

# Entwurf und Implementierung eines RDQL-Adapters zur Integration RDF-basierter Informationsquellen in ein schema-basiertes Peer-to-Peer-Netzwerk

## Studienarbeit

Helge Reinsch  
Helmstedter Strasse 12  
30419 Hannover, Germany

20. Juli 2003

# Inhaltsverzeichnis

<b>1</b>	<b>Suche nach Informationen</b>	<b>4</b>
<b>2</b>	<b>RDF</b>	<b>6</b>
2.1	Resource Description Framework	6
2.2	Beispiele	8
2.3	RDF-Syntax	10
2.3.1	Basic Serialization Syntax	10
2.3.2	Basic Abbreviated Syntax	12
2.3.3	Container	14
2.3.4	Aussagen über Aussagen	15
2.4	RDF-Schema	16
2.4.1	Kernklassen (Core Classes)	17
2.4.2	Kerneigenschaften (Core Properties)	18
2.4.3	Beschränkungen (Constraints)	18
<b>3</b>	<b>Edutella</b>	<b>20</b>
3.1	Allgemeines	20
3.2	JXTA-Project	20
3.3	Das JXTA Peer-to-Peer System	21
3.4	Technische Konzepte	22
3.5	Edutella Peers	25
3.6	Edutella Services	26
3.6.1	Query Service	26
3.6.2	Edutella Annotation	27
3.6.3	Edutella Replication	28
3.6.4	Edutella Mapping, Mediation, Clustering	28
<b>4</b>	<b>Edutella Query Model (EQM)</b>	<b>29</b>
4.1	Datalog	29
4.1.1	Prädikate	29
4.1.2	Fakten	29
4.1.3	Anfragen	30
4.1.4	Regeln und datenfreie Prädikate	30
4.1.5	Rekursionen	31
4.1.6	Auswertung	31
4.2	EQM	32
4.2.1	Query	32
4.3	Systematisierung verschiedener Syntax-Level	34
4.3.1	Kriterien zur Systematisierung	35
4.3.2	RDF-QEL	35
4.3.3	Regelfreie Anfrage (Rule-less Query)	36
4.3.4	Konjunktive Anfragen (Conjunctive Query)	36
4.3.5	Disjunktive Anfragen (Disjunctive Query)	36

4.3.6	Linear-Rekursive Anfragen (Linear Recursive Query) . . . . .	36
4.3.7	Generelle rekursive Anfragen (General Recursive Query) . . . . .	36
<b>5</b>	<b>RDQL</b> . . . . .	<b>37</b>
5.1	Allgemeines . . . . .	37
5.2	Sprachbeschreibung . . . . .	37
5.3	Übersetzung der Anfragen aus dem EQM in RDQL . . . . .	42
5.3.1	Rule-Less Queries . . . . .	43
5.3.2	Conjunctive Queries . . . . .	46
5.3.3	Disjunctive Queries . . . . .	49
5.4	Übersetzung der Anfrageergebnisse von RDQL in das Edutella Resultset . . . . .	57
<b>A</b>	<b>BNF</b> . . . . .	<b>58</b>
A.1	BNF-Datalog . . . . .	58
A.2	BNF-RDF . . . . .	59
A.3	BNF-RDQL . . . . .	60
A.4	BNF-SqishQL . . . . .	61

# 1 Suche nach Informationen

Das World Wide Web hat sich in den letzten Jahrzehnten von einem hierarchisch aufgebauten Dienst zum Austausch von Informationen zwischen Universitäten zu einem Informationspool für jedermann entwickelt. Behörden, Firmen, Privatpersonen, Lehrinstitute und viele weitere Institutionen und Nutzer stellen Informationen über nahezu jedes Themengebiet im Internet zur Verfügung. Die Informationen werden auf Homepages veröffentlicht, von denen sie über Clients abgerufen und angezeigt werden können.

Um es den Anwendern zu ermöglichen, die Informationen zu finden, wurden Suchmaschinen entwickelt. Diese durchsuchen mit Hilfe von Robots – kleinen Programmen zur Informationssuche – die Server und katalogisieren gefundene Stichworte in Datenbanken.

Ist ein Anwender auf der Suche nach Informationen, kann er Anfragen an Suchmaschine stellen und erhält daraufhin eine Liste vom Server zurück, die die bzgl. der Anfrage passenden Daten bereithält. Anschließend kann der Anwender die Informationen direkt von den angegebenen Servern abrufen.

Ein Problem bei dieser Art der Informationssuche ist, dass die Robots, die die Server nach Informationen durchsuchen, nur in der Lage sind, die als ASCII-Files, also z.B. als HTML- oder Textdateien, vorliegenden Daten zu untersuchen. Eine Anfrage an evtl. vorhandene Datenbanken ist i.Allg. nicht möglich, da die Anfragen oft über Formulare laufen, die sinnvoll ausgefüllt werden müssen.

Darüber hinaus können Änderungen auf den Servern nicht in Echtzeit in die Datenbanken übernommen werden. Die Änderungen werden erst "sichtbar", wenn der Robot das nächste Mal den Server durchsucht und die Datenbank aktualisiert.

Neuere Entwicklungen verfolgen einen anderen Ansatz der Informationssuche. Dabei stellt jeder Informationsanbieter eine Schnittstelle – einen Peer – auf seinem Server bereit, über den Anfragen entgegengenommen, verarbeitet und beantwortet werden können.

Die Peers melden sich bei zentralen Servern an, die diese Schnittstellen in Listen oder Datenbanken verwalten. Stellt ein Anwender eine Anfrage, so gibt es zwei Möglichkeiten, wie diese verarbeitet wird. Entweder wird sie an einen zentralen Server und von dort aus an alle bekannten Informationsanbieter weitergeleitet, oder der Peer des Anwenders tritt mit den Peers der Informationsanbieter direkt in Kontakt und stellt seine Anfrage über ein standardisiertes Protokoll.

Die Informationsanbieter beantworten dann diese Anfragen anhand der aktuellen Daten, die sich auf dem Server befinden. Dabei können sowohl die ASCII-Dateien als auch die Datenbanken in Echtzeit durchsucht werden. Die Ergebnisse werden entweder an den zentralen Server zurückgeschickt, der sie sammelt und anschließend an den Anwender sendet, oder direkt an den Anwender zurückgegeben.

Auf diese Weise sind die gesammelten Informationen auf dem aktuellen Stand, und es werden keine zentralen Server benötigt, die Stichwörter katalogisieren. Eine der bekanntesten Anwen-

dungen dieser Technologie sind die sog. Tauschbörsen – wie Gnutella oder Kaasar –, in denen über den Peer-to-Peer-Ansatz Dateien ausgetauscht werden können.

## 2 RDF

Im aktuellen Stadium des Edutella-Projektes gibt es eine Reihe von Möglichkeiten, Metadaten für den Austausch im Netzwerk bereitzustellen. So gibt es Peers, die die Daten aus einer Datenbank auslesen oder solche, die auf Basis von DBXML arbeiten. Für Anbieter von Daten, die nicht über die finanziellen oder personellen Möglichkeiten verfügen, eine Datenbank auf ihrem Server bereitzustellen, bietet es sich an, die Daten in Form einer Datei auf dem Server zur Verfügung zu stellen und diese in einer geeigneten Form abzufragen. Eine Möglichkeit, Metadaten in Form von Dateien bereitzustellen, ist, sie als RDF-Dateien zu speichern.

### 2.1 Resource Description Framework

Das Resource Description Framework – kurz: RDF – ist die Grundlage für die Beschreibung und den Austausch von Metadaten.

RDF kann in den verschiedensten Bereichen eingesetzt werden, wie z.B. in der Ressourcen-suche, um bessere Suchmaschinen zu implementieren, in der Katalogisierung, um Inhalte und Zusammenhänge zu beschreiben, die auf Webseiten oder in digitalen Büchereien zur Verfügung stehen, oder im Bereich intelligenter Softwareagenten, um Wissen gemeinsamer zu nutzen und auszutauschen. Das primäre Ziel von RDF ist, einen Mechanismus zur Beschreibung von Ressourcen zu definieren, ohne jegliche Annahmen über die späteren Anwendungen zu treffen oder die jeweilige Semantik im Vorfeld festzulegen. Die Definition dieses Mechanismus ist anwendungsneutral und dennoch passend für die Beschreibung von Informationen jeglicher Art.

Das RDF-Datenmodell ist ein syntax-neutraler Weg, um RDF-Ausdrücke darzustellen. Die Datenmodell-darstellung wird genutzt, um Übereinstimmungen im Sinne von Aussagen zu finden, wobei zwei Aussagen genau dann gleich sind, wenn ihre Darstellung im Datenmodell identisch ist. Diese Definition der Übereinstimmung erlaubt einige syntaktische Variationen in Ausdrücken, ohne dass die Aussage verändert wird.[2]

Die Grundlage des RDF-Modells sind benannte Eigenschaften (Properties) und Werte von Eigenschaften (Values). RDF-Eigenschaften sollte man sich als Attribute von Ressourcen und in diesem Zusammenhang als ein klassisches Ressource-Attribut-Wert-Tripel vorstellen. Dieses Tripel entspricht in der Logik einer Aussage und wird im Datenmodell durch drei Objekttypen realisiert:

Ressourcen (Resources):

Alles, was mit RDF-Ausdrücken beschrieben wird, nennt man Ressource. Eine Ressource kann z.B. ein einzelnes HTML-Dokument einer Web Site sein – etwa das HTML-Dokument <http://www.w3.org/Overview.html> – oder die ganze Web Site <http://www.w3.org>. Eine Ressource kann aber auch ein Objekt sein, das nicht direkt im Web verfügbar ist, etwa ein gedrucktes Buch. Ressourcen werden mittels einer URI (Universal Resource Identifier) identifiziert. Die Erweiterbarkeit der URIs erlaubt die Einführung von Erkennungsmarken (Identifiers) für jede vorstellbare Entität.

### Eigenschaften (Properties):

Eine Eigenschaft ist ein spezieller Aspekt, eine Charakteristik, ein Attribut oder eine Beziehung, mit der eine Ressource beschrieben wird. Jede Eigenschaft besitzt eine spezielle Bedeutung, definiert ihre erlaubten Werte, die Typen von Ressourcen, die durch sie beschrieben werden, und ihre Beziehung zu anderen Eigenschaften. RDF-Eigenschaften können auch Beziehungen zwischen Ressourcen darstellen, weshalb ein RDF-Modell einem Entity-Relation-Diagramm (ER-Diagramm) ähnelt. (Präziser: RDF-Schemata, die selbst Instanzen von RDF-Modellen darstellen, sind ER-Diagramme.) In der Terminologie eines objektorientierten Designs bedeutet dies: Ressourcen entsprechen Objekten und Eigenschaften entsprechen Instanzvariablen.

### Aussagen (Statements):

Eine bestimmte Ressource zusammen mit ihrer benannten Eigenschaft plus dem Wert, den die Eigenschaft für diese Ressource annimmt, ergibt eine RDF-Aussage. Diese drei individuellen Teile der Aussage nennt man das Subjekt, das Prädikat und das Objekt der Aussage. Das Objekt einer Aussage (z.B. der Wert der Eigenschaft) kann eine andere Ressource oder ein Literal, z.B. eine Ressource (spezifiziert durch eine URI), eine einfache Zeichenkette oder ein einfacher Datentyp aus XML sein. In der Sprache von RDF kann ein Literal einen XML-Inhalt haben, der aber nicht weiter vom RDF-Prozessor ausgewertet wird.

Darüber hinaus besteht RDF aus einem Klassensystem, ähnlich wie bei objektorientierter Programmierung, in dem das zugrunde liegende Vokabular festgelegt wird. Eine Menge von Klassen nennt sich ein Schema. Die Klassen sind hierarchisch organisiert und bieten eine Erweiterungsmöglichkeit durch Bildung von Unterklassen. Auf diese Weise kann leicht ein neues Schema erzeugt werden, das sich nur wenig von einem bereits existierenden unterscheidet. Es lässt sich anhand eines bekannten Schemas die Bedeutung einer Ressource erkennen.

Durch die Möglichkeit, Schemata gemeinsam zu nutzen, unterstützt RDF die Wiederverwendbarkeit von Metadaten. Die Möglichkeit der gemeinsamen Nutzung und der Erweiterbarkeit von RDF erlaubt es Metadatenautoren ebenfalls, verschiedene Vererbungen zu verwenden, um Definitionen zu vermischen, wodurch verschiedene Sichtweisen auf ihre Daten möglich werden. So kann z.B. ein Buch, das im Internet veröffentlicht wurde, auch die entsprechenden Eigenschaften einer Internetseite besitzen.

