von Quelleneigenschaften erforderlich machen, wie dies u.a. in [RS97] diskutiert wird.

5 Caching von Anfrageergebnissen

Bei der virtuellen Integration von Datenbeständen ergeben sich hohe Anforderungen an die Anfragebearbeitung, um global eine effiziente Ausführung zu gewährleisten. Dies trifft insbesondere zu, wenn die Integration über Datennetze mit begrenztem Durchsatz wie dem Web geschieht. Dabei umfasst die Effizienz einerseits, dem Nutzer des globalen Systems in

angemessenen Antwortzeiten Ergebnisse bereitzustellen, und andererseits die Auslastung der integrierten Quellen durch redundante oder irrelevante Anfragen zu minimieren.

Das semantische Caching von Anfragen und zugehörigen Ergebnissen hat sich in vergleichbaren Szenarien als geeignetes Mittel zur Erfüllung der genannten Anforderungen erwiesen. Dabei wird auf der Basis zuvor zwischengespeicherter Anfragen entschieden, ob eine neue Anfrage ganz oder teilweise aus den zugehörigen zwischengespeicherten Anfrageergebnissen beantwortet werden kann. Für die Umsetzung eines solchen semantischen Caches sind zu berücksichtigen *(i)* die Integration in die globale Anfrageverarbeitung, *(ii)* die Anbindung an das Konzeptmodell, *(iii)* die physische Speicherung der Anfragen und Ergebnisse sowie *(iv)* eine Cache-Management-Strategie.

Da im YACOB-System der Schwerpunkt des Caching auf der Unterstützung einer schrittweisen interaktiven Anfrageformulierung liegt, betrachten wir im Folgenden vor allem folgende Fälle der Anfrageverfeinerung:

1. Einschränkung der Ergebnismenge durch die Verminderung der Menge relevanter Konzepte bzw. die Verminderung der Menge relevanter Instanzen durch Angabe zusätzlicher konjunktiv verknüpfter Prädikate oder die Entfernung disjunktiv verknüpfter Prädikate.

2. Erweiterung der Ergebnismenge durch die Erweiterung der Menge relevanter Konzepte bzw. die Erweiterung der Menge relevanter Instanzen durch die Entfernung konjunktiv verknüpfter Prädikate oder die zusätzliche Angabe disjunktiv verknüpfter Prädikate.

Die angegebenen Fälle korrespondieren zu der in Abschnitt 3 beschriebenen Trennung der Konzept- und der Instanzebene entsprechend der **LET**-Klausel. Durch ein Caching unterhalb der Konzeptebene im Mediator kann die Berücksichtigung zwischengespeicherter Inhalte auf die Untersuchung der Filterprädikate entsprechend der **WHERE**-Klausel reduziert werden. Das heißt, zu jedem Konzept existiert somit ein Cache-Fragment, in dem Paare bestehend aus einer Anfrage und den dazugehörigen transformierten Ergebnismengen bestehen. Dabei bestehen die Anfragen an dieser Stelle nur aus einfachen Selektionsbedingungen konjunktiv verknüpfter Konstantenselektionen. Disjunktiv verknüpfte Prädikate erhalten jeweils einen eigenen Cache-Eintrag.

Ein Cache-Eintrag besteht jeweils aus der oben beschriebenen *condition* und dem zugehörigen Ergebnis *instances*. Davon und vom Algorithmus aus Abb. 3 ausgehend wird der Cache entsprechend dem Algorithmus in Abb. 4 ausgewertet. Dazu wird zuerst die Filterbedingung in konjunktive Normalform zerlegt, d.h. eine Menge von Konjunktionen entsprechend der Beschreibung von Cache-Einträgen im *condition*-Teil gebildet. Die Konjunktionen, bestehend aus einer Menge von Prädikaten, werden einzeln mit den Cache-Einträgen verglichen. Bestehen beide aus gleichen Prädikaten, mit gleichem Attributnamen, Operator und Vergleichswert, repräsentieren die zwischengespeicherten Instanzen das zugehörige Ergebnis. Ist die Anfragekonjunktion spezieller, d.h. sie umfasst zusätzliche Prädikate, wird die Teilanfrage in den zwischengespeicherten Instanzen ausgewertet. Ein weiterführender Algorithmus könnte bei Überlappung der Prädikatmengen oder der von den Prädikaten beschriebenen Datenbereiche eine komplementäre Anfrage bilden, wozu jedoch

