# Effiziente Speicherung von RDF (Resource Description Framework)

*Diplomarbeit von Valerie Bures*

# Inhaltsverzeichnis

**1 EINLEITUNG ........ 3**

**2 GRUNDLAGEN VON RDF ........ 4**

2.1 EINFÜHRUNG ........ 4

2.2 RDF UND RDF SCHEMA ........ 4

2.3 RDF ANFRAGESPRACHE UND SPEICHER ........ 10

2.4 ZUSAMMENFASSUNG ........ 13

**3 SPEICHERUNG VON RDF DATEN ........ 14**

3.1 EINFÜHRUNG ........ 14

3.2 SPEZIFIKATION DES FORMALEN GRAPHENMODELLS IN RDF ........ 14

3.3 ÜBERBLICK ÜBER DIE EIGENSCHAFTEN OBJEKTORIENTIERTER DATENBANKEN ........ 17

3.4 VORTEILE UND DARSTELLUNG DER ABBILDUNG DES RDF-GRAPHEN AUF EIN OBJEKTORIENTIERTES DATENBANKSCHEMA ........ 18

3.5 ZUSAMMENFASSUNG ........ 20

**4 IMPLEMENTIERUNG ........ 20**

4.1 EINFÜHRUNG ........ 20

4.2 SESAME: EINE GENERISCHE SPEICHER- UND ANFRAGEARCHITEKTUR FÜR RDF UND RDF SCHEMA ........ 21

4.3 IMPLEMENTIERUNGEN VON SESAME ........ 24

4.4 VERGLEICH UND BEURTEILUNG ........ 32

**5 PERFORMANCE ........ 33**

5.1 HOCHLADEN ........ 33

5.2 RQL ........ 33

5.3 ANFRAGEN FÜR DIE TESTS ........ 37

5.4 ZEIT FÜR RQLANFRAGEN ........ 38

5.5 PLATZ ........ 38

**6 DISKUSSION UND AUSBLICK ........ 38**

**7 ANHANG ........ 38**

7.1 ........ 40

**8 LITERATUR UND QUELLEN ........ 40**

# 1 Einleitung

Allgemeines.......

Die Einführung in RDF und RDF Schema zeigt, dass die Verwertung von RDF Daten zur praktischen Nutzung die Einbindung eines Schematas erfordert, um den Daten eine gewisse Semantik zu geben.
Zur Verarbeitung dieser Daten aus Schema und reiner Information muß der sich aus den Ausdrücken ergebene Graph interpretiert werden.
Anfragen an die RDF Daten sind damit nur über den Graph sinnvoll, was alleine in RQL möglich ist.
Als grundlegende Kriterien eines effizienten RDF Speichersystems ergeben sich damit die Abbildbarkeit des formalen Graphenmodells auf die Speicherstruktur und die Anfragemöglichkeit in RQL.
Im Folgenden werden die bekanntesten bestehenden Systeme nach diesen Kriterien klassifiziert:

Es exestieren bisher sechs RDF-Systeme, von denen RDFSuite [RDFSuite] und Sesame [Sesame] die bekanntesten sind. Dazu gehören jedoch auch kleinere Projekte von RDF Systemen und Speichern [RDFProjekte]. Alle bestehenden Ansätzte haben gemeinsam, dass die Speicherung der RDF Daten mit einer objektrelationalen Datenbank realisiert wird. Jedoch ermöglichen nur RDFSuite und Sesame Anfragen in RQL und bilden das formale Graphenmodell somit auf Tabellen ab. Alle anderen Projekte unterstützen weniger mächtige oder noch keine RDF spezifischen Sprachen zur Anfrage.

Die Abbildung des formalen Graphen auf einen objektrelationalen Entwurf hat jedoch einige Nachteile.

Diese Arbeit beschreibt einen völlig anderen Ansatz. Die Idee ist es, ein die obengenannten Kriterien erfüllendes Speichersystem zu entwickeln ohne die Probleme zu haben, die durch die Abbildung des Graphen auf Tabellen entstehen. Diesen Anspruch erfüllt der Ansatz die Speicherung von RDF Daten in einem objektorientierten DBMS zu realisieren. Durch die direkte Abbildung des Graphen auf das objektorientierte Datenmodell entstehen große Vorteile gegenüber dem objektrelationalen, die ich im folgenden näher erläutern werde.

Dieser Vergleich bezieht sich jedoch nur auf strukturelle Aspekte. Die später beschriebenen Tests zeigen nämlich, dass in der Praxis noch andere Kriterien einen große Bedeutung bei der Wahl der Art der Abbildung des Graphen gewinnen.

Kurz RDF............

kurz oodb....

kurz sesame.........

kurz tests...........

# 2 Grundlagen von RDF

## 2.1 Einführung

Resource Description Framework (RDF) und RDF Schema sind als W3C Standard entwickelt worden, um die im Web zur Verfügung stehenden Quelldaten mit semantischen Daten anzureichern.
Durch diese Erweiterung entsteht das sogenannte Semantische Web[Semantic Web]. Dies ist also nicht nur eine Menge von Quellen, sondern bietet auch Beschreibungen der Quelldaten(sogenannte Metadaten). Die Kompatibilität und Verarbeitungsmöglichkeiten der Metadaten spielen eine zentrale Rolle für die vollständige Informationsverwertung.
RDF unterstützt die Entstehung und Verarbeitung von Metadaten der webbasierten Quellen durch Vorgabe einer festgelegten Struktur. Diese Struktur kann jedoch auch für beliebige Daten benutzt werden, da RDF über die Art der Daten keinerlei Vorgaben macht. So sind die Anwendungsbereiche vielfältig und die meisten Nachrichtenportale(z.B ABCNews, CNN, Time Inc.), Webportale(wie OpenDirectory, CNET ) und Browser(wie Netscape 6.0, W3C Amaya) unterstützen und akzeptieren RDF.
Da es sich hier jedoch um große Mengen von Daten handelt, reicht es nicht, dass die Daten nur zur Verfügung stehen. Es wird ein skalierbarer, persistenter RDF Speicher benötigt, um mit Hilfe einer mächtigen Anfragesprache die Daten effizient verarbeiten und verwerten zu können.

## 2.2 RDF und RDF Schema

Das Resource Description Framework (RDF) [RDF] wurde ursprünglich zur Standardisierung einer Notation von Metadaten über webbasierte Quellen entwickelt. Die W3C Empfehlung schlägt für diese Metadaten ein Datenmodell vor, welches auf einem gerichteten beschrifteten Graph basiert, dessen Knoten Quellen(engl. resourcen) und dessen Kanten Eigenschaften (engl. properties) genannt werden. Zusätzlich zum Datenmodell ist eine Syntax zur Kodierung und Austausch der Metadaten eingeführt worden. Diese Syntax benutzt XML. Die XML Syntax ermöglicht die RDF Metadaten in einer Form darzustellen, in der sie für Menschen und Maschinen lesbar und austauschbar sind .

Das RDFSchema [RDFSchema] erweitert die ursprünglich anwendungsneutrale Repräsentationssprache RDF um eine spezifische Terminologie. Entwickler haben die Möglichkeit ein Vokabular und Objektstrukturen des Modells genauer zu spezifizieren, um Teilen von RDFDaten eine bestimmte Semantik vorgeben zu können.

## 2.2.1 RDF

### 2.2.1.1 RDF Datenmodell

Das RDF Datenmodell ist ein syntaxneutraler Weg zur Repräsentation von RDF Ausdrücken. Es besteht aus drei Datentypen:

- Quellen (resources):
  Alle durch RDF Ausdrücke beschriebenen Dinge (z.B. ein gedrucktes Buch, eine Website etc.).
- Eigenschaften (properties):
  Eine Eigenschaft ist ein spezifischer Aspekt, Merkmal oder Beziehung zur Beschreibung einer Quelle.
- Ausdrücke (statements):
  Eine spezifische Quelle ergibt zusammen mit einer benannten Eigenschaft und deren Wert einen RDF Ausdruck. Diese Bestandteile werden Subjekt (Quelle), Attribut (Eigenschaft) und Objekt (Wert der Eigenschaft) genannt.

Das RDF Datenmodell repräsentiert Eigenschaften von Quellen und deren Werte als Attribut-Wert-Paar. Somit ist der RDF Ausdruck als Subjekt- Attribut -Objekt Tripel die Basisstruktur von RDF. Die Aussage des Subjekt-Attribut-Objekt Tripel ist:
Ein Subjekt S hat ein Attribut A mit Wert O.
Die Darstellung dieser Beziehung als Graph ist eine beschriftete Kante zwischen zwei Knoten:

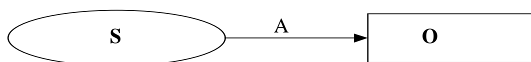



**Figure 1: Darstellung des Subjekt- Attribut -Objekt Tripels als Graph**

Es exestiert keine standardisierte Schreibweise des Subjekt-Attribut-Objekt Tripels. Das Sesame Projekt [Sesame] benutzt jedoch die Notation *Attribut(Subjekt, Objekt)*, welche im folgenden übernommen wurde.

**Beispiel 1: Ein einfaches Beispiel eines Subjekt- Attribut -Objekt Tripels**

**Daten:** *Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*
**Tripel :** *hasCreator(http://www.w3.org/Home/Lassila , Ora Lassila)*
**Graph:**

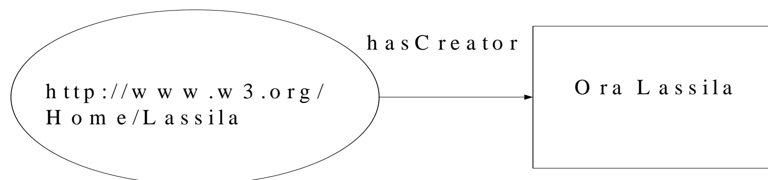



**Figure 2 : Graph zum Beispiel 1**

Das RDF Datenmodell unterstützt neben der Beschreibung von Quellen auch die Beschreibung von Beziehungen zwischen Quellen. Wenn der Wert einer Eigenschaft selbst wieder eine Quelle ist entsteht eine Beziehung zwischen zwei Quellen. Das bedeutet, dass ein Subjekt eines Tripels die Rolle des Objektes in einem anderen übernehmen kann, was die Verkettung zweier beschrifteter Kanten im Graph zur Folge hat.

**Beispiel 2: Ein Beispiel mit Beziehungen zwischen Quellen**

**Daten:** *The individual referred to by employee id 85740 is named Ora Lassila and has the email address lassila@w3.org.The resource http://www.w3.org/Home/Lassila was created by this individual.*
**Tripel :** *Name('http://www.w3.org/staffId/85740', "Ora Lassila")*
*Creator('http://www.w3.org/Home/Lassila', 'http://www.w3.org/staffId/85740' )*
*Email('http://www.w3.org/staffId/85740', "lassila@w3.org")*

**Graph:**

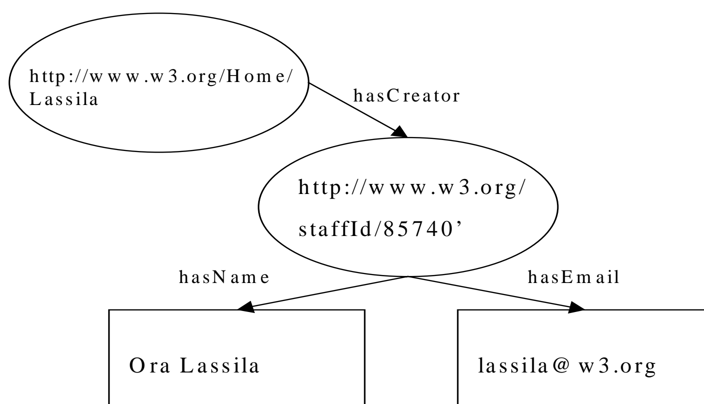



**Figure 3: Graph zum Beispiel 2**

RDF erlaubt es nicht nur Ausdrücke über Quellen, sondern auch Ausdrücke über andere RDF Ausdrücke zu machen. Dies wird ermöglicht, indem der ursprüngliche Ausdruck als eine neue Quelle mit zusätzlichen Eigenschaften modelliert wird. Die Eigenschaften sind die ursprünglichen Bestandteile des Ausdrucks (*subject, attribut, object*) und *type*. Die *type* Eigenschaft gibt den Typ der neuen Quelle(rdf:Statement) an. Der Prozess, bei dem ein RDF Ausdruck über einen anderen(in Form einer neuen Quelle) gemacht wird, nennt man Vergegenständlichung (reification). Dadurch können Zweifel oder Unterstützung von anderen Quellen bezüglich eines Ausdrucks dargestellt werden. Die Aussage des ursprünglichen Ausdrucks ist dabei nur gültig und existent in Verbindung mit dem übergeordneten Ausdruck.