## 2.2 Beispiele

Ressourcen werden durch einen *resource identifier* identifiziert. Ein resource identifier ist eine URI zusammen mit einer *anchor id*. Zunächst sollen hier Eigenschaften jedoch nur durch einen einfachen Namen referenziert werden. Zu Beginn soll ein einfaches Beispiel betrachtet werden:

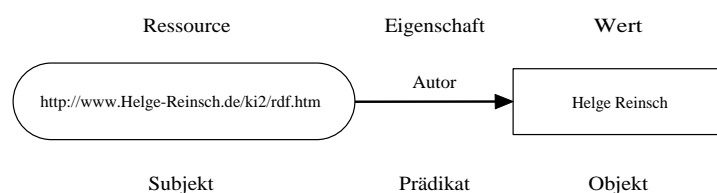
*Der Autor der Ressource <http://www.Helge-Reinsch.de/rdf.htm> ist Helge Reinsch.*

Dieser Satz hat die Bestandteile:

Subjekt	<a href="http://www.Helge-Reinsch.de/ki2/rdf.htm">http://www.Helge-Reinsch.de/ki2/rdf.htm</a>
Prädikat	Autor
Objekt	Helge Reinsch

Diese Aussage lässt sich in einem Diagramm in Form eines gerichteten benannten Graphen darstellen. In diesem Diagramm repräsentieren die Knoten – als Ovale gekennzeichnet – Ressourcen, die Pfeile repräsentieren benannte Eigenschaften, und Knoten, die Zeichenketten darstellen, werden als Rechtecke dargestellt.

Beispiel 1:

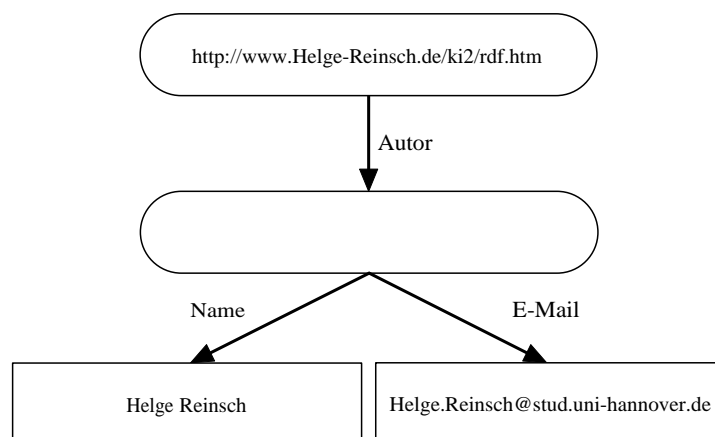


Die Richtung des Pfeiles ist wichtig. Der Pfeil beginnt immer am Subjekt der Aussage und endet am Objekt. Im Folgenden ein Beispiel, das etwas mehr über den Autor aussagt:

*Die Person mit dem Namen Helge Reinsch, E-Mail: [Helge.Reinsch@stud.uni-hannover.de](mailto:Helge.Reinsch@stud.uni-hannover.de), ist der Autor der Seite <http://www.Helge-Reinsch.de/ki2/rdf.htm>.*

Der Sinn dieses Satzes ist, für den Wert der Autor-Eigenschaft ein strukturiertes Entity einzusetzen. Dieses Entity wird in RDF durch weitere Ressourcen repräsentiert. Der Satz gibt keinen Namen für diese Ressource an, d.h. sie ist anonym, sodass sie im folgenden Diagramm durch ein leeres Oval gekennzeichnet wird.

## Beispiel 2:

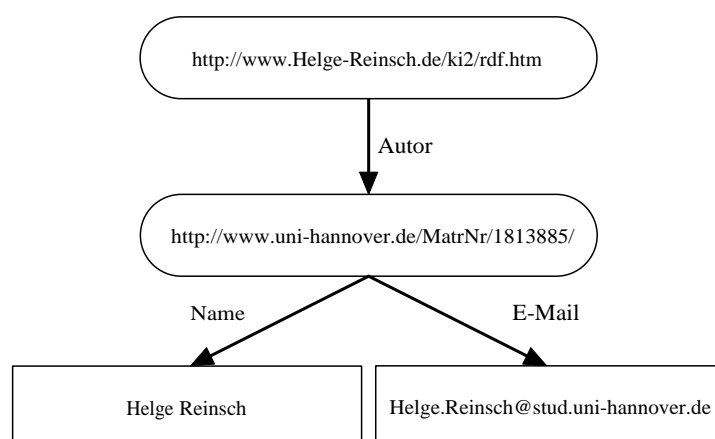


Korrespondierend zu dem zuvor Aufgeführten, ließe sich das Diagramm folgendermaßen lesen:

"Die Ressource `http://www.Helge-Reinsch.de/ki2/rdf.htm` hat den Autor IRGENDWAS, und IRGENDWAS hat den Namen Helge Reinsch und die E-Mail-Adresse `Helge.Reinsch@stud.uni-hannover.de`"

Dem strukturierten Entity des vorherigen Beispiels kann ebenso ein eindeutiger Verweis zugeordnet werden. Um das Beispiel weiter zu führen, stelle man sich vor, dass die Immatrikulationsnummer (MatNr) ein eindeutiger Verweis auf einen Studenten ist. Die URIs, die als eindeutige Verweise auf einen Studenten dienen, könnten die Form `http://www.uni-hannover.de/MatrNr/1813885/` haben. Nun könnte die Aussage folgendermaßen dargestellt werden:

## Beispiel 3:



*Das Individuum, auf das die Immatrikulationsnummer 1813885 verweist, heißt Helge Reinsch und hat die E-Mail Adresse `Helge.Reinsch@stud.uni-hannover.de`.*

## 2.3 RDF-Syntax

In RDF gibt es zwei mögliche Arten der Syntax. Zum einen die **Basic Serialization Syntax**, die jede Eigenschaft in einem eigenen Element ablegt, und die **Basic Abbreviated Syntax**, die es erlaubt, die Aussagen kompakter zu formulieren.

### 2.3.1 Basic Serialization Syntax

RDF-Aussagen treten selten allein auf. Viel üblicher ist es, dass mehrere Eigenschaften einer Ressource zusammen beschrieben werden. Die RDF-XML-Syntax wurde so entworfen, dass mehrere Aussagen über eine Ressource in einem `description`-Element gruppiert werden können. Das `description`-Element gibt in einem `about`-Attribut an, auf welche Ressource sich die Aussagen innerhalb des Elements beziehen. Sollte die Ressource bisher noch nicht existieren oder sollte es sich um eine physikalische Ressource handeln, die nicht über eine URI identifizierbar ist, so kann das `description`-Element diese identifizieren, indem sie das `id`-Attribut verwendet. Anders gesagt: das `id`-Attribut erzeugt eine neue Ressource, während das `about`-Attribut sich auf eine existierende bezieht. Ein einzelnes `description`-Element kann mehrere Eigenschaftselemente mit dem gleichen Namen beinhalten. Jedes dieser Elemente fügt dem gerichteten Graphen eine weitere Kante hinzu. Beispiel 1 sähe in dieser Notation wie folgt aus:

Beispiel 4:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:autor>Helge Reinsch</s:autor>
  </rdf:Description>
</rdf:RDF>
```

Die erste Zeile (`<?xml version="1.0"?>`) stellt die XML-Deklaration dar. Das `rdf`-Element in der zweiten Zeile ist lediglich ein Art Hülle, die die Grenzen innerhalb eines XML-Dokuments markiert, zwischen denen sich die Aussagen befinden. In dem `rdf`-Element werden außerdem noch zwei Namensräume angegeben. Zum einen der Namensraum von RDF selbst und zum anderen ein fiktiver Namensraum, in dem das Vokabular für Beschreibungen festgelegt sein könnte. Natürlich lässt sich einer der Namensräume auch als default namespace festlegen. Hier ein Beispiel, in dem der RDF-Namensraum als Vorgabe festgelegt ist:

Beispiel 5:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <Description about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:autor>Helge Reinsch</s:autor>
  </Description>
</rdf:RDF>
```

Wie man sieht, wird in diesem Fall, wie bereits aus XML bekannt, das Prefix vor den Elementen aus diesem Namensraum weggelassen. Darüber hinaus können Namensraumangaben auch für

einzelne Elemente innerhalb einer Beschreibung verwendet werden.

```
(<s:autor xmlns:s="http://description.org/schema/">Helge Reinsch</s:autor>)
```

Werte von Eigenschaften können aber nicht nur Zeichenketten, sondern auch wiederum Ressourcen sein. Im Beispiel 2 wird von dieser Möglichkeit Gebrauch gemacht. Der Wert von s:Autor ist die Ressource <http://www.uni-hannover.de/MatrNr/1813885/>.

Beispiel 6:

```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
    xmlns:s="http://description.org/schema/">

    <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
      <s:Autor rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/">
    </rdf:Description>

    <rdf:Description rdf:about="http://www.uni-hannover.de/MatrNr/1813885/">
      <s:Name>Helge Reinsch</s:Name>
      <s:Email>Helge.Reinsch@stud.uni-hannover.de</s:Email>
    </rdf:Description>
  </rdf:RDF>
```

Um das Beispiel zu erweitern, soll an dieser Stelle einmal angenommen werden, dass es zu der beschriebenen Seite Alternativseiten gibt, die sich mit demselben Thema beschäftigen. Eine mögliche Notation wäre:

Beispiel 7:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:autor>Helge Reinsch</s:autor>
    <s:alternativen>
      <rdf:alt>
        <rdf:alt rdf:resource="www.example1.com/alt1.htm"/>
        <rdf:alt rdf:resource="www.example2.com/alt2.htm"/>
      </rdf:alt>
    </s:alternativen>
  </rdf:Description>
</rdf:RDF>
```

Zum Schluss noch ein Beispiel, in dem mehrere Namensräume für die Beschreibung einer Ressource verwendet werden, um die Verwendung von Dublin Core zu zeigen.

## Beispiel 8:

```
<?xml version="1.0"?>
<rdf:RDF

  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s   = "http://description.org/schema/"
  xmlns:dc  = "http://purl.org/metadata/dublin_core#"

  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:autor>Helge Reinsch</s:autor>
    <dc:Title>Semantic Web: RDF (Resource Description Framework)</dc:Title>
    <dc:Subject>Dublin Core, Metadaten, Warwick Framework, RDF</dc:Subject>
    <dc:Author>Helge Reinsch</dc:Author>
    <dc>Date>2001-06-09</dc>Date>
    <dc:Identifier>http://www.Helge-Reinsch.de/ki2/</dc:Identifier>
    <dc:Language>de</dc:Language>
  </rdf:Description>

</rdf:RDF>
```

Die RDF zugrundeliegende BNF findet sich im Anhang [A.2](#).

### 2.3.2 Basic Abbreviated Syntax

Oft ist es wünschenswert, eine kompaktere Schreibweise zu verwenden. Diese Möglichkeit erhält man durch die RDF-Abbreviated-Syntax (abkürzende RDF Schreibweise). Es gibt drei Formen dieser abkürzenden Schreibweise.

Die erste Form kann für Eigenschaften verwendet werden, die innerhalb eines `description`-Elements nur einmal vorkommen und deren Werte Zeichenketten sind. In diesem Fall können die Eigenschaften als Attribute des `description`-Elements geschrieben werden.

## Beispiel 9:

```
<rdf:Description
  rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm"
  s:autor="Helge Reinsch"
  s:title="Semantic Web: RDF (Resource Description Framework)"
  s:date="2001-06-08" />
```

Ein weiterer Vorteil dieser Schreibweise ist, dass Browser, die kein RDF verstehen und deshalb die Werte der Eigenschaften, die zwischen den Elementtags stehen, anzeigen, bei der abgekürzten Schreibweise nichts anzeigen.

Die zweite abkürzende Schreibweise wirkt sich auf verschachtelte `description`-Elemente aus. Diese Schreibweise kann verwendet werden, wenn der Wert einer Eigenschaft eine Ressource ist, die innerhalb der gleichen Instanz beschrieben wird. In Beispiel 2 kann also diese abkürzende Schreibweise angewendet werden. Dazu nochmal die normale Schreibweise und anschließend die verkürzte:

## Beispiel 10:

```

<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
        xmlns:s="http://description.org/schema/">

  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:Autor rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/">
  </rdf:Description>

  <rdf:Description rdf:about="http://www.uni-hannover.de/MatrNr/1813885/">
    <s:Name>Helge Reinsch</s:Name>
    <s:Email>Helge.Reinsch@stud.uni-hannover.de</s:Email>
  </rdf:Description>

</rdf:RDF>

<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
        xmlns:s="http://description.org/schema/">

  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:Autor>
      <rdf:Description rdf:about="http://www.uni-hannover.de/MatrNr/1813885/">
        <s:Name>Helge Reinsch</s:Name>
        <s:Email>Helge.Reinsch@stud.uni-hannover.de</s:Email>
      </rdf:Description>
    </s:Autor>
  </rdf:Description>