Gegeben:
Filterausdruck *IExpr*(c) über Konzept c
Cache P_c für Konzept c
Ergebnis $R := \{\}$

cache-lookup(*IExpr*(c)):
DNF := disjunctive-normalform(*IExpr*(c))
forall *conj* $\in$ *DNF* **do**
forall *ce* $\in P_c$ **do**
if *ce.condition* $=$ *conj* **then**
/ Cache-Eintrag ist Ergebnis der Teilanfrage */*
$R := R \cup$ *ce.instances*
else if *ce.condition* $\subset$ *conj* **then**
/ Teilanfrage kann aus Cache-Eintrag beantwortet werden */*
Führe die *conj* entsprechende Anfrage über die
Cache-Daten in *ce.instances* aus, Ergebnis ist R_{conj}
$R := R \cup R_{conj}$
fi
od
od

Abbildung 4: Auswertung des Cache während der Anfrageverarbeitung

eine Auswertung der Prädikatsemantik erforderlich wäre. Das beschriebene Vorgehen ist für die betrachtete Anwendung jedoch ausreichend, da die Prädikate aufgrund der Charakteristika der Daten hauptsächlich Tests auf Zeichenketten-Gleichheit enthalten.

Die physische Speicherung der Anfragen und Dokumentsegmente erfolgt mit Hilfe der XML-Datenbank Xindice. Neben Vorteilen wie Ausfallsicherheit und besserer Skalierbarkeit, die durch eine derartige persistente Verwaltung der Anfrageergebnisse gewährleistet wird, basiert der oben beschriebene Algorithmus zur Auswertung des Cache auf der integrierten XPath-Anfragekomponente von Xindice. Diese wird benutzt, um Teilanfragen aus Cache-Segmenten zu beantworten, wenn das korrekte Ergebnis einer spezielleren Anfrage im Segment enthalten ist.

Das Cache Management ist in diesem Szenario denkbar einfach, da der Cache hauptsächlich zur Unterstützung interaktiver Sitzungen vorgesehen ist. Die Verdrängung von Cache-Segmenten ist daher mit einem einfachen Zeitstempel an den Kopfelementen umgesetzt, der bei einem Zugriff aktualisiert wird. Überschreitet der Zeitstempel ein Vielfaches der durchschnittlichen Sitzungszeit, wird der Eintrag entfernt.

6 Architektur und Implementierung

Das YACOB-Mediatorsystem ist vollständig in Java implementiert, wobei auf eine Reihe von Standardtechnologien und frei verfügbaren Modulen zurückgegriffen wurde. So er-

folgt die Verarbeitung von XML-Daten (Parsing, XSLT-Transformation) mit JAXP (Java API for XML Processing) und der Zugriff auf die Quellsysteme bzw. Wrapper über Web Services. Zur Verwaltung und zur Manipulation des auf RDFS basierenden Konzeptmodells wird Jena – das Java-API for RDF – eingesetzt. Dieses Paket stellt nicht nur einen RDF-Parser und die entsprechenden Zugriffsschnittstellen bereit, sondern unterstützt auch die RDF-Anfragesprache RDQL sowie die persistente Speicherung von RDF-Modellen in einer relationalen Datenbank. Als Cache wird das XML-Datenbanksystem Xindice genutzt, die Benutzerschnittstelle ist über Java Server Pages realisiert.

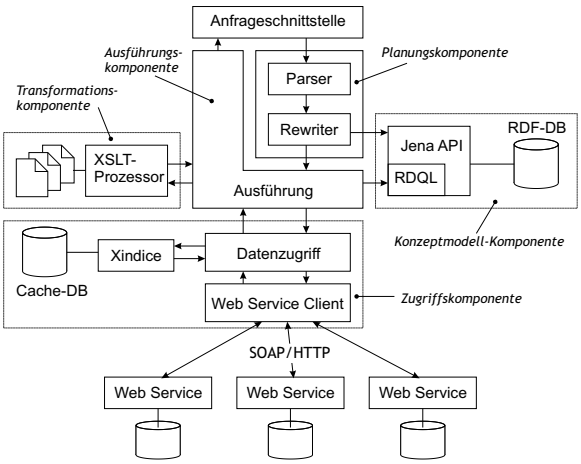

Abbildung 5: Architektur des Mediatorsystems

Das Zusammenwirken dieser Module sowie die Verbindung mit dem eigentlichen Anfragesystem ist in Abb. 5 dargestellt. Es lassen sich grob die folgenden Komponenten unterscheiden:

- die Konzeptmodell-Komponente, die auf Basis des Jena-API die Verwaltung von Konzepten, deren Eigenschaften und Beziehungen sowie der Abbildungsinformationen auf die Strukturen der Quellsysteme realisiert,
- die Anfrageplanungskomponente bestehend aus dem CQuery-Parser und dem Modul zur Anfragetransformation, -zerlegung und -übersetzung,
- die Anfrageausführungskomponente mit der Implementierung der Anfrageoperatoren sowie der Überwachung und Steuerung der Ausführung,
- die Transformationskomponente, die im Wesentlichen durch den mit JAXP bereitgestellten XSLT-Prozessor realisiert wird und die Transformation der Ergebnisdaten aus den Quellsystemen vornimmt, sowie
- die Zugriffskomponente, die XPath-Anfragen von der Ausführungskomponente entgegennimmt und entweder über das Web-Service-Protokoll SOAP an die Quellsys-

teme bzw. spezifische Wrapper weitergibt oder mit Hilfe von Xindice aus dem Cache beantwortet.

Eine Besonderheit des Systems ist die enge Verzahnung von Anfrageplanung und -ausführung. Da erst nach Ermittlung der Konzepte und deren Abbildung auf die Quellstrukturen festgestellt werden kann, welche Quellen anzufragen sind und wie die Anfragen zu übersetzen sind, muss zunächst der **FOR**-Teil einer Anfrage ausgeführt werden. Die entsprechenden Operatoren sind als Teil der Ausführungskomponente implementiert und greifen über das Jena-API und die RDF-Anfragesprache RDQL auf die RDF-Daten des Konzeptmodells zu. RDQL wird zur teilweisen Realisierung der Algebraoperatoren der Konzeptebene genutzt. Aufgrund der bestehenden Restriktionen der Sprache sind jedoch einzelne Operatoren wie die Berechnung der transitiven Hülle durch direkten Zugriff auf den RDF-Graphen implementiert. Das Ergebnis dieser Ausführung – eine Menge von Konzepten – wird an die Planungskomponente zurückgegeben, die wiederum die Mapping-Informationen über Jena bzw. RDQL ermittelt und davon ausgehend die Zerlegung und Übersetzung der Teilanfragen entsprechend den lokalen Schemata vornimmt. Dieser Plan – ein Graph aus Algebraoperatoren – wird schließlich wieder an die Ausführungskomponente übergeben, die wiederum zur Verarbeitung der entfernten Anfragen die Dienste der Zugriffskomponente in Anspruch nimmt.

XML-Daten, die als Ergebnis der Ausführung einer entfernten Anfrage von der Zugriffskomponente geliefert werden, müssen zunächst durch Anwendung der quellenspezifischen XSLT-Regeln in das globale, durch das Konzeptschema vorgegebene Schema transformiert werden. Parallel zur Weiterverarbeitung durch weitere eventuell in der Anfrage formulierte globale Operationen (z.B. Verbund, Projektion usw.) werden die transformierten Daten in der Cache-Datenbank gespeichert. Gleichzeitig werden die Mapping-Informationen zum entsprechenden Konzept mit dem Cache-Eintrag aktualisiert.

Der Zugriff auf die Quellsysteme bzw. gegebenenfalls notwendige Wrapper erfolgt über Web Services. Hierzu muss jede beteiligte Quelle eine Anfrageschnittstelle in Form eines einfachen Web Services unterstützen, der einen XPath-Ausdruck als Parameter erwartet und ein XML-Dokument mit den Anfrageergebnissen zurückliefert. Auf Mediatorseite wird der Zugriff über das Java API for XML Messaging (JAXM) realisiert, das eine einfache Generierung und Verarbeitung von SOAP-Nachrichten ermöglicht.

7 Benutzerschnittstelle

Für den Mediator wurde eine grafische Benutzerschnittstelle entwickelt, die berücksichtigt, dass die Anwender in dem betrachteten Szenario eher Kunsthistoriker, Anwälte, Vertreter von Auktionshäuser sowie andere Interessierte sind und somit mit der direkten Formulierung von Anfragen in CQuery sicher überfordert wären. Daher sollten die Informationen über Konzepte, deren Eigenschaften und Kategorien explizit einbezogen werden. Die Benutzungsoberfläche besteht dabei aus drei Teilen: die Darstellung der Konzepte als einen Baum, die Darstellung der Eigenschaften sowie die Repräsentation der Ergebnisse.