**Beispiel 3**: **Ein Beispiel zur Vergegenständlichung**

**Daten***: Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

**Graph:**

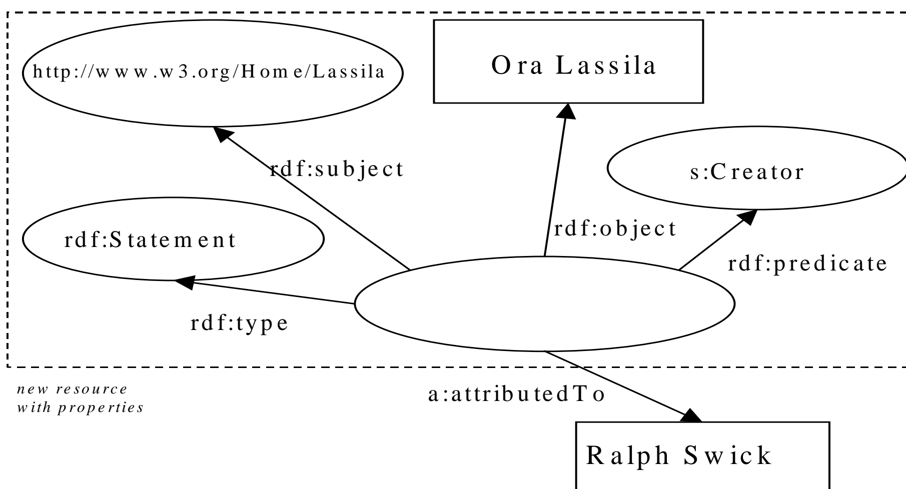



**Figure 4**: Graph zum Beispiel 3

### 2.2.1.2 RDF Syntax

Das RDF Datenmodell stellt ein abstraktes konzeptionelles Gerüst zur Definition und Benutzung von Metadaten zur Verfügung. Es wird jedoch eine spezifizierte Syntax gebraucht, um die Metadaten austauschen zu können. Für diesen Austausch sieht die Spezifikation von RDF des W3C [RDF] eine Kodierung in XML Syntax(definiert in EBNF) vor.
Eine mögliche Serialisierung von Beispiel 3 in XML Syntax könnte so aussehen:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:a="http://description.org/schema/">
 <rdf:Description>
  <rdf:subject resource="http://www.w3.org/Home/Lassila" />
  <rdf:predicate resource="http://description.org/schema/Creator" />
  <rdf:object>Ora Lassila</rdf:object>
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
  <a:attributedTo>Ralph Swick</a:attributedTo>
 </rdf:Description>
</rdf:RDF>
```

**Figure 5: XML Syntax zum Beispiel 3**

Es wurden zwei XML Syntaxen definiert:

1. Die Serialisierungssyntax drückt die volle Kapazität des Datenmodells aus.
2. Die abgeleitete Syntax repräsentiert das Datenmodell in kompakterer Form.

Diese beiden Syntaxen sind beliebig mischbar. Dies bedeutet, dass viele Alternativen der Darstellung eines RDF Modells möglich sind und die oben vorgestellte Syntax nur ein Beispiel von vielen Wegen ist diese Beispieldaten in XML niederzuschreiben .

Aber auch die Wahl der Mark-Up-Sprache XML als Syntax ist nur eine Möglichkeit zur Darstellung des RDF Datenmodells.

RDF ist anwendungsneutral, denn es definiert nur eine Struktur zur Beschreibung von Quellen. Es exestieren keine reservierten Terme für die Datenmodellierung. Das bedeutet, dass das RDF Datenmodell keinen Mechanismus bietet, um Namen von Eigenschaften zu deklarieren. Aus diesem Grund wurde das RDF Schema entwickelt.

### 2.2.2 RDF Schema

Ein RDF Schema ist ein Mechanismus um Objekttypen und Attributnamen von RDF Ausdrücken zu definieren. Dadurch wird ein Basistypsystem spezifiziert. Dieses Typsystem kann jedoch selbst wieder auf einem anderen basieren. Für jede Definition eines Typsystems kann ein anderes Schema benutzt werden. Beispiele verschiedener Schemata können unter [Schemata] gefunden werden.

Das vom W3C vorgeschlagene RDFSchema [RDFSchema] spezifiziert z.B. eine Terminologie wie *Class* und *Property* als Objekttypen und *subClassOf* und *subPropertyOf* als Eigenschaftsnamen. Dies sind wichtige RDFSchema Konstrukte, denn RDF Objekte können Instanzen von einer oder mehreren Klassen sein, was durch die *type* Eigenschaft angezeigt wird. Die *subClassOf* Eigenschaft erlaubt die Spezifikation von einer hierarischen Organisation von diesen Klassen. *subPropertyOf* macht das selbe für Eigenschaften. So ermöglicht RDFSchema

ein hierarisches Klassensystem, wie viele andere objektorientierte Programmierungs- und Modellierungssysteme . Außerdem können für Eigenschaften Benutzungsbeschränkungen durch Anwendung der Konstrukte *range* und *domain* definiert werden.

RDF Schema Ausdrücke sind demnach ebenso gültige RDF Ausdrücke. Der einzige Unterschied zu den reinen Subjekt-Attribut-Objekt Tripeln ist, dass durch ein RDF Schema eine Vereinbarung über die Semantik von reservierten Terme und über die Interpretation von bestimmten Ausdrücken getroffen wurde. Das RDF Schema ermöglicht also die Erweiterung des Vokabulars und die Definition einer gezielten Interpretation von RDF Ausdrücken.

**Beispiel 4: RDFSchema zum Beispiel 1**

**Daten:** *Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*
**Tripel :** *type( Resource Class)*
*type( Creator Class)*
*type( FamousCreator Class)*
*subClassOf( FamousCreator Writer)*
*type( hasCreator Property)*
*domain( hasCreator Creator)*
*range( hasCreator Creator)*
*type( Home/Lassila FamousCreator)*
*type( staffId/85740 Resource)*
*hasCreator( Home/Lassila staffId/85740)*

**Graph:**

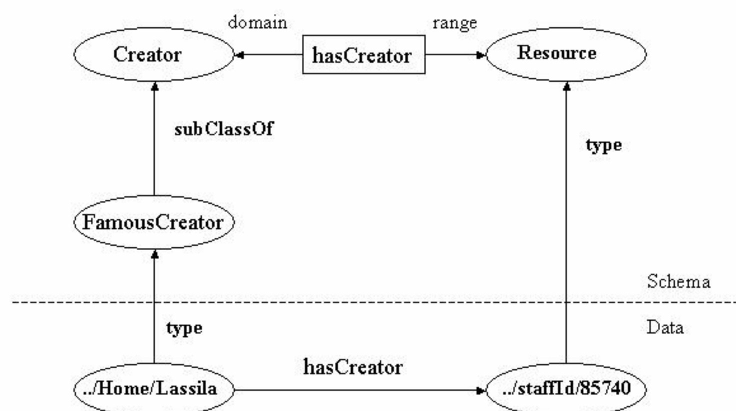



**Figure 6 : Graph des RDFSchematas zum Beispiel 1**

## 2.3 RDF Anfragesprache und Speicher

RDF Daten können als XML Dokument (auf der syntaktischen Ebene), als eine Menge von Tripeln(auf der strukturellen Ebene) und als Graph (auf der semantischen Ebene) vorliegen. Die Art der Speicherung ist für die Anfragemöglichkeiten entscheidend. Im folgenden werden die drei verschiedenen Abstraktionsebenen bezüglich der Anfrageformen untersucht.
Die Ebene, die sich am besten für Anfragen eignet wird dann die Speicherungsart der Daten bestimmen.

### 2.3.1 Anfragen auf der syntaktischen Ebene

Wie in Abschnitt 2.2.1.2 beschrieben kann jedes RDF Modell und so auch jedes RDF Schema in XML Notation ausgedrückt werden. So scheint es sinnvoll für RDF Anfragen eine XML Anfragesprache zu benutzten (z.B XQuery [XMLQuery]).
Da bei dieser Lösung die Tatsache vernachlässigt wird, dass RDF nicht nur ein XML Dialekt ist, sondern sein eigenes Datenmodell hat, entstehen folgende Probleme:

- RDF bildet einen Graph, was bedeutet, dass die dort enthaltenen Beziehungen in der Baumstruktur von XML nur schwer erkennbar und damit nur schwer anzufragen sind. XML Anfragesprachen wie XQuery [XMLQuery] formulieren Ausdrücke zur Traversierung der Datenstruktur eines Baums mit beschrifteten Knoten und unterstützen keine Verlinkung innerhalb des Baums. Der Graph des RDF Datenmodells ist jedoch kein Baum und neben den Kanten(Attribute) sind auch die Knoten(Subjekte/Objekte) beschriftet.

- Anfragen von Beziehungen auf der syntaktischen Ebene sind nicht möglich ohne die Syntax zu kennen, mit der die RDF Daten in XML kodiert wurden. Bestehen solche Beziehungen nur indirekt über die Beschreibung einer anderen Quelle, so müssten Syntaxelemente in Verbindung gesetzt werden, was ohne Kenntnis der Syntax unmöglich ist.
  Betrachtet man die XML Beschreibung des RDF Modells von Beispiel 2:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"/>
  </rdf:Description>

  <rdf:Description about="http://www.w3.org/staffId/85740">
    <v:Name>Ora Lassila</v:Name>
    <v:Email>lassila@w3.org</v:Email>
  </rdf:Description>
</rdf:RDF>
```

  Das Beispiel zeigt, dass man die Beziehung zwischen dem Objekt '*http://www.w3.org/Home/Lassila*' und '*lassila@w3.org*' nicht anfragen kann, ohne die Syntax zu kennen.

Die Formulierung einer Anfrage auf der syntaktischen Ebene benötigt also immer die Kenntnis von der zur Kodierung benutzten Syntax.

- Geht man davon aus, dass die Syntax bekannt ist besteht ein weiteres Problem durch die Variabilität dieser. Möchte man eine Anfrage formulieren wie
  "*Gib mir alle existierenden Beziehungen zwischen Lassila und lassila@w3.org*"
  so könnte man die XML Syntax nicht dazu benutzten um zu fragen
  "*Gib mir alle Elemente die eingebettet sind in ein `Description` Element mit einem `about` Attribut mit dem Wert ' http://www.w3.org/Home/Lassila', dessen Wert des Attributs wieder irgendwo anders als `about` Attributwert eines `Description` Elements auftaucht, welches ein eingebettetes Element `Email` mit dem Wert 'lassila@w3.org' ist.*",
  da die Daten nicht in dieser Form kodiert sein müssen.
  Die XML Syntax für RDF ist nicht eindeutig ist. Verschiedene Wege (siehe Abschnitt 2.2.1.2) zur Kodierung der selben Information in XML sind möglich und werden benutzt. Die und jeder andere Anfrage wird folglich niemals garantieren, daß man eine oder alle Antworten des RDF Modells erhält.

Die dargestellten Probleme zeigen deutlich, dass es nicht nur unbequem, sondern auch sehr schwierig ist die Daten auf der syntaktischen Ebene vollständig und fehlerfrei anzufragen.

### 2.3.2 Anfragen auf der strukturellen Ebene

Jedes RDF Dokument repräsentiert eine Menge von Tripeln, und jedes Tripel repräsentiert eine Aussage in Subjekt–Attribut–Objekt Form. Für in dieser Form dargestellte RDF Daten gibt es verschiedene Vorschläge von Anfragesprachen, deren Implementationen und Vergleich unter [RDF-Query] gefunden werden können.
Alle Beispiele von Anfragen in diesem Abschnitt sind in Squish [Squish]formuliert. Squish ist eine RDF Anfragesprache, die SQL-ähnliche Anfragen ermöglicht.

Betrachtet man die Menge der Tripel des RDF Modells von Beispiel 4:
*(type Resource Class)*
*(type Creator Class)*
*(type FamousCreator Class)*
*(subClassOf FamousCreator Writer)*
*(type hasCreator Property)*
*(domain hasCreator Writer)*
*(range hasCreator Writer)*
*(type Home/Lassila FamousCreator)*
*(type staffId/85740 Resource)*
*(hasCreator Home/Lassila staffId/85740)*

Mit Hilfe von Squish ließe sich fragen, welche Quellen vom Typ *FamousCreator* bekannt sind:
*select ?x from ... where (type ?x FamousCreator)*

Der Vorteil einer so formulierten Anfrage ist, dass sie direkt das RDF Datenmodell in der Tripelstruktur anfragt. Die Anfrage ist damit unabhängig von der für die Repräsentation gewählten spezifischen XML Syntax.
Ein Hauptmangel der Anfragesprachen auf dieser Ebene ist aber, dass das RDF Dokument nur als Menge von einzelnen Tripeln interpretiert wird. Sind einzelne RDF Tripel durch eine spezielle Semantik eines RDF Schematas verbunden, so wird dies nicht interpretiert, weil die Tripel einzeln betrachtet werden. Daraus entsteht folgendes Problem:
Sei *http://../Home/Lassila* vom Typ *FamousCreator* und *FamousCreator* sei eine Unterklasse von *Creator*.
Aufgrund der RDFSchema Semantik von `*type*` und `*subClassOf*` folgt, dass somit auch *http://../Home/Lassila* vom Typ *Creator* ist. Aber es gibt kein Tripel, welches explizit diese Tatsache ausdrückt. Somit würde das Ergebnis der Anfrage:
*select ?x from ... where (type ?x Creator)*

fehlschlagen, obwohl das Tripel *(type Home/Lassila Creator)* impliziert wird durch die Semantik des RDF Schematas.
Die Erweiterung der Anfrage zu:

*SELECT ?x ?c1 ?c2 ?c3*
*FROM (type ?x ?c1),*
*(subClassOf ?c2 ?c3)*
*WHERE ?c1 = ?c2*

würde das Problem in diesem konkreten Fall lösen, denn hier wird explizit nach Daten gesucht, die eine Stufe tiefer in der Klassenhierarchie liegen. Bei einer weiteren Verkettung von *subClassOf* Tripeln wäre diese Anfrage aber schon nicht mehr anwendbar, da sich die Daten noch tiefer befinden würden.
Somit können Anfragen auf einfache Art und Weise Daten einer Menge von RDF Tripeln extrahieren, es entstehen jedoch Probleme, wenn zur Anfrage die Interpretation eines RDF Schematas nötig ist.