</rdf:RDF>

```

Die ersten beiden Verkürzungen nutzend, kann man dieses Beispiel nun zusammenfassen zu:

## Beispiel 11:

```

<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
        xmlns:s="http://description.org/schema/">

  <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:Autor rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/"
            v:Name="Helge Reinsch"
            v:Email="Helge.Reinsch@stud.uni-hannover.de" />
  </rdf:Description>

</rdf:RDF>

```

Die dritte Form der Abkürzung kann verwendet werden, wenn das `description`-Element eine `type`-Eigenschaft enthält. `Type`-Eigenschaften werden später noch genauer erklärt. Die Aussage:

## Beispiel 12:

```
<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:Autor>
      <rdf:Description rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
        <rdf:type rdf.about="http://www.uni-hannover.de/MatrNr/1813885/person"/>
        <s:NameHelge Reinsch</s:Name>
        <s:Email>Helge.Reinsch@stud.uni-hannover.de</s:Email>
      </rdf:Description>
    </s:Autor>
  </rdf:Description>
</rdf:RDF>
```

kann mit Hilfe der dritten Abkürzung geschrieben werden als:

```
<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105\#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <s:Autor>
      <s:person rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
        <v:Name>Helge Reinsch</v:Name>
        <v:Email>Helge.Reinsch@stud.uni-hannover.de</v:Email>
      </rdf:Description>
    </s:Autor>
  </rdf:Description>
</rdf:RDF>
```

### 2.3.3 Container

Manchmal ist es notwendig, eine Gruppe von Ressourcen bereitzustellen. Dies wurde bereits im Beispiel 6 bei der Angabe von Alternativen verwandt. RDF stellt drei Typen von Containern zur Verfügung, die solche Gruppen beinhalten können.

Bag	beschreibt eine nicht sortierte Liste von Ressourcen oder Zeichenketten. Bags werden verwendet, wenn eine Eigenschaft mehrere Werte haben kann, aber es keine spezielle Anforderung an die Anordnung gibt.
Sequence	bezeichnet eine geordnete Liste von Ressourcen oder Zeichenketten. Sequence wird verwendet, wenn eine Eigenschaft mehrere Werte haben kann und es auf die Reihenfolge dieser Werte ankommt.
Alternative	beschreibt eine Liste von Ressourcen oder Zeichenketten, die Alternativen für den Wert dieser Eigenschaft darstellen.

Ein Beispiel in diesem Zusammenhang wäre die Liste aller Teilnehmer eines Kurses:

Beispiel 13:

```
<rdf:RDF>
  <rdf:Description about="http://meineuni.edu/seminar/rdf">
    <s:studenten>
      <rdf:Bag>
        <rdf:li resource="http://meineuni.edu/MatrNr/1235658"/>
        <rdf:li resource="http://meineuni.edu/MatrNr/1254687"/>
        <rdf:li resource="http://meineuni.edu/MatrNr/1864525"/>
        <rdf:li resource="http://meineuni.edu/MatrNr/1813885"/>
        <rdf:li resource="http://meineuni.edu/MatrNr/1764587"/>
      </rdf:Bag>
    </s:studenten>
  </rdf:Description>
</rdf:RDF>
```

### 2.3.4 Aussagen über Aussagen

Zusätzlich zu der Eigenschaft, dass mit RDF Aussagen über Ressourcen gemacht werden können, bietet RDF auch noch die Möglichkeit, Aussagen über Aussagen zu treffen. Diese sollen als Aussagen höherer Ordnung (High-order statements) bezeichnet werden.

Dazu betrachte man die Aussage:

*Helge Reinsch ist der Autor der Seite <http://www.Helge-Reinsch.de/ki2/rdf.htm>*

und die Aussage:

*Wolf Siberski sagt, dass Helge Reinsch der Autor der Seite <http://Helge-Reinsch.de/ki2/rdf.htm> ist.*

Mit der zweiten Aussage wird nichts über die Ressource <http://Helge-Reinsch.de/ki2/rdf.htm> ausgesagt. Stattdessen wird eine Aussage über eine Aussage von Wolf Siberski gemacht. Um diese Fakten in RDF auszudrücken, muss die ursprüngliche Aussage als eine Ressource mit vier Eigenschaften modelliert werden. Im Bereich Wissensrepräsentation wird dies als Vergegenständlichung (reification) bezeichnet. Um Aussagen zu modellieren, stellt RDF folgende Eigenschaften zur Verfügung:

subject	Die subject-Eigenschaft identifiziert die Ressource, die durch das modellierte Statement beschrieben wird. Das bedeutet, der Wert des subjects ist die Ressource, über die die ursprüngliche Aussage gemacht wurde. ( <a href="http://www.Helge-Reinsch.de/ki2/rdf.htm">http://www.Helge-Reinsch.de/ki2/rdf.htm</a> )
---------	---

predicate	Die predicate-Eigenschaft verweist auf die ursprüngliche Eigenschaft in der modellierten Aussage. Der Wert von predicate ist die Ressource, die durch die spezielle Eigenschaft in der ursprünglichen Aussage spezifiziert wurde (hier: autor).
object	Die object-Eigenschaft verweist auf den Eigenschaftswert in der modellierten Aussage. Der Wert der object-Eigenschaft ist das Objekt der ursprünglichen Aussage (hier: Helge Reinsch).
type	Der Wert von type beschreibt den Typ der neuen Ressource. Alle "Vergegenständlichungen" sind Instanzen von rdf:Statement. Das bedeutet, sie haben eine type-Eigenschaft, deren Objekt ein rdf:Statement ist. Die type-Eigenschaft wird allgemeiner dazu benutzt, Typen beliebiger Ressourcen zu deklarieren, so wie Container.

Notiert sähe die Beispielaussage so aus:

Beispiel 14:

```
<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.Helge-Reinsch.de/ki2/rdf.htm" />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Helge Reinsch</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
    <a:attributedTo>Wolf Siberski</a:attributedTo>
  </rdf:Description>
</rdf:RDF>
```

## 2.4 RDF-Schema

RDF selbst bietet keine Möglichkeiten, die Beziehungen zwischen Eigenschaften und Ressourcen zu definieren. Um diese Beziehungen zu deklarieren, werden RDF-Schemata benutzt. Ressourcenbeschreibungen benötigen die Fähigkeit, bestimmte Dinge über bestimmte Ressourcen zu auszusagen. Um bibliographische Ressourcen zu beschreiben, beinhalten beschreibende Attribute für gewöhnlich "Autor", "Titel" und "Thema". Digitale Zertifikate könnten Attribute der Art "Checksumme" und "Authorisation" besitzen. Diese Eigenschaften (Attribute) und die dazugehörige Semantik werden, im Zusammenhang mit RDF, in RDF-Schemata deklariert. Das Schema definiert nicht nur die Eigenschaften (Titel, Autor, Farbe), sondern kann darüberhinaus die Art der Ressource (Buch, Webseite, Person, Firma) definieren, zu der diese Eigenschaften gehören.

Diese Dokumente beschreiben nicht ein Vokabular deskriptiver Elemente, stattdessen beschreiben sie einen Mechanismus, der benötigt wird, um Elemente festzulegen, um Klassen zu definieren, in denen sie benutzt werden, um mögliche Kombinationen von Klassen und Relationen

zu beschränken und Verletzungen dieser Beschränkungen zu erkennen.

Die Schema-Beschreibungssprache ist einfach eine Menge von Ressourcen und Eigenschaften, die durch die RDF-Schema-Spezifikation definiert wurden und implizit Bestandteil jedes RDF-Modells, das auf RDF-Schemata aufbaut, sind. RDF-Schemata definieren Eigenschaften in den Begriffen der Klassen von Ressourcen, zu denen sie gehören.

Das RDF-Schema-System ist ähnlich dem Klassensystem objektorientierter Sprachen. Allerdings besteht ein wesentlicher Unterschied: Bei den objektorientierten Sprachen wird eine Klasse von Objekten dadurch definiert, welche Eigenschaften (in C++ z.B. Membervariablen) sie besitzt. Bei den RDF-Schemata wird von der Idee ausgegangen, dass Klassen von Objekten auch identifizierbar sind, ohne dass ihnen sofort bestimmte Eigenschaften zugeordnet werden. Dagegen wird für jede Eigenschaft festgelegt, für welche Klassen von Objekten sie anwendbar ist.

Dies stellt eine wesentlich flexiblere Lösung dar, da die interessierenden Eigenschaften ganz wesentlich vom Anwendungsbereich abhängen. Mit dem RDF-Schema-System können bei Bedarf jederzeit neue Eigenschaften definiert und bestimmten Klassen zugeordnet werden, ohne dass bestehende Anwendungen beeinträchtigt werden. Außerdem können so definierte Eigenschaften mit ihrer Charakteristik, wie Bedeutung, Wertebereich usw., sofort für eine andere Klasse unter Beibehaltung dieser Charakteristik verwendet werden.

In dem Dokument [RDFS99] wird nun nicht ein Schema definiert, in dem verschiedene Klassen und Eigenschaften festgelegt werden, sondern eine "Schema Definition Language", mit deren Hilfe die eigentlichen Schemata definiert werden. Diese eigentlichen Schemata werden auch als Vokabulare bezeichnet, da sie festlegen, welche Eigenschaften mit welcher Bedeutung für Beschreibungen zu Verfügung stehen.[15]

#### 2.4.1 Kernklassen (Core Classes)

rdfs:Resource	bezeichnet die Klasse aller Ressourcen, d.h. alle Dinge, die mit RDF beschrieben werden, sind Instanzen dieser Klasse.
rdfs:Property	ist die Klasse der Eigenschaften. Sie repräsentiert eine Untermenge der Ressourcen.
rdfs:Class	korrespondiert zum generischen Konzept eines Typs, alle Klassen sind Instanzen des Typs Klasse (auch die Klasse rdfs:Class selbst). Ein ähnliches Konzept existiert z.B. in Java. Beachtenswert ist hier, dass einerseits rdfs:Class eine Unterklasse von rdfs:Resource ist, da die Klassendefinitionen selbst Ressourcen sind, andererseits rdfs:Resource auch eine Instanz von rdfs:Class ist, da es sich um eine Klasse handelt.

### 2.4.2 Kerneigenschaften (Core Properties)

Es werden die folgenden grundlegenden Eigenschaften festgelegt:

rdf:type	ermöglicht die Zuordnung einer Ressource zu einer Klasse. Die Ressource ist dann Instanz der Klasse, und es kann angenommen werden, dass sie der typischen Charakteristik der Klasse entspricht.
rdfs:subClassOf	spezifiziert eine Untermengen/Obermengen-Relation zwischen Klassen. Die Relation ist transitiv. Ressourcen, die Instanzen einer bestimmten Klasse sind, sind auch Instanzen von deren Oberklassen.
rdfs:subPropertyOf	ist eine Instanz von rdf:Property und wird benutzt, um anzugeben, dass eine Eigenschaft eine Spezialisierung einer oder mehrerer anderer Eigenschaften ist. Wenn eine Eigenschaft E2 Spezialisierung einer anderen Eigenschaft E1 ist, und eine Ressource hat eine Eigenschaft E2 mit dem Wert B, dann wird damit implizit ausgedrückt, dass sie auch eine Eigenschaft E1 mit Wert B hat. Beispiel: Wenn biologicalFather eine Spezialisierung von biologicalParent ist, und Max ist der biologicalFather von John, dann ist Max auch ein biologicalParent von John.
rdf:seeAlso	verweist auf eine andere Ressource, die Informationen über die vorliegende Ressource enthält.
rdfs:isDefinedBy	ist eine Spezialisierung von rdfs:seeAlso und verweist auf eine Ressource, die die vorliegende Ressource definiert.

### 2.4.3 Beschränkungen (Constraints)

In einem RDF-Schema können den Klassen und Eigenschaften Beschränkungen zugeordnet werden. Insbesondere werden damit die Konzepte von Domain und Range umgesetzt. Die Domain einer Eigenschaft gibt die Klassen an, auf die die Eigenschaft sinnvollerweise angewendet werden kann. Die Range legt die gültigen Werte für eine Eigenschaft fest. So könnte z.B. festgelegt werden, dass die Werte einer Eigenschaft "Autor" nur vom Typ "Person" sein dürfen und die Eigenschaft nur bei Ressourcen vom Typ "Buch" angewendet werden darf.

Das RDF-Schema verwendet die Beschränkungs-Eigenschaften rdfs:range und rdfs:domain, um festzulegen, in welcher Weise seine Eigenschaften benutzt werden dürfen.[16]

Beispiel

- ```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303\#">
```
- ```
<rdfs:Class rdf:ID="Person">
  <rdfs:comment>The class of people.</rdfs:comment>
```

```

    <rdfs:subClassOf
      rdf:resource="http://www.classtypes.org/useful/_classes/#Animal"/>
</rdfs:Class>