Im ersten Teil wird das RDF-Modell ausgehend von dem Wurzelkonzept „Kulturgut“

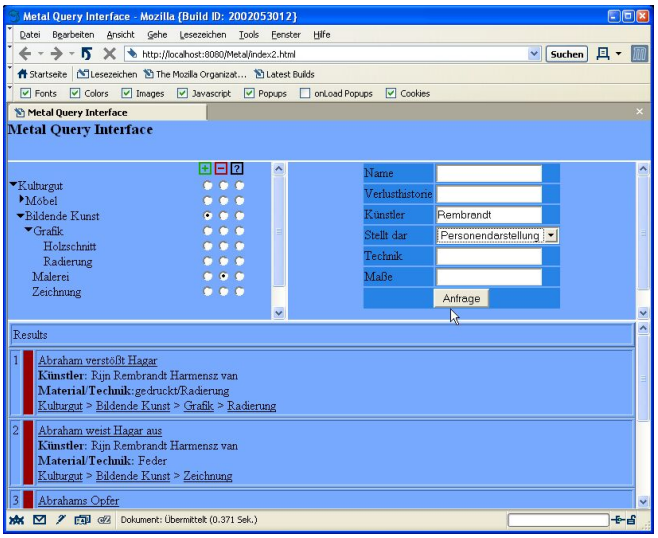

Abbildung 6: Anfragesitzung mit der Suchschnittstelle

durchlaufen und die Konzepte in einer Baumstruktur dargestellt (siehe Abb. 6). Der Nutzer kann dabei für jedes Konzept festlegen, ob Objekte des Konzeptes im Ergebnis enthalten sein dürfen, sein müssen bzw. nicht enthalten sein dürfen. Diese Informationen werden auf die **FOR**-Klausel innerhalb einer CQuery-Anfrage abgebildet.

Der zweite Teil der Oberfläche unterstützt die Behandlung der Eigenschaften zur weiteren Verfeinerung der Suche. Die Eigenschaften werden schrittweise durch die Auswahl von weiteren Konzepten ergänzt, wobei das Suchformular entsprechend angepasst wird. Hierbei sind Eigenschaften mit und ohne unterstützenden Kategorien, wie z.B. Motiv oder Epoche zu unterscheiden. Im Fall ohne Kategorieelemente wird dem Nutzer ein einfaches Texteingabefeld zur Freitextsuche angeboten. Des Weiteren werden die eingegebenen Werte in die **WHERE**-Klausel übernommen. Im zweiten Fall wird dem Nutzer eine Auswahl der verschiedenen Kategorien dargestellt. Für eine Anfrage müssen sowohl die **LET**- als auch die **WHERE**-Klausel benutzt werden.

Der dritte Teil der Benutzeroberfläche dient der Präsentation der Ergebnisse und der Verfeinerung der Anfrage. Die Ergebnisliste umfassen neben den Informationen über die Eigenschaften der Kulturgüter auch zusätzliche Angaben über die Zugehörigkeit zu den Extensionen der Konzepte. Die Zugehörigkeit wird durch einen Pfadausdruck vom Wurzelkonzept „Kulturgut“ bis zum eigentlichen Konzept ausgedrückt. Durch eine Auswahl des Konzeptes ist es möglich, die Anfrage direkt einzuschränken bzw. um zusätzliche Objekte zu erweitern. Dieses hat den Vorteil, dass dem Anwender ermöglicht wird, zusätzliche Konzepte zur Anfrage hinzufügen, sobald er ähnliche Objekte zu seinem geforderten Suchergebnis entdeckt. Diese Art der iterativen Anfrageverfeinerung profitiert besonders stark von den vorgestellten Caching-Möglichkeiten des Mediators (siehe Abschnitt 5).

8 Verwandte Arbeiten

Systeme auf der Basis der Mediator-Wrapper-Architektur haben sich in den letzten Jahren als ein geeigneter Ansatz zur Integration von heterogenen, semistrukturierten Datenquellen im Web etabliert. Als Vorreiter auf diesem Gebiet können die bekannten TSIMMIS [GMPQ+97] und Information Manifold [LRO96] angesehen werden. Aktuelle Arbeiten wie etwa MIX [BGL+99] nutzen auch bereits XML als Austauschformat bzw. XML-basierte Anfragesprachen.