### 2.3.3 Anfragen auf der semantischen Ebene: RQL

RQL ist eine getypte Sprache, deren Syntax OQL [OQL] angelehnt ist. Sie wurde in Zusammenhang mit dem European IST Projekt C-Web und dem Folgeprojekt MESMUSES des Instituts der Informatik in FORTH, Griechenland entwickelt.

Meinem Wissen nach ist RQL [RQL] die einzige deklarative Anfragesprache für RDF auf der semantischen Ebene. Das bedeutet, dass die Anfragen sensitiv gegenüber der Semantik des benutzten RDF Schematas sind. Im Gegensatz zu allen anderen auf Tripeln basierenden Anfragesprachen basiert RQL auf einem formalen Graphenmodell. Es repräsentiert die RDF Tripelstruktur als Graph, wie in den vorhergehenden Abschnitten gezeigt. Damit wird die Interpretation der Bedeutung der RDF Schemata ermöglicht. Somit kombiniert RQL Anfragen auf Schema und Daten.

Das folgende Beispiel zeigt, dass die Struktur der Klassenhierarchien erkannt und umgesetzt wird.
Die Anfrage *subClassOf(Creator)* gibt alle direkten und indirekten Unterklassen von Creator zurück, das heißt *FamousCreator* erhält man als Ergebnis.

Eine entscheidene Besonderheit von RQL sind die Pfadausdrücke. Diese erlauben es entlang vollständiger Pfade von RDF Graphen Muster zu suchen. Folgende Anfrage für Beispiel 4:

*SELECT Y FROM FamousCreator {X}. hasCreator{Y}*

gibt alle von 'FamousCreators' geschaffenen Resourcen zurück, indem Mustersuche entlang des Graphen Figure 6 durchgeführt wurde.

## 2.4 Zusammenfassung

Die Erläuterung der Grundlagen von RDF und der Bedeutung der RDF Schemata ermöglichten die Darstellung einer Auswahl von drei verschiedenen Speicherungsformen.

Aus dieser folgt, dass RDF Daten nicht auf der Ebene der XML Syntax und nicht nur als Menge von RDF Tripeln betrachtet werden sollten, da damit die beabsichtigte Semantik des RDF Schematas verloren geht. So ist nur eine Anfragesprache sinnvoll, die sensitiv gegenüber der RDF Schema Semantik ist. Mometan ist RQL der einzige Kandidat für eine solche Sprache.
Da RQL auf einem formalen Graphenmodell basiert, muß der angefragte RDF Speicher dieses unterstützen. Alle RDF Schema Informationen müssen verwertet und umgesetzt werden können, um effizientes Laden und Anfragen mit RQL in einem DBMS zu ermöglichen.

# 3 Speicherung von RDF Daten

## 3.1 Einführung

Die Entwicklung eines RDF-Speichersystems auf Basis einer objektorientierten Datenbank erfordert die Kenntnis über die Struktur von dem die Semantik enthaltenden Graph, der sich aus den RDF Daten bildet.

Aufgrund der Spezifikation des formalen Graphenmodells wird dann in Kenntnis der objektorientierten Konzepte ein Speicherentwurf entwickelt werden. Dieser Entwurf ist die Abbildung des semantischen Graphen auf eine Speicherstruktur. Die Vorteile, die sich für das gesamte System und im Speziellen für die Anfragen an den Graph, ergeben werden dann die Basis für die Umsetzung meines Ansatzes sein.

## 3.2 Spezifikation des formalen Graphenmodells in RDF

Das formale Gaphenmodell einer Menge von RDF Ausdrücken(Tripeln) entsteht durch Interpretation der Eigenschaft(Prädikat) als Verweis von einer(Subjekt) auf die andere Quelle(Objekt) siehe Abschnitt 2.2.1.1. Jedoch erst durch Hinzunahme der RDF Ausdrücke, die ein Schema repräsentieren, erhält das formale Graphenmodell eine spezifizierte Semantik siehe Abschnitt 2.2.2. Die Definition der Semantik des Datenmodells muß formal, explizit und eindeutig sein, um dieses anfragen zu können. Die in Referenz 10: [RDFSuite] vorgeschlagene Formalisierung des RDF Datenmodells klärt einige Mehrdeutigkeiten der offiziellen [RDF] und [RDFSchema] Spezifikationen. Der Vorschlag von RDFSuite enthält einige Restriktionen hinsichtlich der RDF und RDFSchema Semantik, die direkte syntaktische Konsequenzen haben. Einer der wichtigesten Punkte der Formalisierung ist die Forderung nach expliziter und eindeutiger Definition von *domain* und *range* für jede Eigenschaft. Das bedeutet keine oder mehrere Domains für eine Eigenschaft sind nicht erlaubt. Auch in der Festlegung von *rdfs:Resource* und nicht *rdfs:Class* als die Wurzel jeder Klassenhierachie unterscheiden sich die Ansätze von [RDFSuite] und [RDFSchema].

Diese Entscheidungen verändern nicht nur die Semantik, sondern haben auch einen großen Einfluß auf das Graphenmodell, welches angefragt werden soll. Für überschaubare Bereiche stellt dies kein Problem dar, jedoch für generelle Anwendungen eines solchen Modells bedeutet dies Probleme, wenn keine Kenntnis über das Verständnis des Schematas vorhanden ist.

Um trotz der beschriebenen Definitionsprobeme ein formales Graphenmodell von einer Menge von Tripeln spezifizieren zu können, werde ich die Annahmen von [ Sesame: RQL] übernehmen.

Die für den Graph entscheidenden, sind:

- **Alle Quellen sind vom Typ *rdfs:Resource*.**

Das im Abschnitt 2.2.1.1 eingeführte RDF Datenmodell besagt, dass alle durch RDF Ausdrücke beschriebenen Dinge Quellen genannt werden. Unter Beachtung des RDFSchema werden diese Quellen nun als Instanzen vom type *rdfs:Resource* genauer definiert.

- **Alle literarischen Werte sind vom Typ *rdfs:Literal.***

  Alle sich selbst beschreibenden Werte werden Literale genannt und gehören zu einer Klasse von *rdfs:Literal*. Eigenschaften und Quellen mit Attributen gehören nicht zu *rdfs:Literal*. Literale sind z.B. Werte wie reine Textbeschreibungen von Eigenschaften von Quellen. Für das Verständnis eines Tripels gilt somit, dass das Subjekt eines Tripels von einer Quelle dargestellt wird, und das Objekt sowohl Quelle(siehe Seite 6) als auch ein Literal sein kann.

  Eine Ausnahme bilden die Tripel, die das RDFSchema repräsentieren. Innerhalb dieser Tripelmenge ist ein Literal auch Subjekt eines Tripels, um Eigenschaften der Literale innerhalb des Schematas definieren zu können. Ein Beispiel ist *rdf:type(xxx rdfs:Literal).*

- ***rdfs:Resource* und *rdfs:Literals* sind disjunkte Klassen.**

  *rdfs:Resource* und *rdfs:Literal* sind unterschiedliche Klassen ohne, dass eine Unterklasse der anderen ist.

- ***rdfs:Resource* und *rdfs:Literals* sind implizit Teil aller Ontologien.**

  Diese Annahme lässt sich aus den ersten beiden ableiten. Wenn alle Quellen vom Typ *rdfs:Resource* sind, dann muss diese Klasse in allen Ontologien existieren. Dasselbe gilt auch für *rdfs:Literals.*

- ***rdfs:Resource* ist die Superklasse von allen anderen Klassen( außer *rdfs:Literals* und seinen Unterklassen).**

  Alle Mengen einer spezifischen Klasse sind immer eine Untermenge von rdfs:Resource, da alle Quellen Instanzen vom Typ rdfs:Resource sind. Dies läßt sich rekursiv auf alle Ebenen der jeweiligen Klassenhierarchie anwenden. Für Literale gilt dies natürlich nicht, weil die Klassen disjunkt sind.

Die folgenden Annahmen über das RDFSchema stellen Abweichungen von der Spezifikation dar. Die Begründung dieser sind unter ??? zu finden.

- ***rdfs:domain***

  Die RDF Schema Spezifikation sagt, dass, wenn es mehrere *domain* - Bedingungen zur Beschränkung einer Eigenschaft gibt, jede Instanz von einer beliebigen der Klassen, die Werte für diese Bedingungen sind, benutzt werden kann. Es wird also eine Vereinigungssemantik auf die verschiedenen *domain* - Bedingungen angewandt. Im Gegensatz dazu wird nach den Annahmen von SESAME die *domain* einer Eigenschaft als Schnitt von den *domain* - Klassen verstanden. Wenn verschiedene *domain* - Bedingungen definiert sind, kann die Eigenschaft nur auf Quellen angewandt werden, die Instanzen von allen diesen *domain* - Klassen sind.

- ***rdfs:range***

Die RDF Schema Spezifikation sagt, dass eine Eigenschaft höchstens eine *range* - Bedingung haben kann. Die hier übernommenen Annahmen erlauben jedoch verschiedene *range* - Bedingungen, die genauso behandelt werden, wie oben beschriebene *domain* - Bedingungen.

Die folgende Abbildung zeigt einen Graphen der auf diesen Annahmen beruht.

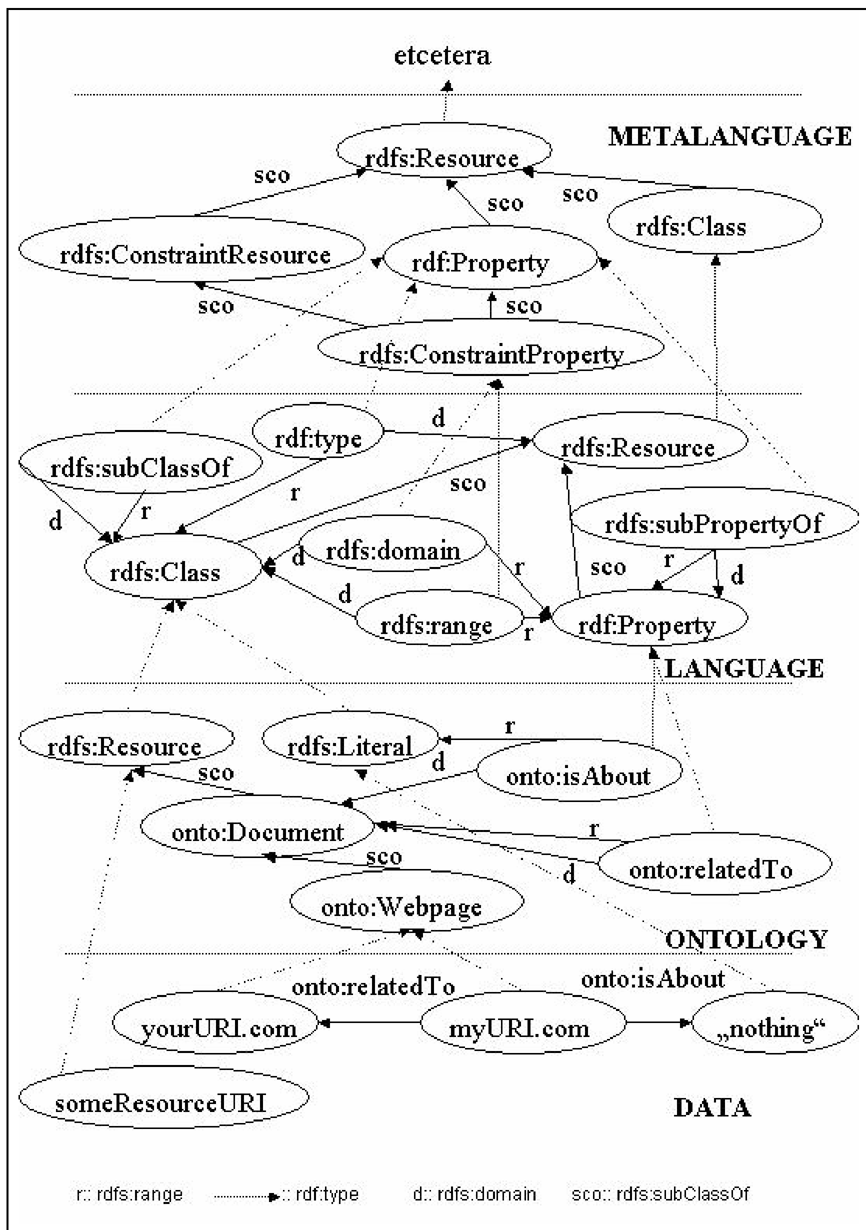


**Figure 7 : Beispielgraph bezüglich der gemachten Annahmen.**

Die vorhergehenden Annahmen haben hinsichtlich der Struktur des Graphen folgende Bedeutung:

Alle Knoten des Graphen sind entweder Literale oder Quellen und somit auch Instanzen vom type *rdfs:Literals* oder *rdfs:Resource*. Da *rdfs:Resource* Superklasse aller anderen ist, so ist sie auch Superklasse von *rdf:property*. Dies bedeutet, dass die sematische Unterscheidung zwischen Eigenschaft und Quelle strukturell nicht gemacht wird. Somit sind auch die Kanten des Graphen Quellen und damit Instanzen von *rdfs:Resource.*

Durch die Definition der Kanten und Knoten nach dem RDFSchema ist die Struktur des Graphen somit eindeutig spezifiziert. Zu Beachten ist jedoch, dass diese Spezifikation nicht nur stark abhängig vom benutzten Schema ist, sondern nur auf dieser Basis existiert.

Der Einfluß des benutzten Schematas auf den Graphen ist ebenso stark wie auf die Abbildung des Graphen auf eine Speicherstruktur.