3. <rdf:Property ID="age">
    <rdfs:range
      rdf:resource="http://www.datatypes.org/useful/_types/#Integer"/>
    <rdfs:domain rdf:resource="\#Person"/>
</rdf:Property>

4. <rdf:Property ID="maritalStatus">
    <rdfs:range rdf:resource="\#MaritalStatus"/>
    <rdfs:domain rdf:resource="\#Person"/>
</rdf:Property>

5. <rdfs:Class rdf:ID="MaritalStatus"/>

6. <MaritalStatus rdf:ID="Married"/>
    <MaritalStatus rdf:ID="Divorced"/>
    <MaritalStatus rdf:ID="Single"/>
    <MaritalStatus rdf:ID="Widowed"/>
</rdf:RDF>

```

Im Abschnitt 1 wird zunächst die Sprache Englisch für das Dokument angegeben, und es werden Namespaces für die RDF-Syntax und das RDF-Schema festgelegt. Abschnitt 2 definiert eine Klasse *Person*, die Unterklasse der Klasse *Animal* ist. Als Beschreibung wird "The class of people" angegeben. Der Abschnitt 3 definiert eine Eigenschaft *age*, die auf Instanzen der Klasse *Person* zutrifft, der gültige Wertebereich der Eigenschaft sind Werte vom Typ *integer*. Im Abschnitt 4 wird eine Eigenschaft *maritalStatus* definiert, die ebenso auf Instanzen der Klasse *Person* angewendet werden kann. Der gültige Wertebereich sind dabei Instanzen der Klasse *MaritalStatus*, die in Abschnitt 5 definiert werden. Instanzen dieser Klasse werden dann in Abschnitt 6 definiert.

## 3 Edutella

### 3.1 Allgemeines

Jede Universität besitzt eine Vielzahl von Lehrmaterialien verteilt über die Institute. Diese Lehrmaterialien werden von den Instituten nur ungern zentralen Einrichtungen überlassen aus Angst, die Kontrolle darüber zu verlieren.

Um die Lehrmaterialien bereitzustellen und trotzdem die Kontrolle zu behalten, könnten die Institute diese auf Servern bereitstellen, auf die anfragegesteuert zugegriffen werden kann. Um die eingangs beschriebenen Probleme mit herkömmlichen Suchmaschinen zu umgehen, können hier Peer-to-Peer Netzwerke eingesetzt werden.

In einem typischen Peer-to-Peer E-Learning Szenario fungieren die einzelnen Universitäten nicht nur als Anbieter, sondern ebenfalls als Nachfrager von Inhalten.

Als Anbieter in einem Peer-to-Peer Netzwerk verlieren sie die Kontrolle über ihre Lehrmaterialien nicht und stellen sie dennoch innerhalb des Netzwerkes den anderen Mitgliedern zur Verfügung.

Als Nachfrager ziehen sowohl Lehrende als auch Studierende Nutzen aus dem Zugang, nicht nur zu lokalen als vielmehr zu einem ganzen Netzwerk von Lehrmaterialien. Sie können durch die Verwendung von Anfragen über die gesamten Metadaten des Netzwerkes die benötigten Quellen finden und so mit einer großen und aktuellen Auswahl von Forschungsergebnissen arbeiten.

Um Lehrmaterialien in einem solchen Netzwerk zu finden, werden die Veröffentlichungen mit Metadaten versehen, um das Thema und den Inhalt der Arbeit zu charakterisieren. Da Lehrmaterialien oftmals in einem hohen Maße fachgebiets- und quellenspezifisch sind, wird eine Auszeichnungssprache für Metadaten benötigt, die die Interoperabilität und die Wiederverwendbarkeit von Lehrmaterialien und einen großen Bereich der Quellen von Lehrmaterialien unterstützt.

Edutella als metadatenbasiertes Peer-to-Peer System ist deshalb in der Lage, heterogene Peers (die verschiedene Quellen, verschiedene Anfragesprachen und verschiedene Funktionalitäten verwenden) ebenso zu integrieren, wie verschiedene Sorten von Metadaten-Schemata.[7]

### 3.2 JXTA-Project

Als Grundlage für das Peer-to-Peer Netzwerk, auf das Edutella aufbaut, wurde das [Projekt JXTA](#) ausgewählt. JXTA ist ein open source Projekt von Sun Microsystems, dessen Ziel es ist, eine Grundlage für die Implementierung von Peer-To-Peer Anwendungen zu schaffen. Die Idee, ein Projekt als open source ins Leben zu rufen, ist, dass ein gegenseitiger Austausch von Technologien und Ideen den Entwicklungsprozess und die Kreativität der Entwickler steigert.[3]

Sun arbeitet dabei mit Firmen zusammen, die sich dem open source Modell verpflichtet haben und in Peer-To-Peer Technologie investieren. Sie teilen ihre Ergebnisse miteinander und ermög-

lichen es Entwicklern auf diese Weise von ihren Erkenntnissen zu profitieren und eigene neue fachgebietspezifische Peer-Services und Anwendungen zu erzeugen.

### 3.3 Das JXTA Peer-to-Peer System

Das Projekt JXTA stellt eine Sammlung einfacher, kleiner und flexibler Mechanismen bereit, die den Aufbau eines Peer-to-Peer Netzwerkes auf jeder Plattform unterstützen.

JXTA verwendet offene Standards wie XML, Java und Schlüsselkonzepte, wie die Möglichkeit, in Shells Kommandos miteinander durch Pipes zu verbinden, um komplexe Aufgaben zu verwirklichen.

Die Abbildung 1 stellt das Layerkonzept dar, das JXTA zugrundeliegt.

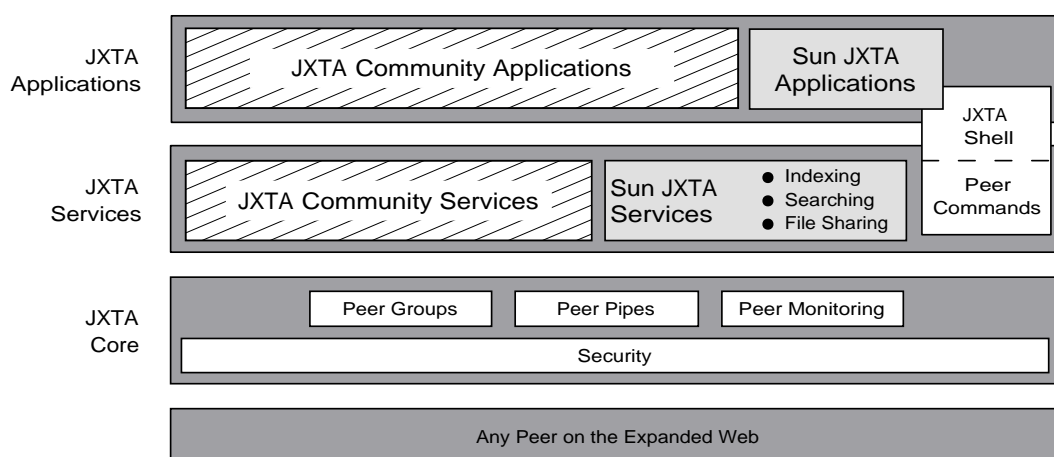


Abbildung 1: JXTA Layerkonzept

Im Kern der Funktionalitäten stehen die Möglichkeiten, Peer-Gruppen zu erzeugen und zu löschen, die Gruppen den potentiellen Mitgliedern bekannt zu geben und anderen zu ermöglichen, die Gruppen zu finden, sich ihnen anzuschließen und sie zu verlassen.

Auf dem nächsten Layer bestehen die Kernmöglichkeiten darin, einige Peer-Services zu erzeugen, inklusive Indizierung, Suchen und File-sharing. Darüber können Peer-Anwendungen erzeugt werden, die diese Mittel nutzen.

**JXTA-Core** Der JXTA-Core stellt die Unterstützung von Peer-to-Peer Services bereit. Er besteht aus:

Peer groups

stellen eine Zusammenfassung von Peers dar. Es werden Mechanismen zum Erzeugen und Löschen, Erlangung einer Mitgliedschaft, Veröffentlichung und Auffinden anderer Peer-Gruppen und Peer-Knoten, zur Kommunikation, für Sicherheit und gemeinschaftlich genutzer Inhalte zur Verfügung gestellt.

Peer pipes	stellen Kommunikationskanäle zwischen Peers bereit. Nachrichten, die innerhalb von Pipes verschickt werden, liegt XML als Strukturierung zugrunde.
Peer monitoring	ermöglicht das Überwachen von Verhalten und Aktivität eines Peers in einer Peer-Gruppe und kann verwendet werden, um Peer-Management-Funktionen zu implementieren inklusive Zugriffskontrolle, Prioritäten, Messung von Datentransfer und Bandbreitenausgleich.

### **JXTA Services**

So wie verschiedene Bibliotheken in UNIX-Systemen Funktionen eines höheren Levels als des Kernels unterstützen, erstrecken sich JXTA Services über die Fähigkeiten des Kerns und erleichtern die Anwendungsentwicklung. Die Möglichkeiten, die in diesem Layer zur Verfügung gestellt werden, beinhalten Mechanismen zur Suche, gemeinsamen Nutzung und Indizierung.

### **JXTA Applications**

In diesem Layer sind die eigentlichen Anwendungen des Peer-to-Peer Netzwerkes, wie auch Edutella angesiedelt.

## **3.4 Technische Konzepte**

Auf dem höchsten Level der Abstraktion ist die JXTA-Technologie eine Menge von Protokollen. Jedes der Protokolle ist definiert über eine oder mehrere Nachrichten, die zwischen den Teilnehmenden ausgetauscht werden. Jede dieser Nachrichten hat ein festgelegtes Format und kann verschiedene Datenfelder beinhalten.

In dieser Hinsicht ist es mit TCP/IP verwandt. Während TCP/IP Internetknoten miteinander verbindet, verbindet die JXTA Technologie Peer-Knoten miteinander. TCT/IP erreicht seine Plattformunabhängigkeit, indem es eine Reihe von Protokollen bereitstellt. Ebenso verhält es sich bei JXTA. Darüber hinaus ist die JXTA-Technologie ebenfalls unabhängig von der Art des Transportes. Es kann TCP/IP ebenso verwenden, wie jeder andere Transportstandard.

Momentan sind folgende Protokolle definiert:

Peer Resolver Protocol (PRP)	ist der Mechanismus mit dessen Hilfe Peers Anfragen zu einem oder mehreren Peers senden und Antworten auf diese Anfragen empfangen können. Das PRP implementiert ein Anfrage- und Antwort-Protokoll. Die Antwort wird der Anfrage über eine eindeutige ID zugeordnet, die innerhalb der Nachricht angegeben wird. Anfragen können zu einer gesamten Peer-Gruppe oder zu einem einzelnen Peer gesendet werden.[6]
------------------------------	--

Peer Discovery Protocol (PDP)	Mit Hilfe dieses Protokolls können Peers ihre eigenen Ressourcen bekanntgeben (advertise) und die Ressourcen anderer Peers (Peer-Gruppen, Services, Pipes und weiterer Peers) finden. Jede Peer-Ressource wird mit Hilfe solcher Bekanntmachungen (advertisements) beschrieben und veröffentlicht. Advertisements sind programmiersprachenneutrale Metadatenstrukturen, die Netzwerkressourcen beschreiben. Sie werden durch XML-Dokumente repräsentiert.[6]
Peer Information Protocol (PIP)	Mit Hilfe dieses Mechanismus kann ein Peer Statusinformationen – wie Status, Kapazität, Transferaufkommen, Onlinezeiten, etc. – über andere Peers bereitstellen.[6]
Pipe Binding Protokoll (PBP)	Mit diesem Mechanismus können Peers einen virtuellen Kommunikationskanal oder eine Pipe zwischen einem oder mehreren Peers errichten. Das PBP wird von einem Peer verwendet, um einen oder mehrere Kommunikationsendpunkte (pipe endpoints) zu binden. Pipes stellen den grundlegenden Kommunikationsmechanismus zwischen Peers zur Verfügung.[6]
Endpoint Routing Protocol (ERP)	Durch diesen Mechanismus kann ein Peer eine Route (oder eine Sequenz von Hops) finden, die verwendet werden, um Nachrichten zu einem anderen Peer zu senden. Wenn der Peer "A" eine Nachricht an den Peer "C" senden möchte und es keine direkte Verbindung zwischen "A" und "C" gibt, muss der Peer "A" dazwischenliegende Peers finden, die die Nachricht zu Peer "C" weiterleiten. Das ERP wird verwendet, um diese Routeninformation zu bestimmen. Wenn sich die Netzwerk-Topologie geändert hat, so dass die Route zum Peer "C" nicht länger verwendet werden kann, weil eine Verbindung entlang der Route nicht länger funktioniert, kann der Peer ERP verwenden, um eine Route, die andere Peers kennt, zu "C" zu finden.[6]
Rendezvous Protocol (RVP)	Mit diesem Mechanismus kann sich ein Peer bei einem Propagation Service anmelden. Innerhalb einer Peer-Gruppe können Peers Rendezvouspeers werden oder Peers, die an Rendezvouspeers "lauschen". Das Rendezvous Protokoll erlaubt Peers, Nachrichten an alle Instanzen eines Services zu senden, die an dem Rendezvouspeer lauschen. Das RVP wird von dem Peer Resolver Protocol und dem Pipe Binding Protocol verwendet, um Nachrichten zu verbreiten.[6]

Um die Protokolle zu untermauern, wurde eine Reihe von Konzepten definiert, die Peers, Peer-Gruppen, Advertisements, Nachrichten, Pipes und mehr enthalten.

Identifiers	JXTA verwendet UUID, einen 128-bit Wert zur Referenzierung von Entitäten (einem Peer, Advertisement, Service, usw.). Es lässt sich leicht garantieren, dass jede Entität eine eindeutige UUID innerhalb einer lokalen Umgebung besitzt. Es gibt allerdings keinen absoluten Weg, eine eindeutige UUID über alle Peers zu gewährleisten, da es Millionen von Peers geben kann. Das ist jedoch kein schwerwiegendes Problem, weil die UUID nur als interne Identifizierung verwendet wird. Es wird erst entscheidend, wenn sie für Sicherheits- oder andere Informationen, wie Netzwerkadressen, verwendet werden.[5]
Advertisements	Ein Advertisement ist ein in XML strukturiertes Dokument, das die Existenz von Ressourcen, wie Peers, Peer-Gruppen und Pipes, und eines Service benennt, beschreibt und veröffentlicht. Die JXTA-Technologie definiert eine grundlegende Menge von Advertisements. Weitere Advertisement-Untertypen können mit Hilfe dieser grundlegenden Typen erstellt werden, indem das zugehörige XML-Schema verwendet wird.[5]