Ein Vertreter der semantischen oder ontologiebasierten Integration ist das System KIND [LGM01], das sogenannte Domain Maps verwendet. Eine Domain Map wird im generischen konzeptuellen Modell GCM spezifiziert, das wiederum auf einer Teilmenge von F-Logik basiert, und als eine Menge von Klassen (Konzepten) und deren Verbindungen über binäre Relationen modelliert. Eine neue Quelle wird in KIND registriert, indem die Objekte den Konzepten der Domain Map als Instanzen zugeordnet („verankert“) werden. Über den so registrierten Quelldaten lassen sich integrierende Sichten in Form von Anfragen (Regeln) definieren. Grundsätzlich verfolgt KIND somit einen Global-as-View-Ansatz.

Ein weiterer Vertreter ist SIMS [ACHK93], das auf der Wissensrepräsentationssprache Loom – einer Erweiterung von KL-ONE – basiert. Damit wird eine hierarchische terminologische Wissensbank spezifiziert, die das Domänenmodell darstellt. Mit diesem Modell werden wiederum die Inhalte der zu integrierenden Quellen unabhängig voneinander beschrieben, im Wesentlichen durch *is-a*-Beziehungen zwischen den globalen und den lokalen Konzepten. Auf diese Weise wird ein LaV-Ansatz realisiert.

Der in [GBMS99] beschriebene Context-Mediator verwendet ebenfalls ein Domänenmodell, das hier eine Menge von primitiven und semantischen Typen umfasst. Instanzen von semantischen Typen können dabei unterschiedliche Werte in verschiedenen Kontexten haben, d.h. in den einzelnen Quellen können verschiedene Annahmen über die Interpretation der Werte getroffen werden. Mit den Quellen lassen sich Kontextaxiome assoziieren, die die Konvertierung der lokalen Werte in das globale Domänenmodell spezifizieren.

Dem hier vorgestellten YACOB-System am nächsten kommt der XML-Mediator STYX [ABFS02], der ebenfalls auf dem LaV-Prinzip basiert und eine Ontologie als Integrationsmodell nutzt. Die Quellenbeschreibungen erfolgen durch XPath-Ausdrücke. Als Anfragesprache wird eine einfache OQL-ähnliche Sprache unterstützt, wobei die Konzepte Klassen entsprechen. Die einzige Operation auf Konzeptebene ist die Traversierung von Beziehungen.

Im Vergleich mit diesen Ansätzen kann das in diesem Beitrag vorgestellte YACOB-Mediatorsystem der semantischen Integration zugeordnet werden, wobei das LaV-Prinzip verfolgt wird. Im Gegensatz etwa zu SIMS erfordert der YACOB-Mediator jedoch keine explizite Modellierung der Quelleninhalte, sondern diese werden den Konzepten des Domänenmodells in Form von Abbildungsbeschreibungen zugeordnet. Die speziellen Kategorieklassen des YACOB-Konzeptmodells lassen sich mit den semantischen Typen des Context-Mediators vergleichen, allerdings werden die dort verwendeten Axiome in YACOB vereinfacht durch Wertabbildungen ersetzt. Die Verwendung semantischer Typen ist jedoch primär auf die Behandlung von Attributwertkonflikten ausgerichtet und nicht wie

beim hier beschriebenen Ansatz auf die Überwindung semantischer Heterogenität bei der Integration. Eine weitere Besonderheit des YACOB-Mediators ist die Unterstützung von Schemaoperationen durch die Anfragesprache CQuery. Dies erlaubt nicht nur die Ermittlung von Properties zur Laufzeit sondern auch die „Berechnung“ von Konzepten, zu denen Daten aus den Quellen angefragt werden sollen.

Gerade im Zusammenhang mit Mediatoren wurden in letzter Zeit Techniken zum Semantic Caching entwickelt. Dabei wir das Caching statt auf physischer Ebene, auf Datenebene zusammen mit Daten zur Beschreibung der Cache-Einträge durchgeführt [DFJ+96]. Diese Technik kommt insbesondere bei der Datenintegration [ASPS96] und im World Wide Web [LC01] zum Einsatz.