## 3.3 Überblick über die Eigenschaften Objektorientierter Datenbanken

Um beuteilen zu können, welche Vorteile die Verwendung einer objektorientierten Datenbank als Speicher für den in dem vorherigen Abschnitt spezifizierten Graph hat, wird in diesem Abschnitt ein kleiner Überblick gegeben.

Nach dem Vorschlag des [Manifesto] zur Standardisierung von Objektorientierten Datenbanksystemen (OODB) gibt es drei Arten von Merkmalen, die OODB's charakterisieren. Für das Verständnis des Konzepts sind jedoch nur die Merkmale wichtig, die erfüllt sein müssen, um überhaupt von einem Objektorientierten Datenbanksystem zu sprechen:

- **Komplexe Objekte**
  Ein OODBS unterstützt komplexe Objekte, deren Attribute selbst wieder Objekte, Listen oder Mengen von Objekten sein können.
- **Objektidentität**
  Jedes Objekt trägt unabhängig von den Werten der Attribute eine eindeutige und unveränderbare Identität.
- **Kapselung**
  Auf Attributswerte kann nur durch Methoden zugegriffen werden.
- **Typen und Klassen**
  Objekte mit äquivalenter Struktur und gleichem Verhalten werden zu Klassen zusammengefasst.
- **Typen- und Klassenhierarchie**
  Typen und Klassen können hierarchisch aufgebaut sein.
- **Überschreiben, Überladen, und spätes Binden**
- **Berechnungsvollständigkeit**
  Jede Operation auf der Datenbank kann durch die Datenmanipulationsspracheausgedrückt werden.
- **Erweiterbarkeit**

Das System kann um neue benutzerdefinierte Typen und Klassen ergänzt werden.

- **Persistenz**
  Der Zustand der Datenbank muss über die Laufzeit eines Programms hinweg erhalten bleiben.
- **Sekundärspeicher-Verwaltung**
  Alle die Performance betreffenden Eigenschaften wie Index Verwaltung, Zwischenspeicherung von Daten usw. Sollten bereitgestellt werden.
- **Nebenläufigkeit**
  Die Möglichkeit der Serialisierung von Operationen sollte gegeben sein.
- **Recovery**
- **Ad-hoc-Abfragemöglichkeit**
  Das OODBS beinhaltet eine Sprache, die nicht triviale, effiziente und anwendungsunabhängige Anfragen auf die Daten ermöglicht.

Eine Datenbank, die diese Eigenschaften erfüllt, ermöglicht es den Graphen direkt abzubilden und einfach anzufragen.

## 3.4 Vorteile und Darstellung der Abbildung des RDF-Graphen auf ein objektorientiertes Datenbankschema

Wird eine OODB als Speicherstruktur für die Abbildung des Graphen genutzt, so können alle Kanten und Knoten von diesem durch komplexe Objekte realisiert werden. Der Graph entsteht nun durch die Verweise der Objekte aufeinander. Es wird der Graph somit direkt abgebildet, denn die Objekte werden persistent gespeichert.

Da Knoten und Kanten entweder Quellen oder Literale sind, basiert mein Datenbankentwurf auf diesen beiden Objekttypen. Ein Tripel entsteht durch den Verweis des „Subjekt"-Objektes über ein „Prädikat"-Attribut auf ein „Objekt"-Objekt. Subjekt, Objekt und Prädikat sind selbst Instanzen von einem der Objekttypen.

Die Abbildung der Objekte und ihrer Verweise in einer OODB für den Graph von Figure 6 folgt.

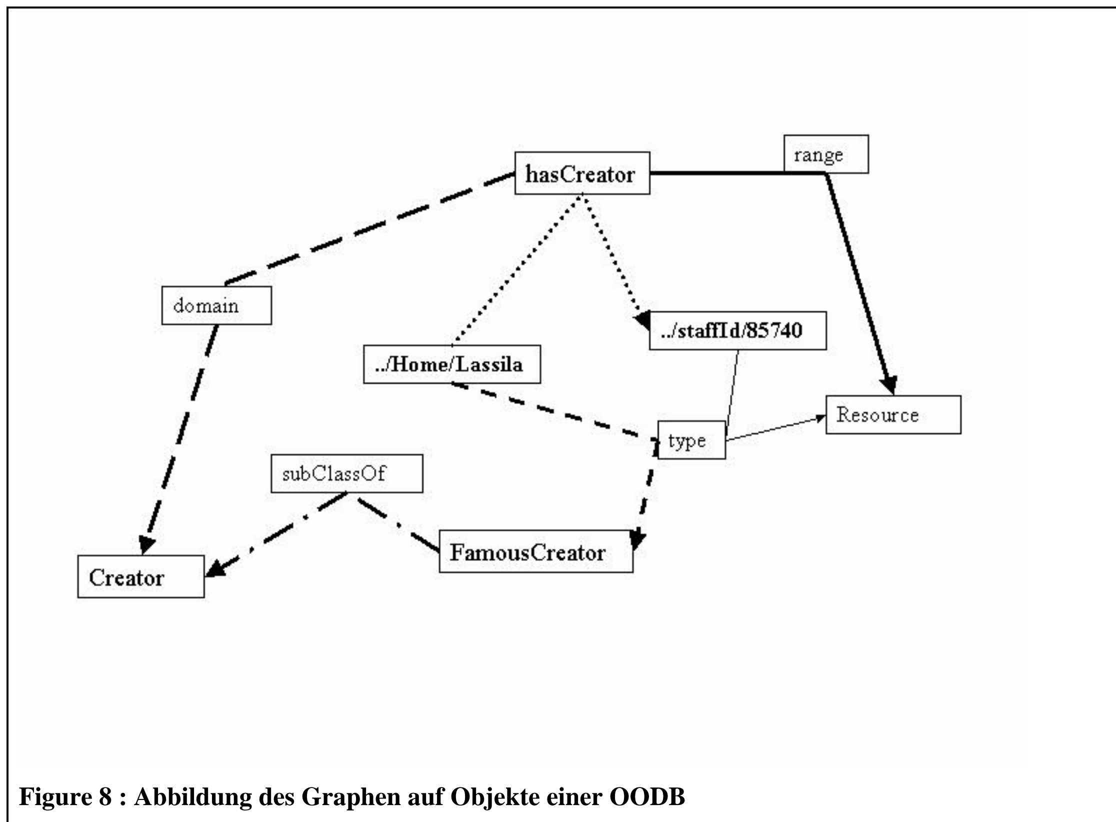



**Figure 8 : Abbildung des Graphen auf Objekte einer OODB**

Die Repräsentation des Graphen im Speicher als persistente Objekte hat große strukturelle Vorteile:

- Der Speicherstrukturentwurf wird stark vereinfacht, da eine direkte Abbildung des Graphen auf Objekte möglich ist, wenn das formale Graphenmodell bekannt ist. Alle Knoten und Kanten können einfach als Objekte modelliert werden. Die Beziehungen von Knoten und Kanten untereinander entstehen durch Verweise. Die Speicherstruktur ist zwar vom Schema stark abhängig, jedoch für alle möglichen verschiedenen einfach zu realisieren.
- Die Nähe der Struktur des Speichers zu der des Graphen beeinflußt das gesamte Speichersystem positiv. Alle möglichen Operationen auf dem semantischen Graphen der RDF Tripel sind ohne großen Übersetzungsaufwand auf dem gespeicherten Objektgraphen gleichermaßen ausführbar.
- Dieser Einfluß gilt im Besonderen für die RQL Anfragen. RQL ist eine getypte und OQL sehr ähnliche Sprache. Somit sind alle RQL Anfragen denen in OQL strukturell sehr nahe und fast ohne weitere Anpassung anwendbar. Es gibt zwei Arten von RQL Anfragen. Es können einmal die durch die Semantik des Schematas entstandenen Strukturen und Hierarchien erfragt werden. Die zweite Art von RQL Anfragen sind die select-from-Anfragen, welche die Zusammenhänge einzelner Tripel erfragen. Beide Arten von Anfragen bedeuten auf dem Graph, dass von einem Knoten aus die, durch die abgehenden Kanten gebildeten, Pfade durchlaufen werden müssen. Die Suche in dem Graph kann durch das Verfolgen der Verweise der Objekte im Speicher direkt vollzogen

werden. Dies ist ein großer Vorteil im Gegensatz zur Abbildung auf eine objektrelationale Datenbank. Da der semantische Graph nicht im Speicher abgebildet ist, sondern als Datensätze einer Tabelle gespeichert ist, muss die in den Tripeln enthaltene Information des Graphen bei Pfadanfragen immer wieder erneut zusammengesetzt werden. Genaueres in Abschnitt **Fehler! Textmarke nicht definiert.**.

## 3.5 Zusammenfassung

Die in dem semantischen Graph strukturierten Daten können also durch das RDF Schema innerhalb eines hierarchischen Klassen- und Typsystems im Sinne der Objektorientierung in Objekte gegliedert werden. Dies hat die eben beschriebenen Vorteile. Folglich besitzt das RDF Datenmodell unter Verwendung des spezifizierten RDF Schematas [RDFSchema] die Merkmale eines Objektorientierten. Die Speicherung von RDF in einer objektorientierten Datenbank entspricht damit der Struktur von RDF Daten.

Die Entwicklung eines Systems, welches die Speicherung von RDF unter Erhaltung möglichst großer Flexibilität, Datentransformation und –anfrage ermöglicht, ist mit einer objektorientierten Datenbank umsetzbar und im folgenden Abschnitt beschrieben.

# 4 Implementierung

## 4.1 Einführung

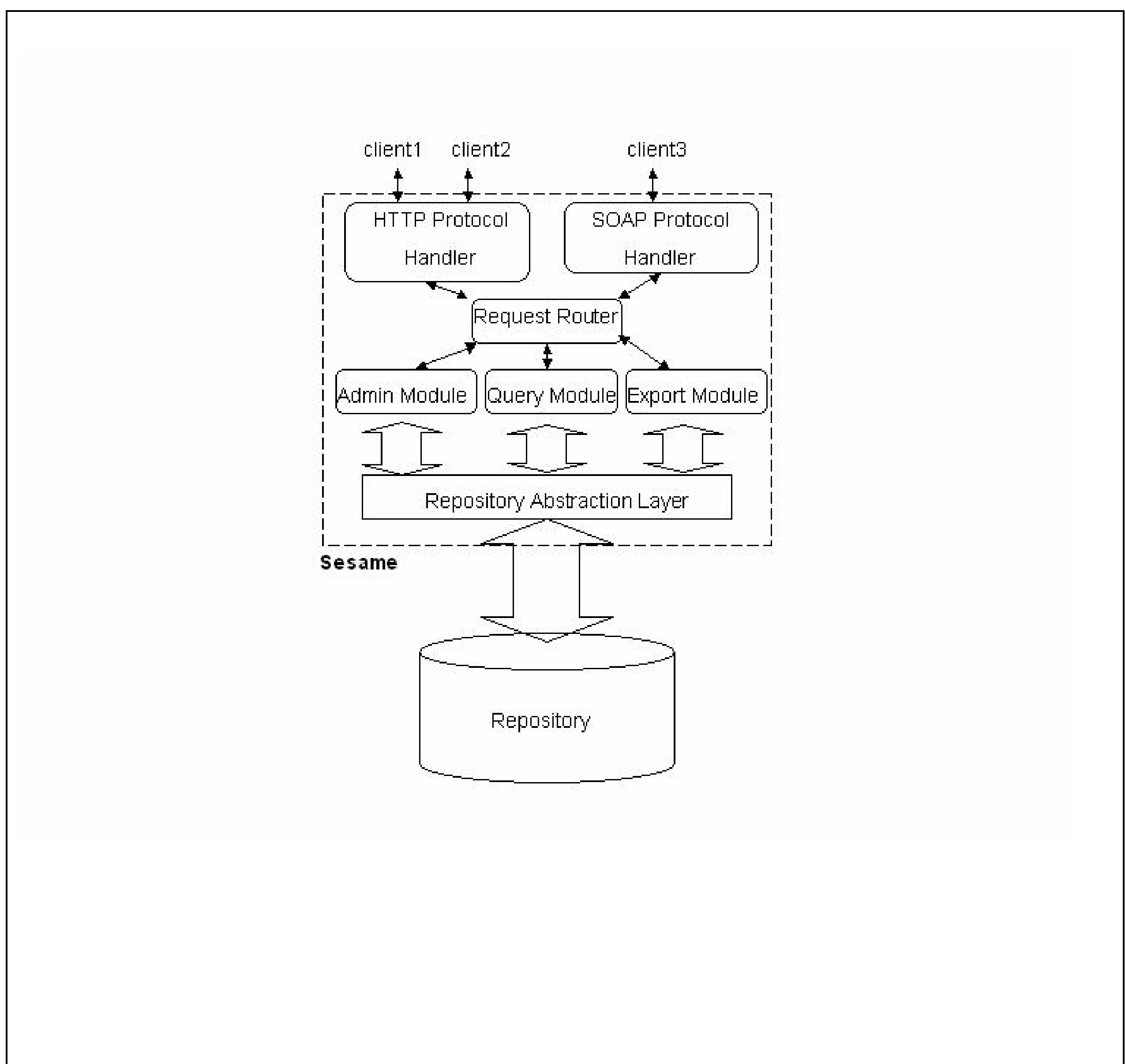
Das vorhergehende Kapitel stellt eine Interpretation von RDF/S Daten als formales Graphenmodell vor und beschreibt die Vorteile, die sich aus der Speicherung eines solchen Graphen mit einer objektorientierten Datenbank ergeben. Die praktische Umsetzung eines RDF Systems mit einem objektorientierten Speicher schien anfänglich im Rahmen einer Diplomarbeit zu komplex. Bei näherer Betrachtung bestehender Systeme ergab sich dann jedoch eine Möglichkeit unter Verwendung des Sesame Projekts. Das Merkmal von Sesame von der Form des Datenspeichers unabhängig zu sei, ermöglichte mir die Umsetzung meiner Idee innerhalb dieses bestehenden RDF Speicher- und Anfragesystems.