Peers	Ein Peer ist eine beliebige Entität, die die Sprache der Protokolle spricht, die für einen Peer benötigt werden. Dies ist ähnlich dem Internet, wo ein Internetknoten alles sein kann, was in der Lage ist, die Gruppe der IP-Protokolle zu verwenden. Wichtig ist, dass ein Peer nicht alle sechs Protokolle, die anfangs angegeben wurden, verstehen muss. Er kann auf einem reduzierten Level arbeiten, wenn er nicht alle Protokolle unterstützt.[5]
Messages	Messages sind dazu gedacht, bei hochgradiger Asynchronität, Unzuverlässigkeit und Einweg-Transport verwendet zu werden. Deshalb ist eine Message als Datagramm gedacht, das eine Hülle und einen Stapel Protokoll-Header mit Inhalt enthält. Die Hülle enthält einen Kopf, eine Nachrichten-Übersicht, optional einen Quellen-Endpunkt und den Ziel-Endpunkt. Ein Endpunkt ist ein logisches Ziel, gegeben in Form einer URI. Endpunkte zeigen typischerweise auf physikalische Adressen eines Nachrichtenschichtes. Ein solches Nachrichtenformat ist dazu ausgelegt, verschiedene Transportstandards zu unterstützen. Jeder Protokollinhalt beinhaltet eine variable Menge von Bytes und eine oder mehrere Nachweise, die den Sender gegenüber dem Empfänger identifizieren. Das genaue Format eines solchen Nachweises über die Herkunft ist nicht festgelegt.[5]

**Peer Groups** Eine Peer-Gruppe ist eine virtuelle Entität, die einige Peer-Gruppen Protokolle versteht. Typischerweise ist eine Peer-Gruppe eine Sammlung von zusammen arbeitenden Peers, die einen gemeinsamen Service anbieten. Die JXTA-Spezifikation schreibt nicht vor, wann, wo oder warum eine Peer-Gruppe erstellt werden soll, von welchem Typ sie ist oder wie eine Mitgliedschaft aussieht. Sie definiert nicht einmal, wie die Gruppe erzeugt wird. In Wirklichkeit kann die Beziehung zwischen einem Peer und einer Peergruppe beliebig sein. Es besteht auch keine Begrenzung darüber, wievielen Gruppen ein Peer angehören darf oder ob verschachtelte Gruppen erzeugt werden dürfen. Es wird lediglich definiert, wie die Gruppen durch Verwendung des Peer Discover Protocols gefunden werden. Es existiert eine spezielle Peergruppe mit dem Namen "World Peer Group", die alle JXTA-Peers beinhaltet.[5]

**Pipes** Pipes sind Kommunikationskanäle zum Senden und Empfangen von Nachrichten. Sie laufen asynchron und sind ebenso unidirektional, weshalb es Input- und Output-Pipes gibt. Sie sind ebenfalls virtuell, indem ihre Endpunkte an einen oder mehrere Peer-Endpunkte gebunden sein können.

Eine Pipe ist üblicherweise zur Laufzeit dynamisch an einen Peer über das Pipe Binding Protokoll gebunden. Dies bedeutet insbesondere, dass eine Pipe bewegt werden kann und zu verschiedenen Zeiten an verschiedene Peers gebunden sein kann. Dies ist z.B. sinnvoll, wenn eine Gruppe von Peers zusammen einen hohen Level von Fehlertoleranz bereitstellen möchten. Wenn ein Peer nicht verfügbar ist, kann ein anderer Peer an einem anderen Ort einspringen und die begonnene Kommunikation weiterführen.

Eine *Point-to-Point Pipe* verbindet genau zwei Peerendpunkte miteinander. Die Pipe ist eine Output-Pipe für den Sender und eine Input-Pipe für den Empfänger, wobei der Transfer immer nur in eine Richtung geht – vom Sender zum Empfänger.

Eine *Propagate Pipe* verbindet multiple Peerendpunkte miteinander, von einer Output-Pipe zu einer oder mehreren Input-Pipes. Das Ergebnis ist, dass jede Nachricht, die in die Output-Pipe geschickt wird, an alle Input-Pipes gesendet wird.[5]

### 3.5 Edutella Peers

Das Edutella-Netzwerk ist in der Lage, heterogene Peers mit unterschiedlichen Datenquellen, verschiedenen Anfragesprachen und unterschiedlichen Metadaten-Schemata zu integrieren.

ren. Darüber hinaus sind die Peers sehr verschieden in den Zeitabschnitten, in denen sie mit dem Netzwerk verbunden sind. Z.B. könnte ein Edutella-Peer eine große Datenbank repräsentieren, die zu 99% verfügbar ist, oder einen anderen Peer, der nur gelegentlich online ist und nur eine geringe Menge an Metadaten zur Verfügung stellt.[8]

### 3.6 Edutella Services

Die Metadaten werden in Form von RDF-Statements zur Verfügung gestellt. Die Andersartigkeit der verschiedenen Metadaten der RDF-Peers, die mit dem Edutella Netzwerk verbunden sind, wird vollständig transparent durch verschiedene Services, die der Peer anbietet, charakterisiert.

#### 3.6.1 Query Service

Der Query Service ist der grundlegendste Service im Edutella-Netzwerk. Peers registrieren sich für die Anfragen, die sie beantworten können. Anfragen, die an das Edutella Netzwerk gestellt werden, werden an die Teilmenge von Peers weitergeleitet, die sich mit ihrem Service für diese Art von Anfragen registriert haben.

So können sich die Peers für spezielle Metadaten-Schemata, wie "dieser Peer bietet Metadaten konform dem LOM 6.1 oder DCMI Standard an" registrieren, oder es können Eigenschaften und Werte für diese Eigenschaften angegeben werden, wie z.B. "dieser Peer bietet Metadaten der Form *dc\_title(x,y)*" oder "dieser Peer bietet Metadaten der Form *dc\_title(x,'Artificial Intelligence')*". Anfragen, die durch das Edutella-Netzwerk gesendet werden, werden zu den Peers weitergeleitet, die sich für Anfragen diesen Typs registriert haben. Die resultierenden Antworten werden an den nachfragenden Peer zurückgesendet.[9]

Der Edutella-Query Service ist ein standardisierter Abfrage-Austausch-Mechanismus für RDF-Metadaten, die in verteilten RDF-Datenquellen gespeichert sind, und dient sowohl als Anfrageinterface für individuelle RDF-Datenquellen eines einzelnen Edutella-Peers wie auch als Anfrageinterface für verteilte Anfragen, die mehrere RDF-Datenquellen umfassen.

Eine RDF-Datenquelle (oder Wissensbasis) besteht aus RDF-Statements (oder Fakten) und beschreibt Metadaten basierend auf beliebigen RDF-Schemata. Die RDF-Statements können z.B. in Textdateien oder Datenbanken gespeichert werden.

Es ist daher notwendig, von allen möglichen Anfragesprachen, die denkbar sind, um RDF-Statements aus lokalen Speichern zu extrahieren (z.B. SQL), zu abstrahieren und einen Mechanismus zur Verfügung zu stellen, der es ermöglicht, standardisiert Anfragen nach RDF-Metadaten auszutauschen. Die Edutella-Query Exchange Language und das Edutella Common Data Model stellen die Syntax und die Semantik für ein solches allgemeines Standard-Anfrageinterface für heterogene RDF-Datenquellen bereit. Das Edutella-Netzwerk verwendet die Familie von Anfrage-Austausch-Sprachen RDF-QEL, die auf der Datalogsemantik basieren und eine Teilmenge davon sind, als standardisiertes Anfrage-Austausch-Format, das in RDF/XML-Format übertragen wird.[7]

### 3.6.2 Edutella Annotation

Im selben Sinn, wie die Metadaten für ein spezielles Dokument leicht bereitgestellt werden sollen, bietet der Annotation Service einen Dokument-Viewer. Derzeit kann der Dokument-Viewer HTML Seiten anzeigen, eine Erweiterung für PDF-Dateien ist in Arbeit.

Darüber hinaus stellt der Annotation Service einen Browser für RDF-Schemata bereit. Das bedeutet, dass eine korrespondierende Definition, z.B. Dublin Core, in das Annotation-Tool geladen werden und dort betrachtet werden kann.

Beschreibende Felder werden passend zu den Schemadefinitionen angezeigt und können durch einfaches Eintippen oder Drag & Drop von Dokumenten-Viewer eingetragen werden.

Nebenbei können Annotationen und Felder für Annotationen verschiedene Gestalt annehmen. In diesem Zusammenhang ist eine Annotation eine Menge von Umschreibungen, die an das HTML-Dokument angehängt werden.

Es werden unterschieden:

- Umschreibungen für RDFS-Klassen
- Umschreibende Eigenschaften einer Klasseninstanz über eine Datentypinstanz
- instantiierte Eigenschaften von einer Klasseninstanz zu einer anderen Klasseninstanz

Klasseninstanzen haben eindeutige URIs. Umschreibungen können an einzelne Markups innerhalb des HTML-Dokuments gehängt werden, URIs und Attributwerte können als Zeichenketten im HTML-Text erscheinen.

Man kann sich also z.B. entscheiden zwischen:

1. Eine Identifizierung für eine Person zu erstellen, indem man `HTTP://WWW.AIFB.UNIKARLSRUHE.DE/WBS/SHA/#HANDSCHUH` von der Klasse `DC:CREATOR` instantiiert und `HTTP://WWW.AIFB.UNIKARLSRUHE.DE/LEHRVERANSTALTUNGEN/WINTER/EBIZ+INTELLIGENTWEB/#COURSE` von der Klasse `SWRC:SEMINAR`.
2. Die Attribute der ersten Identifizierung durch Namen wie "Siegfried Handschuh" oder "Siggi" instantiiert.
3. Die Instanzen in Beziehung setzt, z.B. indem die erste Identifizierung mit der zweiten über die Eigenschaft `SWRC:TEACHES` in Beziehung gesetzt wird.

Diese Arten der Instanziierung können als eine Dimension des Prozesses zur Erstellung von Metadaten angesehen werden.

Eine andere, orthogonale Dimension ist durch die Art, wie Umschreibungen erzeugt, benutzt und gewartet werden, definiert:

Unlinked Facts	sind Felder des Schemas. Es gibt keinen Zusammenhang zu dem betrachteten Dokument, das durch die Maschine entdeckt wurde.
Quotations	sind Zitate aus dem Dokument. Z.B. könnte der Name "Tim Berners-Lee" in einem Dokument auftauchen und ebenso ein Feld im beschreibenden RDF-Schema füllen.
References	sind Zeiger auf einen Teil eines Dokuments. Es werden XPointer verwendet, um auf Teile von Dokumenten zu zeigen. Z.B. könnte man sagen, dass eine bestimmte Zelle einer HTML-Tabelle den Namen des Präsidenten der USA beinhaltet, und in diesem Zusammenhang würde man erwarten, dass sich der Wert ändert, wenn ein neuer Präsident gewählt würde.

[9]

### 3.6.3 Edutella Replication

Dieser Service ergänzt den lokalen Speicher indem er Daten auf andere Peers repliziert, um eine bessere Verfügbarkeit und Persistenz der Daten und eine bessere Lastenverteilung bei Erhaltung der Datenintegrität und Konsistenz zu ermöglichen.[7]

### 3.6.4 Edutella Mapping, Mediation, Clustering

Während sich Peer-Gruppen normalerweise auf die Verwendung eines gemeinsamen Schemas einigen, könnten Erweiterungen und Veränderungen in einigen Bereichen notwendig sein. Der Edutella Mapping Service ist in der Lage, Übersetzungen zwischen verschiedenen Schemata zu verwalten und diese Übersetzungen dazu zu benutzen, Anfragen auf dem Schema A in Anfragen auf dem Schema B zu übersetzen. Darüber hinaus soll der Mapping Service die Zusammenarbeit von RDF- und XML-basierten Datenquellen ermöglichen.

Der Mediation Service vermittelt somit zwischen verschiedenen Services.

Der Cluster Service verwendet semantische Informationen, um semantisches Routing und semantische Cluster zu erstellen.[7]

## 4 Edutella Query Model (EQM)

Das Edutella Query Model (EQM) ist das Datenmodell, das Anfragen und deren Ergebnisse innerhalb des Edutella Netzwerkes repräsentiert. Für die Notation von Anfragen, die im EQM gestellt werden sollen, soll im Weiteren Datalog verwendet werden, da der Aufbau anschaulich übereinstimmt.

### 4.1 Datalog

Datalog ist eine Sprache für logische Daten und Regeln. Mit Datalog kann eine Anfrage an eine Wissensbasis mit Fakten oder Aussagen gestellt werden. Eine Wissensbasis von Fakten enthält lediglich binäre Relationen beschrieben durch *predicate(subject,object)* oder als ternäre Aussage *s(subject,predicate,object)*.

Datalog ist eine nicht-prozedurale Anfragesprache, die auf Horn-Klauseln basiert. Eine Horn-Klausel ist eine Disjunktion von Literalen mit höchstens einem nicht-negativen Literal. Zusätzlich zu reinen Fakten sind Literale auch Prädikate, die eine beliebige Relation zwischen einer beliebigen Anzahl von Konstanten und Variablen ausdrücken.

#### 4.1.1 Prädikate

Prädikat-Ausdrücke sind die elementaren Konstrukte in Datalog. Ein Ausdruck für ein Prädikat besteht aus einem Prädikat-Symbol gefolgt von einer Liste von Argumenten. Zum Beispiel:

```

eqlas(X,Y)
title (book, "Artificial Intelligence")

```

Argumente können Werte (wie "Artificial Intelligence"), Konstanten (wie book) und Variablen (wie X) sein. Werte werden immer in Anführungszeichen gesetzt, Konstantenbezeichnungen immer in kleinen und Variablen immer in großen Buchstaben geschrieben. Prädikate werden gelegentlich auch als atomare Formeln bezeichnet, ebenso die Namen von Konstanten.[10]

#### 4.1.2 Fakten

Daten werden in Datalog durch Prädikat-Ausdrücke beschrieben, die nur Werte und Konstantenbezeichnungen als Argumente haben.

```

father(matthias, peter).
father(emma, peter).
name(matthias, "Matthias").
name(peter, "Peter").

```

Hier wird festgelegt, dass Peter der Vater von Matthias und Emma ist, dass Matthias den Namen "Matthias" und Peter den Namen "Peter" trägt. Diese Art der Prädikat-Ausdrücke werden Fakten genannt.[10]

### 4.1.3 Anfragen

Wenn ein Prädikat-Ausdruck eine oder mehrere Variablen enthält, dann wird dieser Ausdruck als Anfrage-Literal bezeichnet.

father(matthias,X)

Anfrage-Literale werden als wahr angesehen, wenn sich die Variable durch Konstanten oder Werte so ersetzen lässt, dass sich ein Tupel ergibt, das mit den Bedingungen der Anfrage übereinstimmt. Dieser Prozess wird Variablen-Bindung genannt. Um eine Anfrage darzustellen, wird die folgende Notation verwendet:

?- father(matthias,X)

Diese Anfrage bedeutet: "Suche alle Väter von Matthias". Unter Verwendung der obigen Fakten würde die Anfrage wahr ergeben, wenn die Variable *X* an *Peter* gebunden ist. Im Gegensatz zu Prolog liefert eine Anfrage in Datalog alle möglichen Variablenbindungen zurück und nicht nur die erste mögliche.

Eine Liste von Anfrage-Literalen, die durch Kommata getrennt ist, wird als Konjunktion aufgefasst.

?- name(Person, "Peter"), father(Child, Person)

Diese Anfrage findet alle Personen, deren Name Peter ist *und* die Väter sind.[10]

### 4.1.4 Regeln und datenfreie Prädikate

In Datalog ist es möglich, Prädikate zu erzeugen, die nicht Teil der Daten, aber von ihnen abgeleitet sind. Dies geschieht, indem für diese Prädikate Regeln erzeugt werden, die ihre Validität prüfen.

$$\overbrace{\text{sibling}(X, Y)}^{\text{Regelkopf}} \quad :- \quad \overbrace{\text{father}(X, Z), \text{father}(Y, Z)}^{\text{Regelinhalt}}.$$

Diese Regel besagt, dass *X* und *Y* Geschwister sind, wenn sie den gleichen Vater haben. Der linke Teil dieser Regel wird Regelkopf genannt, der rechte Regelinhalt.

Die Anfrage:

?-sibling(X, Y)

würde als Ergebnis Matthias und Emma zurückgeben. Allerdings wurde bei der Formulierung der Regel übersehen, dass es sich bei Personen ebenfalls um Geschwister handelt, wenn sie die gleiche Mutter haben. Daher muss noch eine weitere Regel ergänzt werden, die diesen Umstand berücksichtigt.

$$\begin{aligned} \text{sibling}(X, Y) & :- \text{father}(X, Z), \text{father}(Y, Z). \\ \text{sibling}(X, Y) & :- \text{mother}(X, Z), \text{mother}(Y, Z). \end{aligned}$$

Regeln desselben Prädikats verhalten sich zueinander als wären sie durch ein logisches ODER verbunden. Bei zwei Personen handelt es sich also um Geschwister, wenn sie den gleichen Vater oder die gleiche Mutter haben.

### 4.1.5 Rekursionen

Es ist erlaubt, dass sich Prädikate rekursiv selbst aufrufen. So könnten Vorväter durch folgende Regeln definiert werden:

```
forefather(X, Y) :- father(X, Y).
forefather(X, Y) :- forefather(X, Z), father(Z, Y).
```

Auf diese Weise würde die Anfrage

```
?- forefather(matthias, Y)
```

sämtliche Vorväter von Matthias finden.

### 4.1.6 Auswertung

Anfragen können mit einem Satz von Regeln kombiniert werden, um ihre Ausdrucksfähigkeit zu erhöhen. Eine solche Anfrage wird als Prozedur bezeichnet.

Um einen besseren Überblick über die Ausdrucksfähigkeit von Datalog-Prozeduren und den Verlauf der Auswertung zu bekommen, soll das folgende Beispiel untersucht werden:

```
h2 ('Artificial Intelligence') :- s(Y,title , 'Artificial Intelligence').
h2 (z) :- s(Y, title, z).
h1 (Y) :- s(X, type, Z), h2 (Y).
?- h1(X)
```

Dies ist eine Datalog-Prozedur mit verschachtelten Regeln. Sie besteht aus einer Menge von Regeln und der eigentlichen Anfrage.

Bei der Ausführung der Anfrage beginnt Datalog mit dem ersten Anfrage-Literal der Anfrage. Dieses sagt aus, dass nach allen Regeln gesucht werden soll, die mit  $h1(X)$  benannt sind. Daraufhin findet Datalog  $h1(Y)$  als einzige Regel, deren Kopf mit dem Anfrage-Literal  $h1(X)$  übereinstimmt. Jedes Auftauchen der Variable  $Y$  innerhalb der Regel wird durch die Variable  $X$  ersetzt. Dieser Prozess wird **Unifikation** genannt.

Als nächstes versucht Datalog, die Regel zu verifizieren, indem es alle Anfrage-Literale aus der Regel verarbeitet. Das erste ist ein Anfrage-Literal, das auf keinen weiteren Regelkopf zutrifft. Das zweite Literal stimmt allerdings mit zwei Regelköpfen überein. Zum einen  $h2('Artificial Intelligence')$  und zum anderen  $h2(Z)$ . Daher wird  $h2(Y)$  in der Regel  $h1(Y)$  durch die Disjunktion der Regelinhalte der beiden  $h2$  Regeln unter Anwendung der Unifikation ersetzt. Dieser Prozess wird **Substitution** genannt.

Am obigen Beispiel konnte man erkennen, wie Datalog arbeitet: Es verarbeitet die Anfrage-Literale eines nach dem anderen. Jedesmal, wenn übereinstimmende Regelköpfe gefunden werden, werden diese verarbeitet. Während Datalog dies tut, folgt es dem Konzept von Substitution und Unifikation.

Eine BNF von Datalog befindet sich im Anhang [A.1](#)

## 4.2 EQM

Das EQM liegt der Kommunikation zwischen einem Edutella-Consumer und einem Edutella-Provider zugrunde. In ihm werden die Anfragen formuliert. Eine solche Anfrage besteht sowohl aus der Anfrage selbst wie auch aus der Antwort, die der Edutella-Provider zurückgibt. Das folgende Bild stellt das Edutella Query Model als UML-Diagramm dar.

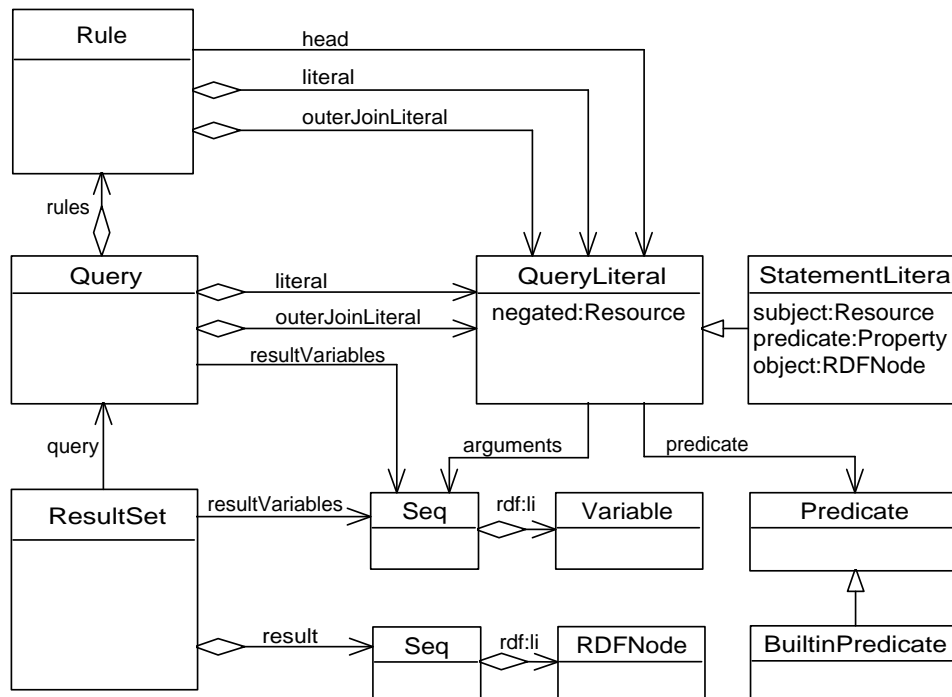


Abbildung 2: UML-Diagramm des Edutella-Query Modells

### 4.2.1 Query

Der Aufbau der Edutella-Anfrage ist analog zum Aufbau der Anfragen in Datalog. Sie besteht aus einem Satz von Regeln, den Rules, und aus den Anfrage-Literalen gebildet aus QueryLiterals.

**Variable** Variablen können anonyme oder nicht-anonyme Ressourcen sein. Dies ist ein Unterschied zu Datalog, wo Variablen immer durch eine Zeichenkette bezeichnet werden müssen.

**Predicate** Prädikate können eine beliebige Anzahl von Eigenschaften haben, die allerdings in der Auswertung der Anfrage nicht berücksichtigt werden.

**OuterJoin** Manchmal ist es nützlich, Informationen, sofern sie vorhanden sind, mit dem Ergebnis auszugeben. Die Ausgabe der Informationen ist aber nicht erforderlich. OuterJoin ist

eine Lösung, die aus dem Bereich der SQL-Sprachen übernommen wurde. In SQL werden teilweise auf diese Weise teilweise überlappende Tabellen zusammengeführt. Fehlende Einträge werden mit NULL aufgefüllt. In Datalog bedeutet diese Konstruktion, dass ein Prädikat *wahr* ist, selbst wenn es keine passenden Daten in der Datenbasis gibt.

**QueryLiteral (Anfrage-Literale)** Anfrage-Literale, die nicht durch ein *s*-Prädikat ausgezeichnet sind, sind vom Typ QueryLiteral. Sie besitzen drei Arten von Eigenschaften:

qel:predicate	zeigt auf eine Instanz von qel:predicate. Manche sind eingebaute Prädikate, die zur Zeit im Wesentlichen Vergleichsoperationen sind; es sind allerdings auch andere Prädikate wie <i>isInteger</i> denkbar.
qel:arguments	beinhaltet eine Liste von Argumenten, wobei ein Argument entweder eine Variable, ein Literal oder eine Ressource sein kann.
qel:negated	gibt an, ob das QueryLiteral negiert aufgefasst werden soll oder nicht.

Anfrage-Literale, die durch das *s*-Prädikat ausgezeichnet sind, sind vom Typ Statement-Literal. Diese entsprechen den klassischen Aussagen und haben die drei Eigenschaften:

rdf:subject	ist ein RDF-Tripel Subjekt
rdf:predicate	ist ein RDF-Tripel Prädikat
rdf:object	ist ein RDF-Tripel Objekt

**Rule** Eine Regel besteht aus drei Teilen:

qel:head	zeigt auf ein QueryLiteral, das den Kopf der Regel darstellt.
qel:literal	besteht aus Instanzen von Query- und Statement-Literalen, die die Nicht-OuterJoinLiterale des Regelinhaltes repräsentieren.
qel:outerjoinLiteral	besteht aus Instanzen von Query- und Statement-Literalen, die die OuterJoinLiterale des Regelinhaltes repräsentieren.

**Query** Um Anfrage-Ergebnisse mit einer Anfrage in Verbindung bringen zu können, dürfen diese nicht anonym sein. Sie besitzen eine eindeutige URI. Eine Query besteht aus vier Teilen:

qel:rule	besteht aus den Regeln der Anfrage.
qel:literal	besteht aus Instanzen aus Query- und StatementLiterals, die die nicht OuterJoinLiterals des Regelinhalt des repräsentieren.
qel:outerJoinLiteral	besteht aus Instanzen aus Query- und StatementLiterals, die die OuterJoinLiterals des Regelinhalt des repräsentieren.
qel:resultVariables	eine Liste von Variablen, die in das Ergebnis aufgenommen werden sollen.

**Result** Das Ergebnis einer Anfrage wird tabellarisch in einem ResultSet zurückgegeben. Ein solches ResultSet kann aus drei Arten von Eigenschaften bestehen:

qel:query	zeigt auf eine Instanz einer Query, auf die sich das Ergebnis bezieht.
qel:resultVariables	zeigt auf eine Liste von Variablen, die in der zum Ergebnis gehörenden Anfrage, verwendet wird.
qel:result	beinhaltet die Variablenbindungen, die das Ergebnis der Anfrage darstellen. Diese Variablenbindungen liegen in der gleichen Reihenfolge vor, wie in der Variablenliste.

### 4.3 Systematisierung verschiedener Syntax-Level

In einem heterogenen System wie dem Edutella Netzwerk stellt sich bei der Informationssuche die Frage, wie Anfragen zwischen den einzelnen Peers ausgetauscht werden sollen. Da in einem heterogenen Netzwerk nicht davon ausgegangen werden kann, dass alle Peers die gleiche Anfragesprache verwenden, bedarf es einer standardisierten Anfrage-Austausch-Sprache, die als Vermittler zwischen den einzelnen Peers fungiert.

Darüber hinaus führen Systembedingte Unterschiede der Peers dazu, dass nicht jeder Peer in der Lage ist, den gleichen Sprachumfang für Anfragen zur Verfügung zu stellen. So können Peers, denen eine Datenbank zugrunde liegt komplexere Anfragen beantworten, als zum Beispiel Peers, die Stichwörter aus einer Datei verwenden.

Damit ein Peer entscheiden kann, ob er in der Lage ist eine Anfrage zu beantworten, muss diese Anfrage innerhalb der Anfrage-Austausch-Sprache in geeigneter Form charakterisiert werden.

Im Folgenden sollen nun zunächst die verschiedenen Kriterien für Anforderungen an eine Anfrage-Austausch-Sprache vorgestellt werden und anschließend ein System, das es ermöglicht, eine Anfrage in standardisierter Form einzuordnen.

### 4.3.1 Kriterien zur Systematisierung

Für die Charakterisierung von Anfrage-Austausch-Sprachen haben sich die folgenden Kriterien als nützlich erwiesen:

Standard Semantik und fehlerfreie RDF-Serialisierung	Einfache und standardisierte Semantik einer Anfrage-Austausch-Sprache ist wichtig, da Übersetzungen von und in diese Sprache innerhalb eines Edutella-Peers die Semantik der originalen Anfrage beibehalten müssen. Darüber hinaus muss eine fehlerfreie Darstellung von Anfragen in RDF möglich sein, um diese zwischen den Peers zu transportieren und innerhalb der Peers zu speichern.
Anpassungsfähigkeit (an unterschiedliche Formalismen)	Eine Anfragesprache muss neutral bezüglich unterschiedlicher semantischer Repräsentationen sein. Sie muss in der Lage sein, ein Prädikate mit vordefinierter Semantik (wie <i>rdfs:subclassOf</i> ) zu verwenden, ohne eine eingebaute Semantik zu besitzen.
Ausdrucksfähigkeit	Die Ausdrucksfähigkeit gibt darüber Auskunft, inwieweit Anfragen verstanden werden können (einfache graphbasierte Anfragen, SQL-Anfrage oder vielleicht sogar Anfragen an Inferenzmaschinen). Es ist also wichtig, dass die Austauschsprache sowohl in der Lage ist, einfache Anfragen zu formulieren, die von einfachen Anbietern direkt verwendet werden können, sowie einem fortschrittlichen Peer die Möglichkeit bietet, seine gesamten Fähigkeiten einzusetzen.
Umformbarkeit	Das zugrunde liegende Modell für eine Anfrage-Austausch-Sprache muss auf einfache Weise in viele verschiedene Anfragesprachen sowohl importiert als auch exportiert werden.

### 4.3.2 RDF-QEL

Um die zuvor dargestellten Kriterien zu handhaben, wurden verschiedene Stufen von Austausch-Sprachen definiert. Derzeit sind fünf Sprachstufen definiert. Die einfachste Anfrage ist eine *Rule-less Query*; sie erlaubt reine konjunktive Anfragen und kann als einfacher RDF-Graph dargestellt werden, wobei die komplexeren Anfragen mehr ausdrücken können, als es RDF vermag und deshalb in vergegenständlichten (reified) RDF-Aussagen dargestellt werden müssen (wie z.B. disjunktive Anfragen, die das relationale Kalkül beinhaltet oder linear-rekursive Anfragen nicht kooperatives Datalog darstellt). Alle Sprachen können jedoch durch das ECDM dargestellt werden.[8][7]

### 4.3.3 Regelfreie Anfrage (Rule-less Query)

Regelfreie Anfragen werden motiviert durch Einfachheit und Lesbarkeit. Anfragen werden durch einfache RDF-Graphen repräsentiert, die exakt die gleiche Struktur haben wie der Antwortgraph. Jede Anfrage in Form eines RDF-Graphen kann als Konjunktion interpretiert werden, die gegen die Wissensbasis geprüft wird. Bezugnehmend auf das Datalog-Modell korrespondieren regelfreie Anfragen mit Datalog-Anfragen, die nur aus der Konjunktion von Anfrage-Literalen ohne Regeln bestehen. [8][7]

### 4.3.4 Konjunktive Anfragen (Conjunctive Query)

Bei diesem Sprachlevel werden die regelfreien Anfragen, die in sich bereits konjunktiv sind, um Regeln erweitert. Dabei ist die Ausdrucksfähigkeit insoweit eingeschränkt, dass nur eine Regel pro Predikat erlaubt ist. Auf diese Weise können auch weiterhin nur Konjunktionen dargestellt werden.

### 4.3.5 Disjunktive Anfragen (Disjunctive Query)

Auf diesem Anfragelevel werden mehrere Regeln pro Prädikat zugelassen, aber keine Rekursionen. Diese Sprache enthält nicht länger reine Aussagen und kann nicht mehr direkt in RDF ausgedrückt werden ohne über die logische Kombination von RDF-Tripel zu sprechen. Zu diesem Zweck wird ein RDF Konstrukt mit Namen *reification* verwendet. Das "Vergegenständlichen" einer RDF Aussage bezieht die Erstellung eines Modells der RDF-Tripel in Form von RDF-Ressourcen vom Typ *Statement* mit ein. Diese Ressource hat als Eigenschaften *rdf:subject*, *rdf:predicate*, *rdf:object* der modellierten RDF-Tripel. Diese *reified statements* sind die Bausteine für jede Anfrage und werden mit Hilfe eines UND-ODER-Baumes verbunden. [8][7]

### 4.3.6 Linear-Rekursive Anfragen (Linear Recursive Query)

Bei diesem Anfragetyp sind Rekursionen erlaubt, die allerdings linear sein müssen. Auf diese Weise können Definitionen für transitive Abschlüsse und lineare rekursive Anfragen ausgedrückt werden und die Anfragen sind kompatibel zu den Möglichkeiten von SQL3.[7]

### 4.3.7 Generelle rekursive Anfragen (General Recursive Query)

Die Rekursion dieser Anfragen muss nicht mehr linear sein. Sie erfordern daher ein Äquivalent zu Datalog oder Prolog Prozessoren für die Ausführung.[7]

## 5 RDQL

### 5.1 Allgemeines

RDQL ist eine datenorientierte Anfragesprache für RDF in Jena Modellen. Datenorientiert, weil sie sich lediglich an den vorhandenen Daten des Modells orientiert und keine Inferenzmechanismen verwendet.

RDQL ist eine Implementation der [myhrefhttp://swordfish.rdfweb.org/rdfquery/](http://swordfish.rdfweb.org/rdfquery/) SquishQLRDF-Anfragesprache, welche selber von [rdfDB](#) abgeleitet ist. Diese Klasse von Anfragesprachen betrachtet RDF als Triple-Daten, ohne dabei Schemata und Ontologieinformationen zu verwenden, außer sie sind explizit in der RDF-Quelle angegeben.

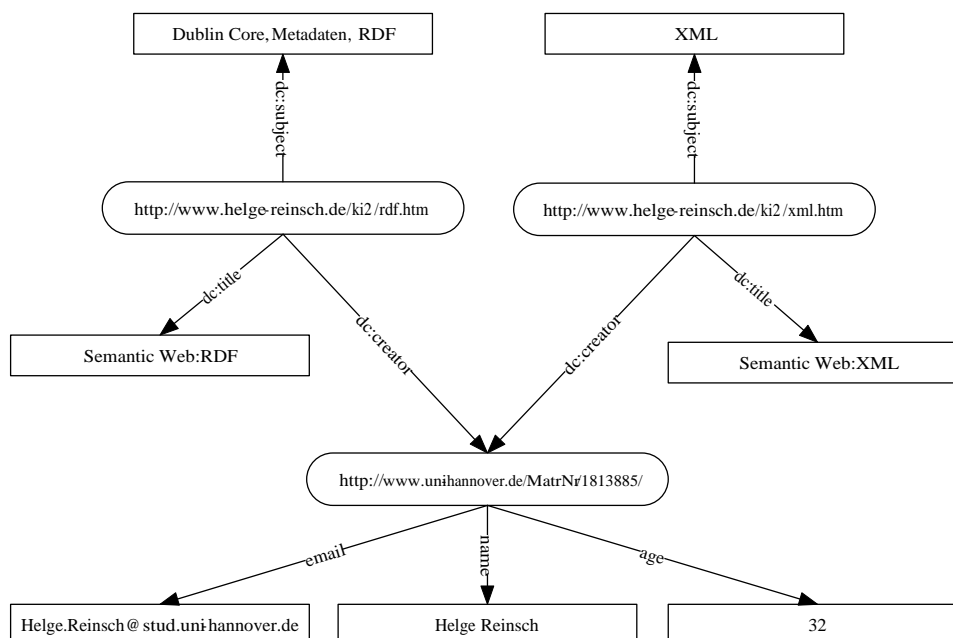
RDF beinhaltet einen gerichteten Graphen, dessen Knoten Ressourcen oder Literale sind, und RDQL bietet die Möglichkeit, einen Mustergraphen zu definieren, der mit dem RDF-Graphen verglichen wird und eine Menge von übereinstimmenden Teilgraphen zurückliefert.

### 5.2 Sprachbeschreibung

RDQL ist eine SQL-ähnliche Anfragesprache. In SQL ist die Datenbank eine abgeschlossene Welt. Die FROM-Klausel identifiziert die Tabellen innerhalb der Datenbank. Die WHERE-Klausel identifiziert die Bedingungen und kann mit AND erweitert werden. In Analogie dazu ist in RDQL das Netzwerk die Datenbank und die FROM-Klausel identifiziert eines oder mehrere RDF-Modelle. Eine Anfrage besteht aus den Elementen:

- **SELECT** gibt an, welche Anfragevariablen im Anfrageergebnis ausgegeben werden sollen.
- **FROM** spezifiziert die zu untersuchenden Modelle anhand von URIs.
- **WHERE** wird gefolgt von einer Liste konjunktiv verbundener Triple-Muster. Diese Muster werden mit allen Tripeln, die im Modell enthalten sind, verglichen und übereinstimmende Tripel in einer Ergebnismenge gesammelt.
- **AND** gibt Bedingungen für den Gültigkeitsbereich von Variablen an.
- **USING** ermöglicht zur Verbesserung der Übersichtlichkeit der Anfrage die Einführung von Platzhaltern.[11]

Der Aufbau von RDQL-Anfragen soll anhand eines Beispiels verdeutlicht werden. Zu diesem Zweck soll eine Ressource, die HTML-Seite "<http://www.Helge-Reinsch.de/ki2/rdf.htm>", beschrieben werden. Die HTML-Seite hat einen Autor, einen Titel und einen Betreff. Der Autor wiederum hat einen Namen und eine E-Mail-Adresse, die unter der Ressource "<http://www.uni-hannover.de/MatrNr/1813885/>" abgelegt sind.