Hauptanliegen der Bemühungen zum Semantic Web ist die Verbesserung der automatisierten Verarbeitbarkeit von Web-Informationen durch Hinzufügen von den Inhalt beschreibenden Metainformationen – d.h. die Interoperabilität auf semantischer Ebene zu erreichen. Eine Schlüsselrolle spielen dabei Ontologien. Als einfachste Form kann in diesem Zusammenhang das hier verwendete RDFS angesehen werden. Für RDF und RDFS wurden auch bereits diverse Anfragesprachen vorgeschlagen, welche die Besonderheiten von RDF-Beschreibungen etwa im Vergleich zu einfachen XML-Dokumenten berücksichtigen [MKA+02]. Allerdings sind RDF-Anfragesprachen nur auf die Metadatenebene beschränkt und können so allenfalls – wie in Abschnitt 6 gezeigt – als eine Teilkomponente eines konzeptbasierten Anfragesystems angesehen werden. Weitergehende Ansätze für Ontologiesprachen wie DAML+OIL [Hor02] bieten Erweiterungen zu RDFS, wie Klassenkonstruktoren, Axiome, mit denen Subsumption oder Äquivalenz von Klassen und Eigenschaften ausgedrückt werden können, und Inferenzmechanismen. Andere Beispiele für die Nutzung von Ontologien zur Informationsintegration sind u.a. Portale [MSS+02] oder semantische Suchmaschinen wie OntoBroker [DEFS99].

9 Zusammenfassung und Ausblick

Die Repräsentation von Hintergrundwissen in Form von Konzepten und deren Beziehungen als semantische Metadaten von Mediatoren kann in Verbindung mit einem Local-as-View-Ansatz die Integration neuer Quellen sowie die Formulierung von globalen Anfragen vereinfachen. Vor diesem Hintergrund wurden in diesem Beitrag ein Anfragesystem zu einer konzeptbasierten XML-Anfragesprache vorgestellt und die Schritte der Transformation und Ausführung von Anfragen beschrieben. Als Integrationsmodell wurde dabei RDF Schema gewählt, wodurch die Nutzung verfügbarer Werkzeuge zur Modellierung und zum Datenaustausch möglich ist.

Das gewählte Anwendungsgebiet – die Integration kulturhistorischer Internet-Datenbanken – wird speziell durch eine Benutzerschnittstelle berücksichtigt, die eine inkrementelle Anfrageverfeinerung erlaubt und durch einen Anfrage-Cache unterstützt wird. Mit dem implementierten Prototypen ist der Zugriff auf die Internet-Datenbanken *www.lostart.de* über eine interne JDBC-Schnittstelle, die vom Web Service genutzt wird, sowie auf *www.herkomstgezocht.nl* über einen Wrapper möglich.

Literatur

[ABFS02] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *ISWC 2002*, LNCS 2342, pages 117–131. Springer-Verlag, 2002.

[ACHK93] Y. Arens, C.Y. Chee, C.-N. Hsu, and C.A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[ASPS96] S. Adah, K. Selcuk Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *SIGMOD'96*, pages 137–148, 1996.

[BGL+99] C.K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In *SIGMOD'99*, pages 597–599, 1999.

[DEFS99] S. Decker, M. Erdmann, D. Fensel, and R. Studer. OntoBroker: Ontology-based Access to Distributed and Semi-Structured Information. In *DS-8: Semantic Issues in Multimedia Systems*. Kluwer, 1999.

[DFJ+96] S. Dar, M.J. Franklin, B.T. Jónsson, D. Srivastava, and M. Tan. Semantic Data Caching and Replacement. In *VLDB'96*, pages 330–341, 1996.

[GBMS99] C.H. Goh, S. Bressan, S.E. Madnick, and M.D. Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.

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

[Hal01] A.Y. Halevy. Answering Queries using Views: A Survey. *The VLDB Journal*, 10(4):270–294, 2001.

[Hor02] I. Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 4–9, 2002.

[LC01] D. Lee and W.W. Chu. Towards Intelligent Semantic Caching for Web Sources. *Journal of Intelligent Information Systems*, 17(1):23–45, 2001.

[LGM01] B. Ludäscher, A. Gupta, and M.E. Martone. Model-based Mediation with Domain Maps. In *ICDE'01*, pages 82–90, 2001.

[LRO96] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB'96*, pages 251–262. Morgan Kaufmann, 1996.

[MKA+02] A. Magkanaraki, G. Karvounarakis, Ta Tuan Anh, V. Christophides, and D. Plexousakis. Ontology Storage and Querying. Technical Report 308, Foundation for Research and Technology Hellas, Institute of Computer Science, April 2002.

[MSS+02] A. Maedche, S. Staab, R. Studer, Y. Sure, and R. Volz. SEAL – Tying Up Information Integration and Web Site Management by Ontologies. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 10–17, 2002.

[RS97] M.T. Roth and P.M. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *VLDB'97*, pages 266–275, 1997.