Im folgenden Abschnitt wird Sesame als Projekt, sowie eine seiner ursprünglichen Implementierungen für eine MySQL-Datenbank vorgestellt. Darauf folgt die Beschreibung meiner Implementierung für eine objektorientierte Datenbank. Der dritte Abschnitt enthält den konzeptionellen Vergleich beider Implementierungen und bewertet den neuen Ansatz im Vergleich zu den bestehenden.

## 4.2 Sesame: Eine generische Speicher- und Anfragearchitektur für RDF und RDF Schema

Sesame wurde als eine effiziente Speicher- und Anfragearchitektur für große Mengen von RDF und RDFSchema Daten im Rahmen des Europäischen IST Projekts On-To-Knowledge (1999) von Aidministrator Nederland BV entwickelt. Einen großen Vorteile gegenüber anderen Systemen bildet die Unabhängigkeit des Systementwurfs und der Implementierung von der Speicherform. Sesame kann auf verschiedene Speicherungsformen aufgesetzt werden, ohne das System z.B das Anfragewerk verändern zu müssen. Das Anfragewerk von Sesame implementiert RQL.

Ein Überblick über die Architektur von Sesame ist in der Figur9 zu sehen.



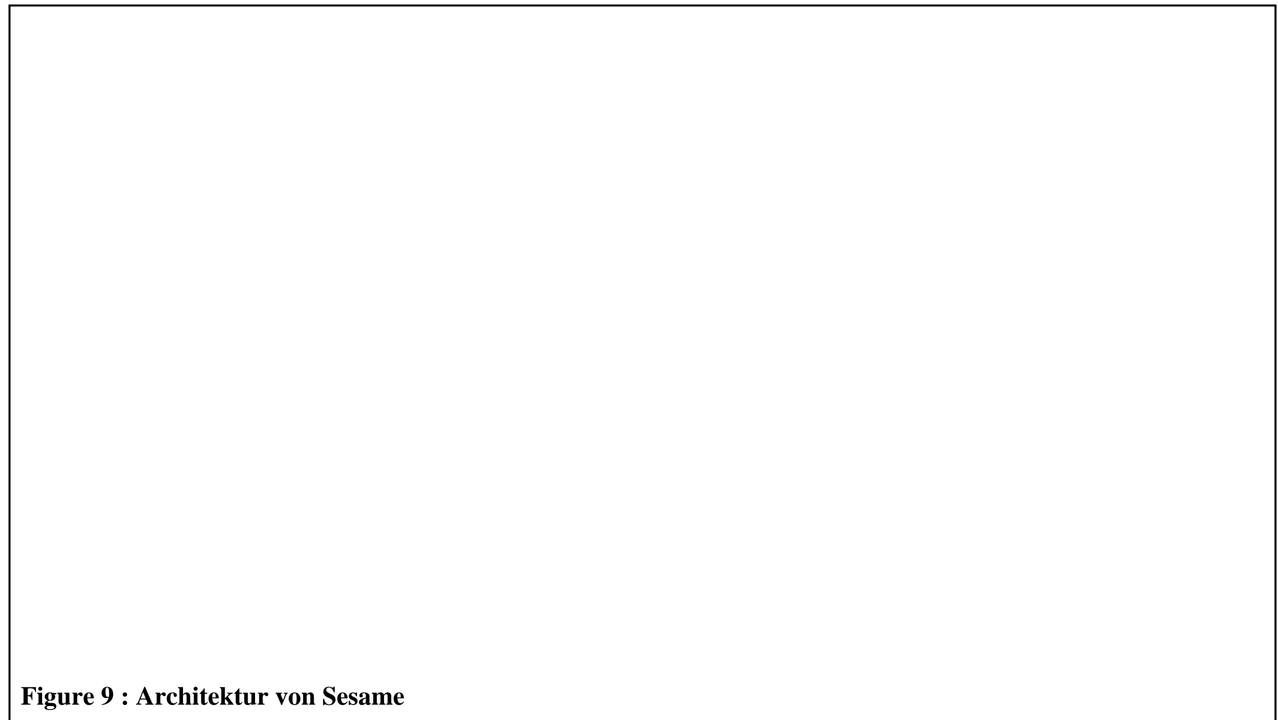


**Figure 9 : Architektur von Sesame**

Im folgenden werden die einzelnen Komponenten des Systems näher erläutert.

Um große Mengen von Daten zu speichern benutzt Sesame als persistenten und skalierbaren Datenspeicher('Repository') ein Datenbanksystem. Eine genauere Beschreibung des Datenspeichers ist im Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** zu finden.

Das zentrale Merkmal Sesame's ist die Realisierung der Unabhängigkeit des Systems von der Art des Datenspeichers. In diesem Sinne ist der für den Datenspeicher spezifische Quellcode in einer einzigen konzeptionell abgegrenzten Schicht konzentriert: dem 'Repository Abstraction Layer'(RAL). Das RAL stellt den drei Modulen des Systems verschiedene RDF - spezifische Methoden zur Verfügung und übersetzt diese in für den Datenspeicher spezifische Aufrufe. Diese separate Schnittstelle ermöglicht die Verwendung von Sesame auf verschiedenen Datenspeichern ohne Veränderungen am Rest des Systems.

Das Anfragemodul(siehe Abschnitt 4.2.3), das Administrationsmodul zum Hinzufügen und Löschen von RDF/S Daten und das Exportmodul für den Export in ein XML-Format verwenden das RAL.

Die Kommunikation mit diesen Modulen ist von außen auf unterschiedliche Arten möglich und von der Umgebung abhängig. Die Benutzung verschiedener Protokolle zwischen Modulen und Systembenutzern wird von Protokollprogrammen('Protocol Handler') gestaltet.

Die Möglichkeit verschiedene Protokollprogramme für die Anbindung verschiedener Betriebssysteme dem System hinzuzufügen und die Anpassung an verschiedene Datenspeicher durch Reimplementierung des RALs, zeigen die Realisierung einer generischen Architektur zur Speicherung und Anfrage von RDF/S durch Sesame.

Im folgenden werden die im Rahmen meiner Implementierung veränderten und wichtigen Komponenten von Sesame näher erläutert.

### 4.2.1 Der Datenspeicher

Das Konzept des RALs erlaubt es Sesame auf jeden beliebigen Datenspeicher, der RDF Daten aufnehmen kann, aufzusetzen. Eine konkrete Implementierung für folgende Arten von Datenspeichern ist vorstellbar:

- Datenbanksysteme(relational, objektrelational etc.)
- bestehende RDF Speicher(siehe Guha)
- Dateien(für kleine Datenmengen gut geeignet)
- Netzwerke

Zur Zeit sind die zwei relationalen Datenbanksysteme MySQL und Postgres für Sesame als Datenspeicher implementiert.

### 4.2.2 Das Repository Abstraction Layer

Wie beschrieben bietet das Repository Abstraction Layer eine stabile Schnittstelle für die Kommunikation des Systems mit der Datenbank. Das RAL wird definiert durch eine API, die Funktionalitäten zum Hinzufügen, Erhalten und Löschen von Daten der Datenbank zur Verfügung stellt. Die RAL-Implementierung übersetzt diese Aufrufe der API Methoden in Operationen auf der unterliegenden Datenbank.

Zusätzlich speichert das RAL alle das Schema betreffenden Informationen in einer festgelegten Struktur im Hauptspeicher ab. Da diese Schema Informationen meist von begrenzter Größe sind und häufig benutzt werden, ist eine Zwischenspeicherung sinnvoll. Außerdem sind die Schema Informationen die, die am schwierigsten von dem Datenbanksystem aufgrund der Transitivität von den subClassOf und subPropertyOf Eigenschaften anfragbar sind.

Das RAL erfüllt noch die weitere wichtige Aufgabe der Behandlung von Nebenläufigkeitsaspekten. Die Implementierung des RALs sieht das ausgewählte Blocken und Freigeben für Lese- und Schreibzugriffe nach dem 'first-come-first-serve' Prinzip vor.

### 4.2.3 Anfragemodul

RQL ist der Vorschlag einer deklarativen Anfragesprache für RDF und RDFSchema von Karvounarkis. Die in Sesame verwirklichte Version von RQL enthält jedoch einige Abweichungen zur Vorgeschlagenen. Der Abschnitt 3.2 erläutert diese und zeigt die Nähe der Version von Sesame zum formalen Graphenmodell der W3C Spezifikation. Eine Einführung in RQL von Sesame ist im Abschnitt 5.2 zu finden.

Die Anfrageverarbeitung erfolgt in einer festgelegten Reihe von Schritten. Nach Analyse der Anfrage wird das erstellte Baummodell dem Anfrageoptimierer übergeben, der die Anfrage in eine inhaltlich äquivalente aber effizientere umformt. Die optimierte Anfrage wird in mehrere Teilanfragen, die sich aus der Baumstruktur ergeben, zerlegt und Schritt für Schritt evaluiert. Die Teilanfragen werden wieder auf diese Weise behandelt, bis sie nicht weiter zerlegbar sind. Diese

einfachen Anfragen werden nun in Methodenaufrufe an das RAL übersetzt, welches die Ergebnisse liefert. Um den Speicher nicht zu belegen werden die Ergebnisse in Datenströmen und nicht als Ergebnismengen zurückgegeben. Sesame übersetzt also eine RQL Anfrage in eine Menge von Aufrufen an das RAL. Das bedeutet, dass der größte Teil der Anfrageevaluierung innerhalb des Anfragemoduls geschieht. Enthält eine Anfrage z.B eine Join-Operation über zwei Unterabfragen, dann wird jede Unterabfrage einzeln evaluiert und der Join wird innerhalb des Anfragemoduls auf den Ergebnissen ausgeführt.

Diese Entwurfsentscheidung hat den großen Vorteil, die Unabhängigkeit vom Datenbanktyp zu erhalten, da die Evaluation des Ergebnisses einer Anfrage durch RAL-Methoden realisiert wird, die sich beliebig implementieren lassen. Es werden jedoch Strategien zur Optimierung der Evaluation innerhalb des Anfragemoduls gebraucht, da man hier die Optimierung komplexer Abfragen des jeweils benutzten Datenbanksystem nicht nutzten kann.

## 4.3 Implementierungen von Sesame

### 4.3.1 Das ursprüngliche Sesame

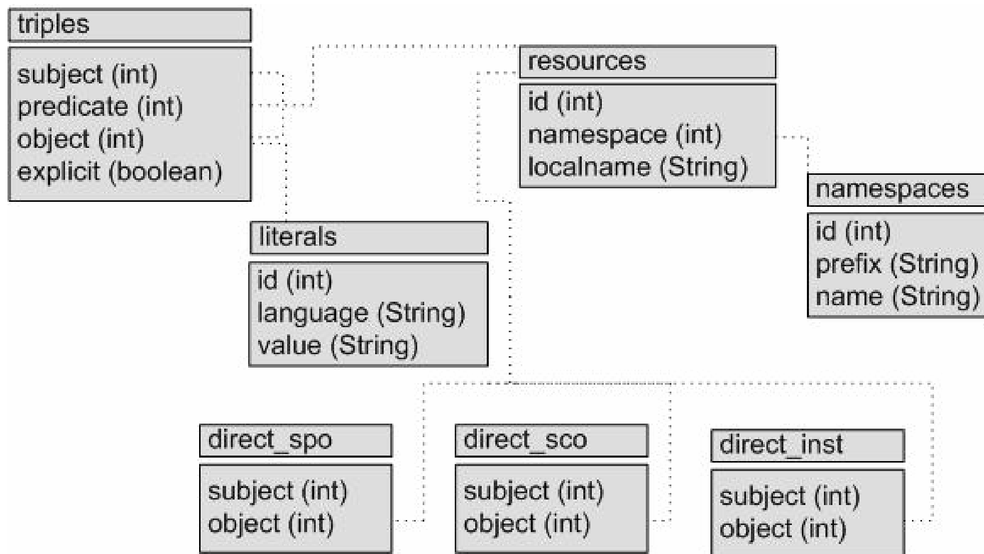
Im vorhergehenden Abschnitt wurde Sesame als Projekt im Ganzen vorgestellt. Die Vorteile und Nachteile der Implementierung eines objektorientierten Speichers für Sesame sind jedoch erst bei genauerer Betrachtung erkennbar. So werden im Folgenden die Implementierungen von Sesame für MySQL und meine Entwicklung für FastObjects näher erläutert und verglichen. Obwohl Sesame für MySQL und Postgres implementiert wurde, werde ich nur MySQL vorstellen, da von diesem relationalen Entwurf aus der objektorientierte entwickelt wurde.

#### 4.3.1.1 RelationaleDatenbank MySql

Das von Sesame benutzte Datenbankschema für MySQL sieht sieben Tabellen vor.

Alle RDF/S Daten werden im Datenbanksystem als Tripel in der Tabelle *triples* abgelegt. Die Einträge der Tripel bestehen aus jeweils drei Schlüsseln für Subjekt, Prädikat und Objekt. Die Schlüssel verweisen auf die Daten der Quellen abgelegt in der Tabelle *resources* oder der Literale abgelegt in der Tabelle *literals*. Die Quellen benutzten in der Tabelle *namespaces* separat verwaltete Namensräume. Die Tabellen *directspo*, *directsco* und *directinst* existieren nur aus Performancegründen für RQL Anfragen. Die dort enthaltene Information ist eigentlich schon in der Menge der gespeicherten Tripel enthalten, benötigt jedoch eine Menge Zeit um jedesmal gefiltert zu werden.

Die Figur??? zeigt die einzelnen Tabellen derMySQL-Datenbank mit ihren Feldern. Welches Feld semantisch auf welche Art von Information verweisen darf, ist durch die gestrichelten Pfeile gekennzeichnet. Das *object* - Feld in der Tabelle *triples* kann zum Beispiel den Schlüssel von einem Datensatz aus der Tabelle *resources* oder von der Tabelle *literals* enthalten, da Objekte von RDF Ausdrücken sowohl Quellen als auch Literale sein können.




Für das Beispiel ???? ist folgend gezeigt, in welcher Weise die Tabellen gefüllt werden würden. Die *namespace* Tabelle und alle *direct**-Tabellen( directinst, directspo und directsco) wurden hier vernachlässigt.

**Tripel :** *type( Resource Class)*
*type( Creator Class)*
*type( FamousCreator Class)*
*subClassOf( FamousCreator Writer)*
*type( hasCreator Property)*
*domain( hasCreator Creator)*
*range( hasCreator Creator)*
*type( Home/Lassila FamousCreator)*
*type( staffId/85740 Resource)*
*hasCreator( Home/Lassila staffId/85740)*

| ***triples*** | | | ***resources*** | | | ***literals*** | | |
|---|---|---|---|---|---|---|---|---|
| ***subject*** | ***predicate*** | ***object*** | ***id*** | ***namespace*** | ***localname*** | ***id*** | ***language*** | ***value*** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *1* | *2* | *3* | *1* | | *Resource* | *11* | *en* | *Home/Lassila* |
| *10* | *2* | *3* | *2* | | *Type* | *12* | *en* | *staffId/85740* |
| *9* | *2* | *3* | *3* | | *Class* | | | |
| *9* | *8* | *10* | *4* | | *Literal* | | | |
| *13* | *2* | *5* | *5* | | *Property* | | | |
| *13* | *6* | *10* | *6* | | *Domain* | | | |
| *13* | *7* | *10* | *7* | | *Range* | | | |
| *11* | *2* | *9* | *8* | | *SubClassOf* | | | |
| *12* | *2* | *1* | *9* | | *FamousCreator* | | | |
| *11* | *13* | *12* | *10* | | *Creator* | | | |
| | | | *13* | | *HasCreator* | | | |

Das vorgestellte Datenbankschema realisiert die Abbildung von RDF/S Daten auf eine relationale Speicherstruktur. Zu erkennen ist jedoch, dass das in Abschnitt 3.2 beschriebene formale Graphenmodell in diesem Entwurf nur als semantische Interpretation der gespeicherten Menge von RDF Tripeln existiert. Die Struktur der Tabellen selbst und damit das Datenbankschema bilden diesen in keiner Weise ab. Da jedoch RQL den Gebrauch der Semantik des formalen Graphenmodells voraussetzt, muß dieser Graph innerhalb des RAL's existieren um benutzt zu werden.

Wie der Graph entsteht und benutzt wird ist im folgenden Abschnitt erläutert.

#### 4.3.1.2 RAL

Das RAL als Schnittstelle zwischen den funktionalen Modulen und dem Datenspeicher stellt verschiedene Arten von Methoden zur Verfügung.

- Initialisierung

  Die Initialisierung bewirkt den Aufruf verschiedener anderer Methoden wie z.B. das Anlegen von Tabellen, Ableiten von RDF Semantik und die Pflege der *direct**-Tabellen.

- Pflege der *direct**-Tabellen

  Die *directinst-* (enthält alle Beziehungen von direkten Instanzen), *directsco-* (enthält alle *subClassOf-* Beziehungen) und *directspo-* Tabelle (enthält alle *subPropertyOf-* Beziehungen) werden benutzt, wenn die genannten Eigenschaftstypen in RQL angefragt werden. Die Anfrage erfordert die Interpretation der RDF Daten als Graph, da Informationen über Beziehungen erfragt werden, die sich aus der Verfolgung der Pfade im Graph ergeben und nicht aus den Tripeln alleine. Das bedeutet, dass eine bestimmte begrenzte Menge von semantischer Information des formalen Graphenmodells innerhalb dieser Tabellen vorrätig gehalten wird, um nicht bei jeder Anfrage aus den Tripeln den Graph aufbauen zu müssen. Die Initialisierung und Pflege setzt jedoch die Existenz des Graphen zu diesem Zeitpunkt voraus. Um den Graphen zu erstellen, müssen die einzelnen Tripel zusammengesetzt werden. Die Verbindung der Tripel über gleiche Subjekte und Objekte bildet den Graph und wird noch innerhalb des Datenbanksystems durch SQL-Anfragen mit Hilfe von Joins realisiert. Um Eigenschaften wie *subClassOf-*

Beziehungen etc. nachzuvollziehen können nun die Pfade des durch die Verkettung der Datensätze der *triples*-Tabelle entstandenen Graphen verfolgt werden. Auch in allen anderen Methoden des RAL's wird mit dieser Methodik der Graph gebildet und benutzt.

Die *direct**-Tabellen werden bei jeder Veränderung der Datenbasis(Hinzufügen oder Löschen von RDF/S Daten) aktualisiert.

- Inferencer

  Nach RDFModelTheory!!!!!! lassen sich aus einer Menge von RDF/S Ausdrücken aufgrund der Semantik des Graphen weitere Ausdrücke ableiten. Die verschiedenen Regeln sind in dieser Arbeit spezifiziert. Beispielsweise gilt, dass wenn zwei Tripel der folgenden Art *subPropertyOf(aaa bbb)* und *subPropertyOf(bbb ccc)* existieren, dann gilt auch *subPropertyOf(aaa ccc)*. Die abgeleiteten Tripel werden der Datenbasis genauso wie die expliziten Ausdrücke hinzugefügt, jedoch gekennzeichnet. Die Anwendung der Regeln erfordert die Interpretation des Graphen und wird wie im vorhergehenden Punkt realisiert.

  Die Aktualisierung der Menge der abgeleiteten Tripel wird bei jeder Veränderung der Datenbasis(Hinzufügen oder Löschen von RDF/S Daten) angestoßen.

- Hinzufügen von ganzen RDF Ausdrücken, Quellen, Literalen und Namensräumen:

  Neue RDF Ausdrücke werden in die Datenbasis aufgenommen indem die Werte von Subjekt, Prädikat und Objekt in den jeweiligen Tabellen einzeln abgespeichert werden und ihre Schlüssel zusammen ein Tripel in der *tripels*-Tabelle bilden.

- Löschen von ganzen RDF Ausdrücken,

  Ein RDF Ausdruck wird gelöscht indem sein Tripel von der Datenbasis entfernt wird. Folgt daraus, dass nun Quellen oder Literale in keinem einzigen RDF Ausdruck der Datenbasis mehr auftauchen, so werden diese auch gelöscht.

- Filterung/prüfen aus Existenz von ganzen RDFAusdrücken

  Vollständige RDF Ausdrücke können in der Datenbasis durch einfaches Suchen in der *tripels*-Tabelle gefunden werden. Der Graph muß nicht betrachtet werden, weil alle im Graph enthaltenen impliziten Ausdrücke durch die Benutzung des Inferencers explizit festgehalten wurden.

- Filterung von Schema spezifischen Eigenschaften

  Die Methoden des RAL, welche die Ergebnisse der nicht weiter zerlegbaren Anfragen einer komplexen Anfrage bereitstellen, filtern die Tripel nach bestimmten Kriterien. Enthalten diese Kriterien Schema spezifische Anforderungen, so muß der Graph benutzt werden, um die Information zu erhalten. Die Interpretation des Graphen wird wie vorhergehend beschrieben realisiert.

- Dateniteratoren

  Dateniteratoren erlauben ein Iterieren durch eine bestimmte Teilmenge der vorhandenen Information und werden benutzt um riesige Mengen von Ergebnissen im Speicher zu

vermeiden. Die Datenbasis wird von diesen Methoden gar nicht berührt, da diese Methoden von anderen des RAL aufgerufen werden, die die Daten bereitstellen.

### 4.3.2 Umsetzung und Benutzung des formalen Graphenmodell bei Sesame

Die Veranschaulichung des Datenspeichers und des RAL zeigt, dass die Abbildung des Graphen auf das Datenbankschema in der ursprünglichen Form von RDF Tripeln realisiert wird. Den Graph als solchen kennt der Datenspeicher nicht. Das Modell des Graphen entsteht erst dort, wo es benutzt wird, nämlich in der Schnittstelle zwischen Datenspeicher und System. Die Entstehung und Benutzung des Graphen wird dort durch die Verbindung der einzelnen Tripel mit sich selbst mit Hilfe von Joins realisiert. Damit ist die Suche eines Pfades im Graph keine Verfolgung von Kanten, sondern nur eine Suche nach übereinstimmenden Werten von Subjekten und Objekten in einer Tripelmenge.

### 4.3.3 Ich

In den vorhergehenden Abschnitten wurde Sesame mit seinem Datenspeicher und der ursprünglichen RAL-Implementierung vorgestellt, um die Umsetzung des formalen Graphenmodells kennen zu lernen. Die Benutzung eines objektorientierten Ansatzes für die Datenspeicherung hat ganz andere Möglichkeiten, die im folgenden dargestellt werden. Zu Beachten ist, dass sich in der Struktur von Sesame nichts ändert. Die Veränderungen am System zum Gebrauch einer OODB betreffen lediglich den Entwurf des Datenspeichers selbst und die RAL-Implementierung.

#### 4.3.3.1 Objektorientierte Datenbank Poet

Meine Version von Sesame benutzt die objektorientierte Datenbank FastObject t7. Wie in Abschnitt 3.3 vorgestellt beeinhaltet der Entwurf einer objektorientierten Datenbank vor allem die Definition der Datenobjekte, welche persistent gespeichert werden sollen.

Mein Datenbankentwurf für diese OODB sieht sieben Objekttypen vor.

Die als Tripel vorliegenden RDF/S Daten werden nicht als Objekte betrachtet, aber die einzelnen Bestandteile Subjekt, Prädikat und Objekt sind Instanzen von zwei möglichen Objekttypen. Subjekte, Prädikate und Objekte können entweder von Literalen oder Quellen repräsentiert werden. Somit werden alle Bestandteile eines Tripels entweder als *Literals*-Objekt oder als *Resources*-Objekt abgespeichert. Um die Tripel selbst speichern zu können benötigen beide Objekttypen noch die Möglichkeit Bestandteil(Subjekt) eines Tripels sein zu können. Um dies zu realisieren, erben beide von dem *Values*-Objekttyp(Superklasse von *Literals* und *Resources*) das Attribut *out_edges*. Die Eigenschaft *out_edges* ist eine eindeutige Abbildung von einer Menge von Schlüssel auf eine Menge von Listen von Objekten. Die jeweilige *Values*-Instanz stellt das Subjekt des Tripels dar und die Schlüssel des enthaltenen Attributs sind referenzierte *Values*-

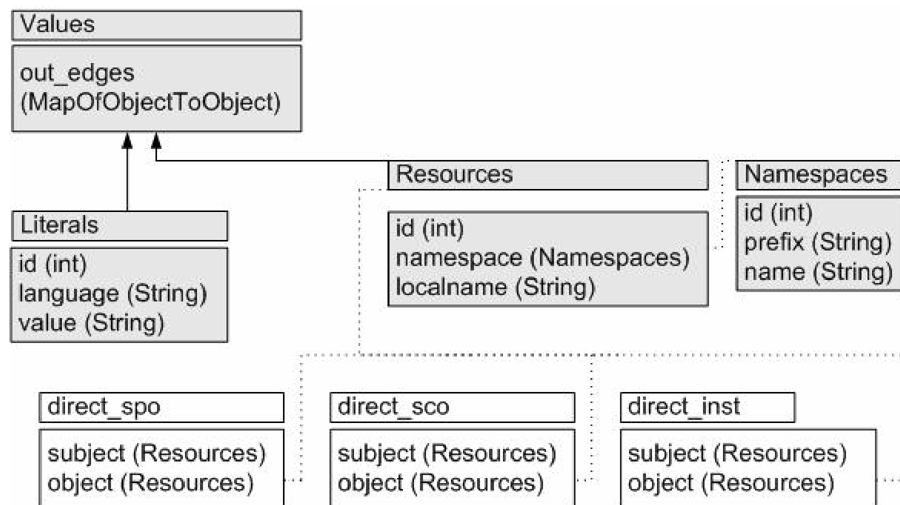
Objekte, welche die Prädikate repräsentieren. Die zu jedem Schlüssel gehörende Liste von Objekten referenziert die *Values*-Objekte, die für das gegebene Subjekt und Prädikat das Objekt bilden. Somit kann jedes beliebige *Values*-Objekt Subjekt eines Tripels sein, indem durch das *out_edges* Attribut Prädikate und Objekte definiert werden. Da jedes *Values*-Objekt immer referenziert werden kann, sind auch die Rollen als Prädikat und Objekt innerhalb eines Tripels zu jeder Zeit definierbar.

Die Tripel werden also nur durch die Verweis- und Objektstruktur der *Values*-Objekte abgespeichert jedoch nicht explizit.

Die Namensräume werden als eigener Objekttyp gespeichert auf den jede Quellinstanz verweist.

Die Objekttypen *directspo*, *directsco* und *directinst* existieren nur aus Performancegründen für RQL Anfragen. Sie enthalten zwei Verweise auf *Values*-Objekte, welche die Subjekte und Objekte der Tripel mit dem Prädikat definiert durch den jeweiligen Objekttyp darstellen. Die dort enthaltene Information ist eigentlich schon in der Verweisstruktur der *Values*-Objekte enthalten, ist in dieser Form jedoch schneller verwendbar.

Die Figur??? zeigt die einzelnen Objekttypen der FastObjects-Datenbank mit ihren Attributen. Welches Attribut auf welchen Objekttyp verweisen darf, ist durch die gestrichelten Pfeile angedeutet. Der *Literals*- und *Resources*-Objekttyp erben von *Values*.




Für das Beispiel ???? ist folgend gezeigt, welche Objektinstanzen angelegt werden würden. Das *outedges*-Attribut ist eine MapOfObjectToObject mit Schlüsseln(key1...keyn) und Listen von Objekten(value1...valuen) .

Beispiel:

*type( Resource Class)*

*type( Creator Class)*
*type( FamousCreator Class)*
*subClassOf( FamousCreator Creator)*
*type( hasCreator Property)*
*domain( hasCreator Creator)*
*range( hasCreator Creator)*
*type( Home/Lassila FamousCreator)*
*type( staffId/85740 Resource)*
*hasCreator( Home/Lassila staffId/85740)*

| ***resources*** | | | | ***literals*** | | | |
|---|---|---|---|---|---|---|---|
| ***id*** | ***namespac e*** | ***localname*** | ***outedges {key1:{value1,value2,..}, key2: {value1,value2,..}, ….}*** | ***id*** | ***language*** | ***value*** | ***outedges {key1:{value1,value2,..}, key2: {value1,value2,..}, ….}*** |
| ***1*** | | ***Resource*** | ***{Type:{Class}}*** | ***1*** | ***en*** | ***Home/Lassila*** | ***{Type:{FamousCreator},*** |
| ***2*** | | ***Type*** | | ***1*** | | | ***HasCreator:{staffId/*** |
| ***3*** | | ***Class*** | | | | | ***85740}}*** |
| ***4*** | | ***Literal*** | | | ***en*** | ***staffId/85740*** | ***{Type:{Resource}}*** |
| ***5*** | | ***Property*** | | ***1*** | | | |
| ***6*** | | ***Domain*** | | ***2*** | | | |
| ***7*** | | ***Range*** | | | | | |
| ***8*** | | ***SubClassOf*** | | | | | |
| ***9*** | | ***FamousCreato*** | ***{SubClassOf:{Creator},*** | | | | |
| | | ***r*** | ***Type:{Class}}*** | | | | |
| ***1*** | | | ***{Type:{Class}}*** | | | | |
| ***0*** | | ***Creator*** | ***{Type:{Property},*** | | | | |
| ***1*** | | ***HasCreator*** | ***Domain:{Creator},*** | | | | |
| ***3*** | | | ***Range: {Creator}}*** | | | | |

Der Datenbankentwurf zeigt eine Möglichkeit der Abbildung von RDF Daten auf eine objektorientierte Speicherstruktur. Verschiedene andere sind denkbar und während der Phase der Implementierung von mir getestet worden. Der große Vorteil dieser objektorientierten Speicherstruktur ist, dass das in Abschnitt 3.2 beschriebene formale Graphenmodell direkt abgebildet wurde. Die Objekte stellen mit ihren Verweisen nicht nur semantisch sondern auch physisch im Speicher einen Graph dar. Für die Verwendung des Graphen bei RQL-Anfragen muss kein Graph innerhalb des RALs erzeugt werden, da er bereits im Speicher besteht. Er kann direkt vom RAL benutzt werden. Wie der Graph benutzt wird ist im folgenden Abschnitt erläutert.

#### 4.3.3.2 RAL

Die verschiedenen Arten von Methoden des RALs wurden schon im Abschnitt 4.3.1.2 eingeführt. Obwohl die Implementierung für eine objektorientierte Datenbank völlig unterschiedlich zur objektrelationalen ist, werden im Folgenden nur konzeptionelle Unterschiede in der Verwendung des Graphen durch die Methoden beschrieben.

- Initialisierung
- Pflege der *direct**-Tabellen:

  Bei der Verwendung dieser Eigenschaftstypen in RQL werden Informationen benutzt, die sich durch die Interpretation der RDF Daten als Graph ergeben. Zur Identifikation dieser Beziehungen müssen die Pfade im Graph verfolgt werden . Diese Informationen sind bei der Initialisierung abgespeichert worden. Bei einem objektorientierten Ansatz muss der Graph aber nicht einmal bei der Initialisierung erzeugt werden, weil er im Speicher bereits besteht. Die Pfade des Graphen lassen sich von jeder Quelle oder Literal über die Verweise direkt verfolgen, um Eigenschaften wie *subClassOf*-Beziehungen etc. nachzuvollziehen.

  Auch in allen anderen Methoden des RAL's kann der Graph auf diese Weise benutzt werden.

- Inferencer:

  Die beschriebene Anwendung der Regeln erfordert die Interpretation des Graphen und wird wie im vorhergehenden Punkt realisiert. Die abgeleiteten Tripel werden dem Graphen im Speicher genauso wie explizite Ausdrücke hinzugefügt, jedoch werden sie nicht wie beim objektrelationalen Speicher gekennzeichnet. Diese Kennzeichnung ist durch die Verwendung von Verweisen unmöglich, da nirgendwo ein Tripel als Ganzes existiert,was gekennzeichnet werden könnte.

- Hinzufügen von ganzen RDF Ausdrücken, Quellen, Literalen und Namensräumen:

  Neue RDF Ausdrücke werden in die Datenbasis aufgenommen, indem die Instanzen der Subjekte entweder um Prädikat und Objekt erweitert werden, oder gegebenenfalls neu angelegt werden.

- Löschen von ganzen RDF Ausdrücken

  Ein RDF Ausdruck wird gelöscht, indem die Verweise der Instanz des Subjekts gelöscht werden. Ist diese Instanz nun von keinem einzigen RDF-Ausdruck der Datenbasis mehr Subjekt, so wird sie selbst auch gelöscht.

- Filterung/Prüfen auf Existenz von ganzen RDF-Ausdrücken

  Da keine Menge von Trieln mehr existiert können vollständige RDF Ausdrücke in der Datenbasis nicht mehr durch einfaches Suchen in dieser Menge gefunden werden. Der Graph im Speicher wird mit einer Tiefensuche von einem beliebigen Knoten( ein Subjekt) aus durchlaufen, während alle Kanten mit den zwei zugehörigen Knoten ein Tripel darstellen. Diese Tripel werden während der Suche auf die spezifizierten Kriterien hin untersucht.

- Filterung von Schema spezifischen Eigenschaften

  Die nicht mehr weiter zerlegbaren RQL-Anfragen verwenden den Graphen wie vorhergehend beschrieben.

- Dateniteratoren

### 4.3.4 Umsetzung und Benutzung des formalen Graphenmodell bei mir

Die in Abschnitt 3.3 beschriebenen Eigenschaften der komplexen Objekte, Objektidentität, Typen und Klassen, Hierarchien und Persistenz einer objektorientierten Datenbank erlauben also schon auf Speicherebene die Repräsentation der RDF Tripel als Graph. Es müssen nur zwei Objekttypen persistent gespeichert werden, da die n-ären Beziehungen durch Verweise von den Objekten modelliert werden können. Da der Graph schon im Speicher bekannt ist, kann dieser von der Schnittstelle zwischen Datenspeicher und System benutzt werden ohne bei jeder Verwendung erzeugt werden zu müssen. Die Suche im Graph ist somit eine Tiefensuche bei der alle möglichen Pfade über die Kanten betrachtet werden.

Dieser Entwurf ist speziell für eine FastObjects t7 Datenbank konzeptioniert worden und wegen mangelnder Standardisierung von objektorientierten Datenbanksystemen nicht unbedingt auf andere übertragbar ist.

## 4.4 Vergleich und Beurteilung

Schon die Darstellung der konzeptionell völlig unterschiedlichen Ansätze zeigt die Unterschiede deutlich. Nun ist ein Vergleich möglich, der besonders hinsichtlich der im Abschnitt 3.4 angesprochenen Vorteile der Abbildung des RDF- Graphen auf ein objektorientiertes Datenbankschema interessant ist.

Wie erwartet wurde beim objektorientierten Ansatz der Speicherentwurf stark vereinfacht und stellt eine direkte Abbildung dar. Anstatt von Tabellen, deren RDF Daten über die Werte ihrer Einträge als Graph interpretiert werden müssen, können wenige Objekttypen verwendet werden, die durch Verweise zwischen ihren Instanzen gleich den beabsichtigten Graph darstellen. Der Graph besteht persistent im Speicher und in der Schnittstelle muß keine Übersetzung einer Speicherstruktur wie z.B Tabellen erfolgen. Es kann unmittelbar, direkt und zu jeder Zeit auf dem Graph gearbeitet werden. Der wohl entscheidende Vorteil konnte jedoch auf Grund der Verwendung von Sesame bisher nicht ausgenutzt werden. Die Nähe von OQL und RQL kann aufgrund des Ansatzes das System unabhängig vom Datenbanksystem zu halten nicht genutzt werden. Alle Anfragen in RQL werden in eine Menge von Teilanfragen zerlegt und ihre Ergebnisse von Methoden des RAL bereitgestellt. Eine Übersetzung von den komplexen Anfragen in die jeweilige Datenanfragesprache für das Speichersystem erfolgt nicht. Dies bedeutet, dass die Benutzung von der Datenanfragesprache nur innerhalb der RAL Methoden möglich ist und zu diesem Zeitpunkt der Vorteil von ähnlichen Sprachkonstrukten schon keine Rolle mehr spielt. Zu diesem Zeitpunkt sind die Anfragen so einfach, dass die Vorteile von Formulierungen zur Pfadsuchen etc. nicht mehr gebraucht werden.

Somit sind die Vorteile des objektorientierten Ansatzes in meiner Implementierung nur teilweise verwertbar gewesen. Die Verwendung von Sesame hat jedoch den großen Vorteil der Vergleichbarkeit. Zur Beurteilung der Effizienz der verschiedenen RDF Speichersysteme können alle denkbaren Tests auf beiden Ansätzen aber innerhalb des gleichen Systems durchgeführt werden.

# 5 Performance

Erwartungen aufgraund der Probleme???????????????

Nur reine Vergleichstests -----keine wirklichen Performancemessungenauf getricksten Datenbasisen!

## 5.1 Hochladen

Korrektheit: anzahlderTripelin Objekte/Tabellenist gleich?

## 5.2 RQL

Der folgende Abschnitt gibt eine kurze Einführung in RQL. Der Überblick zeigt die verschiedenen Arten von Anfragen in RQL. Jedoch werden nur diese vorgestellt die in SESAME auch implementiert wurden. Dies sind nicht alle siehe RQLUser Manual(ICS FOrth).

Wie in der Einführung beschrieben ist RQL eine getypte Sprache die OQL sehr ähnlich ist. Es gibt eine Menge von grundlegenden Anfragen, welche die zusammensetzbaren Blöcke der Sprache sind. Diese Anfragen werden benutzt um mit Hilfe funktionaler Komposition komplexere Ausdrücke zu bilden, welche die für jede Operation spezifischen Typbedingungen respektieren.

RQL Anfragen können Informationen auf verschiedenen Ebenen von RDF anfragen. Die folgenden Unterscheidungen in Schema-Anfragen, Anfragen bezüglich der Quellenbeschreibungen, und ??????? sind strukturelle bezüglich der RDF Graph Semantik. Alle Arten jedoch fragen das aufgrund der RDFDaten entstandene formale Graphenmodell an.

### 5.2.1 Anfragen an das Schema

Grundlegende Anfragen an das Schema traversieren die im Schema definierten Hierarchien von Klassen und Eigenschaften. Der '^'-Operator vor einer Operation gibt nur die vollständigen Instanzen einer Klasse oder Eigenschaft zurück. Eine "Vollständige" Instanz einer Klasse bedeutet, dass diese Instanz nicht noch Instanz einer Unterklasse oder Untereigenschaft der Klasse ist.
Der '^'-Operator hinter einer Operation gibt nur die direkten Instanzen einer Klasse oder Eigenschaft zurück.

| | |
|---|---|
| **subClassOf**( #a_resource ) | gibt alle Unterklassen der Klasse #a_resource zurück |
| **subPropertyOf**( #a_property ) | gibt alle Untereigenschaften der Eigenschaft #a_property zurück |
| **subPropertyOf**^( #a_property ) | gibt alle direkten Untereigenschaften der |

| | Eigenschaft #a_property zurück |
|---|---|
| **subClassOf^**( #a_resource ) | gibt alle direkten Unterklassen der Klasse #a_resource zurück |
| **domain**( #a_property ) | gibt alle in den RDF Daten spezifizierten *domains* der Eigenschaft #a_property zurück |
| **range**( #a_property ) | gibt alle in den RDF Daten spezifizierten *ranges* der Eigenschaft #a_property zurück |
| **^subClassOf**( #a_resource ) | gibt alle vollständigen Unterklassen der Klasse #a_resource zurück |

Anfragen an das Meta-Schema fragen das gesamte Schema an wie normale Information durch vorher definierte Metaklassen wie z.B. Class und Property in RDFSchema.

| **Class** | gibt alle im Repository existenten Klassen zurück |
|---|---|
| **property** | gibt alle im Repository existenten Eigenschaften zurück |

Anfragen zu Klassen und Eigenschaften benutzten einen select-from-where Filter um über Mengen mit Hilfe eingeführter Variablen zu iterieren. Schema Variablen werden mit einem vorhergehenden `$ im Falle einer Klasse und einem` @ im Falle einer Eigenschaft gekennzeichnet. Die Menge { **:** `#a_resource` } `beinhaltet diese.` Die Menge { X **:** `#a_resource` } `beeinhaltet zusätzlich zu der Klasse und allen Unterklassen noch die Instanzen dieser Klassen.`

| **select @P**<br>**from { :** #a_resource **} @P** | gibt alle (auch geerbte) Eigenschaften zurück, die für die Quellen der Klasse #a_resource definiert sind, wenn #a_resource Teil der *domain* von der Eigenschaft ist |
|---|---|
| **select @P from** #a_resource **. @P** | gibt alle (auch geerbte) Eigenschaften zurück, die für die Quellen der Klasse #a_resource definiert sind, aber #a_resource muß nicht Teil der *domain* von der Eigenschaft sein |
| **select @P, $Y**<br>**from { :** #a_resource **} @P {:$Y}** | gibt alle Eigenschaften und alle *range* - Klassen zurück, die für die Quellen der Klasse #a_resource definiert sind |
| **select domain(@P), @P, range(@P)**<br>**from @P where @P <** #a_property | gibt die *domain* - und *range* - Klassen selbst zurück, wie sie in den RDF Daten spezifiziert sind, jedoch nur von den Eigenschaften die Unterklassen von #a_property sind |

Um Anfragen zu formulieren die die Navigation innerhalb eines Schemas erfordern werden Pfadausdrücke benutzt. Benutzt man den '.'- Operator so kann man explizit entlang der Pfade im Schemagraph traversieren

| **select $X, $Y**<br>**from** #a_property1 **{:$X}.**<br>#a_property2 **{:$Y}** | gibt alle *range* - Klassen von der Eigenschaft #a_property2 zurück, die erreichbar sind von einer der *range* - Klassen von #a_property1 |
|---|---|
| **select $X, $Y**<br>**from** #a_property1 **{:$X},** | nur andere RQL Formulierung |

| | |
|---|---|
| #a_property2 **{:$Y} where $X=$Y** | |
| **select $X, @P, range(@P) from** #a_property **{:$X}. @P** | gibt alle Eigenschaften und deren *ranges* die auf *range* - Klassen von Eigenschaft #a_property anwendbar sind zurück |

### 5.2.2 Anfragen an die Quellbeschreibungen

Die Kernanfragen in RQL greifen auf die RDF Daten zu ohne das benutzte Schema genau zu kennen.

Grundlegende Anfragen an die RDF Daten traversieren die RDFDaten im Graph.

| | |
|---|---|
| #a_resource | gibt alle Instanzen der Klasse #a_resource zurück |
| ^#a_resource | gibt alle vollständigen Instanzen der Klasse #a_resource zurück |
| #a_property | gibt alle Quellen zurück, die die Eigenschaft #a_property oder eine Untereigenschaft von dieser haben und die Werte, welche diese Eigenschaft dann hat |
| ^#a_property | gibt alle Quellen zurück, die die Eigenschaft #a_property haben und die Werte, welche diese Eigenschaft dann hat |

Zur Filterung von Information werden die Standardvergleichsoperatoren benutzt. Vergleiche mit Stringwerten oder Namen von Quellen oder Eigenschaften sind möglich.

| | |
|---|---|
| **select Y from {X}** #a_property **{Y} where X =** #a_resource | gibt für Quelle #a_resource den Wert zurück, den diese für die Eigenschaft #a_property hat |

Um innerhalb des Graphs der Quellbeschreibungen zu navigieren kann man Datenpfadausdrücke formulieren.

| | |
|---|---|
| **select X, Y from** #a_resource **{X}.** #a_property **{Y}** | gibt für alle Quellen von Klasse #a_resource den Wert der Eigenschaft #a_property zurück<br>(gibt nicht das gleiche Ergebnis wie vorhergehende Anfrage, denn keine Beschränkung durch *domain/range* ) |
| **select X, Y from** #a_resource **{X}, {Z}** #a_property **{Y} where X = Z** | nur andere RQL Formulierung |
| **select Y from** #a_resource **.** #a_property1 **.** #a_property2 **{Y}** | gibt vom Wert der Eigenschaft #a_property1 der Quelle #a_resource wiederum den Wert der Eigenschaften für #a_property2 |

### 5.2.3 Nutzung des Schemas zur Filterung von Quellbeschreibungen

In den vorhergehenden Abschnitten wurde die Navigation des Graphen bezüglich der Quellbeschreibung ohne Beachtung des Schemas und die Navigation bezüglich des Schematas beschrieben. RQL erlaubt die Verbindung von beidem mit Hilfe von gemischten Pfadausdrücken.

| | |
|---|---|
| **select X, Y**<br>**from {** X **:** #a_resource1 **}**<br>#a_property **{** Y **:** #a_resource2 **}** | gibt die Quellen von Klasse #a_resource1 und die Quellen von Klasse #a_resource2, die Werte der Eigenschaft #a_property sind, zurück |
| **select X, $X, Y, $Y from {X : $X}**<br>#a_property **{X : $Y}** | gibt alle Quellen der range - und *domain* - Klassen zurück, die für die Eigenschaft #a_property definiert sind |
| **select $X, $Y from { : $X}**<br>#a_property **{ : $Y}** | gibt nur die range - und *domain* - Klassen zurück, die für die Eigenschaft #a_property definiert sind |

### 5.2.4 Darstellung des Schemas von Quellbeschreibungen

RQL ermöglicht das Schema ausgehend von Quellbeschreibungen anzufragen. Möchte man wissen unter welchen Klassen eine Quelle klassifiziert wird so kann man es wie folgt tun.

| | |
|---|---|
| **typeof(** #a_resource )<br>**typeof(** #a_property ) | gibt die die Klasse/Eigenschaft klassifizierende Metaklasse zurück |

### 5.2.5 Geschachtelte Anfragen

Wie schon erwähnt lassen sich komplexe anfragen durch die Zusammensetzung einfacher formulieren. Die Schachtelung von Operationen ist dabei sehr hilfreich.

| | |
|---|---|
| **subClassOf(range(** #a_property )) | gibt die Unterklassen der *range* - Klassen von der Eigenschaft a_property zurück. |

## 5.3 Anfragen für die Tests

Reine schema anfragen auf Klassen und eigenschaftsDefinitionen

| | |
|---|---|
| **select domain(@P), @P, range(@P)**<br>**from @P** | Finde alle Eigenschaften mit im Schema spezifizierten domain und range. |
| **select $super, $sub**<br>**from Class {$super}, Class {$sub}**<br>**where $sub in subClassOf^($super)** | Finde alle Klassen mit ihren direkten Unterklassen. |
| **select $super, $sub**<br>**from Class {$super}, Class {$sub}**<br>**where $sub in subClassOf($super)** | Finde alle Klassen mit ihren Unterklassen. |

| | |
|---|---|
| `select $super, $sub`<br>`from Class {$super}, Class {$sub}`<br>`where $sub in ^subClassOf($super)` | Finde nur Klassen mit ihren vollständigen Unterklassen. |
| `select @P, @Q`<br>`from Property {@P}, Property {@Q}`<br>`where @P in subPropertyOf(@Q)` | Finde alle Eigenschaften mit ihren Untereigenschaften. |
| `select @P, @Q`<br>`from Property {@P}, Property {@Q}`<br>`where @P in subPropertyOf^(@Q)` | Finde alle Eigenschaften mit ihren direkten Untereigenschaften. |
| `select $X, $Y`<br>`from @P {:$X}. @P {:$Y}` | Fine alle Klassen und deren Unterklassen zu denen die Quellen und Werte gehören. |
| `select $X, $Y`<br>`from @P {:$X}, @P {:$Y} where`<br>`$X=$Y` | Siehe oben |

Anfragen auf Quellbeschreibungen unter Benutzung des jeweiligen Schema

- *Find the resources having a property with a speci_c (or range of) value(s)*
- *Find the instances of a class that have a given property*

Nicht nötig weil oben schon enthalten.

| | |
|---|---|
| http://www.w3.org/2000/01/rdf-schema#Resource | Finde alle Instanzen der Klasse #Resource. |
| ^http://www.w3.org/2000/01/rdf-schema#Resource | Fine alle vollständigen Instanzen der Klasse #Resource. |
| http://www.w3.org/1999/02/22-rdf-syntax-ns#type | Finde alle Instanzen der Eigenschaft #type. |
| ^http://www.w3.org/1999/02/22-rdf-syntax-ns#type | Fine alle vollständigen Instanzen der Eigenschaft #type. |

Schemaanfragen für spezifische Quellbeschreibungen

| | |
|---|---|
| `select * from {X} @P . @Q {Y}` | Finde für die Werte aller Eigenschaften von allen möglichen Instanzen wiederum alle Eigenschaften und deren Werte. |
| `select *`<br>`from {X : $X} @Q {Y : $Y}` | Finde für alle möglichen Eigenschaften alle *range* - und *domain* - Klassen mit ihren Instanzen. |
| *`select *`*<br>*`from @P { X : $X}. @Q { Y : $Y}`* | Finde für die *range* - Klassen und deren Instanzen aller möglichen Eigenschaften wiederum alle Eigenschaften und *range* - Klassen mit deren Instanzen. |

## 5.4 Zeit für RQLAnfragen

## 5.5 Platz

Vorherige Betrachtung nur auf Struktur

# 6 Diskussion und Ausblick

Vorherige Betrachtung nur auf Struktur

Verbesserungen:

RQLModul unter Benutzung von OQL!!!!!!!!

# 7 Anhang

1. Beschreibung der Implementation
2. zeitplan

   1 Monat Literatursuche

   2.Monat Schreiben der Einführung Entwicklung einer Zielsetzung

   3 Monat Einarbeitung in OODB und Sesame

   4.Monat Implementierung

   5.Monat Tests

   6.Monat Zusammenschreiben

3. Probleme mit Poet:

- Kein Casting mit Interfaces (Beispiel resource, literal, value!) nur mit Vererbung
- Probleme mit Transaktionen
  - Abbrechen bei DuplicateKey etc.
  - Nur eine aktive
- Keine Joins oder MultipleResultsets
- Dokumentation falsch siehe Vergleich mit Strings bei SetFilterMethode
- Count nicht auf Gruppierungen sondern nur auf ganze selects
- Indices vergleichen nur auf erste paar Buchstaben : sind die gleich gibt er fehler obwohl String unterschiedlich

**7.1**

# 8 Literatur und Quellen

**Referenz 1: [RDF]**
Ora Lassila und Ralph R. Swick; W3C Recommendation 22 February 1999**,**Resource Description Framework(RDF) Model and Syntax Specification

**Referenz 2: [RDFSchema]**
Dan Brickley und R.V. Guha; W3C Working Draft 30 April 2002, RDF Vocabulary Description Language 1.0: RDF Schema

**Referenz 3: [Sesame]**
Jeen Broekstra, Arjohn Kampman und Frank van Harmelen; Sesame: An Architecture for Storing and Querying RDF Data and Schema Information

**Referenz 4: [Schemata]**
Dan Brickley, R.V. Guha und Andrew Layman; W3C Working Draft 9 April 1998, Resource Description Framework(RDF) Schemas

**Referenz 5: [XMLQuery]**
Chamberlin, D., Florescu, D., Robie, J., Simeon, J. und Stefanescu, M. (2001); World Wide Web Consortium Working draft, XQuery: A Query Language for XML

**Referenz 6: [RDF-Query]**

Tuan Anh TA (2001); RDF Query: current status and future siehe unter http://perso.enst.fr/˜ta/web/rdf/rdf-query.html

**Referenz 7: [RQL]**
G. Karvounarakis, V. Christophides und D. Plexousakis; RQL : A Declarative Query Language for RDF
Greg Karvounarakis, Vassilis Christophides, Dimitris Plexousakis; Querying Semistructured (Meta)Data and Schemas on the Web: The case of RDF & RDFS

**Referenz 8: [OQL]**
Cattel, R., Barry, D., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T. und Velez, F. (2000); ODMG 3.0. Morgan Kaufmann: The Object Database Standard

**Referenz 9: [Semantic Web]**

Stefan Decker, Frank van Harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michel Klein, Sergey Melnik(2000); The Semantic Web - on the respective Roles of XML and RDF

**Referenz 10: [RDFSuite]**
S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis und K. Tolle; The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases

**Referenz 11: [RDFProjekte]**

Sergey Melnik (1999); RDF Resources siehe unter http://www-db.stanford.edu/~melnik/rdf/db.html

W3C; Survey of RDF/Triple Data Stores siehe unter http://www.w3.org/2001/05/rdf-ds/DataStore

**Referenz 12:[Squish]**

Squish; RDF Query siehe unter http://swordfish.rdfweb.org/rdfquery/

**Referenz 13:[Manifesto]**

Atkinson, DeWitt, Maier, Dittrich, Zdonik, Bancilhon; The Object-Oriented Database System Manifesto

**Referenz 14:[ Sesame: RQL]**

Jeen Broekstra, Arjohn Kampman(May 2001); Query Language Definition siehe unter www.aidministrator.nl

**Referenz 15:[ Karvounarkis: RQL]**

?????