```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s   = "http://description.org/schema/"
  xmlns:dc  = "http://purl.org/metadata/dublin_core#">

  <rdf:description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
    <dc:creator rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/" />
    <dc:title>Semantic Web: RDF</dc:title>
    <dc:subject>Dublin Core, Metadaten, RDF</dc:subject>
  </rdf:description>

  <rdf:description rdf:about="http://www.helge-reinsch.de/ki2/xml.htm">
    <dc:creator rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/" />
    <dc:title>Semantic Web: XML</dc:title>
    <dc:subject>XML</dc:subject>
  </rdf:description>

  <rdf:Description rdf:about="http://www.uni-hannover.de/MatrNr/1813885/">
    <s:name>Helge Reinsch</s:name>
    <s:email>Helge.Reinsch@stud.uni-hannover.de</s:email>
    <s:age>32</s:age>
  </rdf:Description>
</rdf:RDF>
```

Eine einfache Fragestellung, die auf ein solches Modell angewendet werden kann, wäre die Frage nach dem Titel der Seite. Eine solche Anfrage hätte die Form:

```
SELECT
  ?x
WHERE
  ( <http://www.helge-reinsch.de/ki2/rdf.htm>,
    <http://purl.org/metadata/dublin_core#title>,
    ?x )
```

Das `?x` nach dem `SELECT` ist eine Variable, deren Belegung in der Ergebnismenge von Interesse ist. Es werden nur Variablen in die Ergebnismenge aufgenommen, die explizit im `SELECT`-Teil der Anfrage angegeben werden. Die Queryengine sucht sämtliche Belegungen der Variablen, die konform zum Mustergraphen in der `WHERE`-Klausel sind, und fügt diese Belegungen anschließend der Ergebnismenge hinzu.

Hier sieht der Mustergraph in der `WHERE`-Klausel so aus, dass nur die Daten aus dem Modell ausgelesen werden, die sich auf die Ressource "`http://www.helge-reinsch.de/ki2/rdf.htm`" beziehen und zu dieser in der Relation "`http://purl.org/metadata/dublin_core#title`" stehen.

Das Ergebnis dieser Anfrage wäre dann eine Belegung von `?x` mit "Semantic Web: RDF".

```
Result 1
x: Semantic Web: RDF
```

Eine etwas erweiterte Fragestellung ist nach allen Ressourcen und ihrer Titel.

```
SELECT
  ?resource, ?title
WHERE
  (?resource, <http://purl.org/metadata/dublin_core#title>, ?title )
```

Als Ergebnis erscheinen die beiden HTML-Seiten und ihr Titel:

```
Result 1
resource: http://www.Helge-Reinsch.de/ki2/rdf.htm
title: Semantic Web: RDF

Result 2
resource: http://www.helge-reinsch.de/ki2/xml.htm
title: Semantic Web: XML
```

Als *Creator* ist im Beispiel eine Ressource angegeben. Die Frage nach dem Namen und der E-Mail-Adresse des Autors muss also durch einen erweiterten Graphen erfolgen. Der erweiterte Graph beinhaltet eine Variable, die die Ressource, die den *Creator* beschreibt, beinhaltet. Da diese Variable für das Ergebnis nicht von Interesse ist, wird sie nicht in die `SELECT`-Klausel aufgenommen, sondern bleibt anonym.

Anfrage:

```
SELECT
    ?resource, ?name, ?email
WHERE
    ( ?resource, <http://purl.org/metadata/dublin_core#creator>, ?anonym ),
    ( ?anonym, <http://www.uni-hannover.de/MatrNr/1813885/email>, ?email ),
    ( ?anonym, <http://www.uni-hannover.de/MatrNr/1813885/name>, ?name )
```

Ergebnis:

```
Result 1
resource: http://www.Helge-Reinsch.de/ki2/rdf.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

```
Result 2
resource: http://www.helge-reinsch.de/ki2/xml.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

Was an diesem Beispiel leicht zu erkennen ist, ist die Tatsache, dass bei längeren Anfragen die Notierung der verwendeten Schemata eine gewisse Zeit in Anspruch nehmen kann. Zur Erleichterung der Schreibweise und Verbesserung der Übersichtlichkeit der Anfragen besteht bei RDQL die Möglichkeit, Kurzschreibweisen zu definieren. Diese werden dann beim Parsen der Anfrage durch die vollständige Version ersetzt. Im folgenden Beispiel wurde die vorherige Anfrage so verändert, dass für das Dublin Core Schema die Kurzform *dc* und für das Schema der Universität die Abkürzung *descr* verwendet wurden.

```
SELECT
    ?resource, ?name, ?email
WHERE
    ( ?resource, <dc:creator>, ?anonym ),
    ( ?anonym, <descr:name>, ?name ),
    ( ?anonym, <descr:email>, ?email )
USING
    dc    FOR <http://purl.org/metadata/dublin_core#>,
    descr FOR <http://www.uni-hannover.de/MatrNr/1813885/>
```

Zuletzt soll noch die Verwendung von Bedingungen demonstriert werden. Zu diesem Zweck wurde die Anfrage noch um zwei Punkte erweitert. Zum einen soll der Titel der Ressource mit ausgelesen und zum anderen das Ergebnis auf Ressourcen von Autoren eines bestimmten Mindestalters eingeschränkt werden.

Anfrage:

```

SELECT
    ?resource, ?name, ?email
WHERE
    ( ?resource, <dc:creator>, ?anonym),
    ( ?resource, <dc:title>, ?title ),

    ( ?anonym, <descr:name>, ?name ),
    ( ?anonym, <descr:email>, ?email),
    ( ?anonym, <descr:age>, ?age )
AND
    ?title eq "Semantic Web: XML",
    ?age >= 25
USING
    dc    FOR <http://purl.org/metadata/dublin_core#>,
    descr FOR <http://www.uni-hannover.de/MatrNr/1813885/name>

```

Ergebnis:

```

Result 1
resource: http://www.helge-reinsch.de/ki2/xml.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de

```

Trennung der Bedingungen durch ein Komma, bedeutet Konjunktion. Es besteht aber auch die Möglichkeit, logische Ausdrücke zu formulieren. Soll z.B. nach Dokumenten gesucht werden, deren Titel entweder "Semantic Web: XML" *oder* das Alter der Autors  $\geq 25$  ist, kann die Anfrage wie folgt abgewandelt werden:

Anfrage:

```

SELECT
    ?resource, ?name, ?email
WHERE
    ( ?resource, <dc:creator>, ?anonym),
    ( ?resource, <dc:title>, ?title ),

    ( ?anonym, <descr:name>, ?name ),
    ( ?anonym, <descr:email>, ?email),
    ( ?anonym, <descr:age>, ?age )
AND
    ?title eq "Semantic Web: XML"
    || ?age >= 25
USING
    dc    FOR <http://purl.org/metadata/dublin_core#>,
    descr FOR <http://www.uni-hannover.de/MatrNr/1813885/name>

```

Ergebnis:

```
Result 1
resource: http://www.Helge-Reinsch.de/ki2/rdf.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

```
Result 2
resource: http://www.helge-reinsch.de/ki2/xml.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

Zu Formulierung der logischen Ausdrücke kann auch Klammerung verwendet werden.

Anfrage:

```
SELECT
    ?resource, ?name, ?email
WHERE
    ( ?resource, <dc:creator>, ?anonym),
    ( ?resource, <dc:title>, ?title ),

    ( ?anonym, <descr:name>, ?name ),
    ( ?anonym, <descr:email>, ?email),
    ( ?anonym, <descr:age>, ?age )
AND
    ( ?title eq "Semantic Web: XML"
      || ?title eq "Semantic Web: RDF" )

    && ( ?age >= 25 )
USING
    dc    FOR <http://purl.org/metadata/dublin_core#>,
    descr FOR <http://www.uni-hannover.de/MatrNr/1813885/name>
```

Ergebnis:

```
Result 1
resource: http://www.Helge-Reinsch.de/ki2/rdf.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

```
Result 2
resource: http://www.helge-reinsch.de/ki2/xml.htm
name: Helge Reinsch
email: Helge.Reinsch@stud.uni-hannover.de
```

### 5.3 Übersetzung der Anfragen aus dem EQM in RDQL

Um RDQL im Zusammenhang mit Edutella nutzen zu können, ist es notwendig, das Edutella Query Model in RDQL zu übersetzen.

Der Aufbau der Anfragen im EQM entspricht dem der Anfragen in Datalog. Im Folgenden sollen die Übersetzung von Edutella-Queries der verschiedenen Level in RDQL beschrieben werden.

### 5.3.1 Rule-Less Queries

Eine Rule-Less Query besteht lediglich aus Anfrage-Literalen und beinhalten keine Regeln. In der Datalog-Notation haben Anfragen dieser Art die Form:

```
?- s(X,Y,Z), n(A,B),...
```

Die Übersetzung der Anfragen besteht aus drei Teilen:

1. Extraktion aller Variablen aus der Anfrage, um diese in die SELECT-Klausel zu übernehmen.
2. Die Statement-Literale müssen in die WHERE-Klausel übernommen werden.
3. Die Bedingungen müssen in der AND-Klausel zusammengefasst werden.

Jedes Anfrage-Literal wird untersucht. Handelt es sich um ein Statement-Literal, so werden subject, predicate und object ausgelesen. Alle drei Bestandteile werden untersucht, ob sie Variablen sind. Wenn dies der Fall ist, werden sie in die Variablen-Notation von RDQL ("?Variablenname") übersetzt. Handelt es sich um eine Literal, wird dieses mit doppelten Anführungszeichen umschlossen, und wenn es sich um eine Ressource handelt, wird diese in die Ressource-Darstellung (<URI>) von RDQL übersetzt. Anschließend werden die variablen Anteile des Statement-Literal in der SELECT-Klausel ergänzt und die Statements an die WHERE-Klausel angehängt. Zuletzt werden die gefundenen Bedingungen in der AND-Klausel zusammengefasst. Negationen von Statement-Literalen sind im Moment noch nicht zugelassen, da nicht feststeht, wie eine solche Negation interpretiert werden soll. So könnte eine Negation in der Form (?x, author, "Helge Reinsch") bedeuten, dass nach allen Autoren gesucht werden soll, die nicht "Helge Reinsch" heißen oder nach allen ?x bei denen "Helge Reinsch" nicht der Autor ist.

In einem konkreten Beispiel sähe die Übersetzung wie folgt aus:

```
?- s(resource, http://purl.org/metadata/dublin_core#title, 'Semantic Web: XML')
```

Die Anfrage besteht aus einem Statement-Literal. Sie setzt sich zusammen aus dem variablen Subjekt, dem Prädikat

"http://purl.org/metadata/dublin\_core#title" in Form einer Ressource und aus dem Literal 'Semantic Web: XML' als Objekt.

Bei der Übersetzung wird die Variable in die Variablen-Notation von RDQL übersetzt. Dies führt zu: ?resource. Die Ressource muss in die Ressource-Notation übersetzt werden:

<http://purl.org/metadata/dublin\_core#title>. Zuletzt wird noch das Literal in die Literaldarstellung überführt: "Semantic Web: XML". Anschließend werden die SELECT-Klausel und die WHERE-Klausel gebildet:

```
SELECT
    ?resource
WHERE
    (?resource, <http://purl.org/metadata/dublin_core#title>,
     "Semantic Web: XML")
```

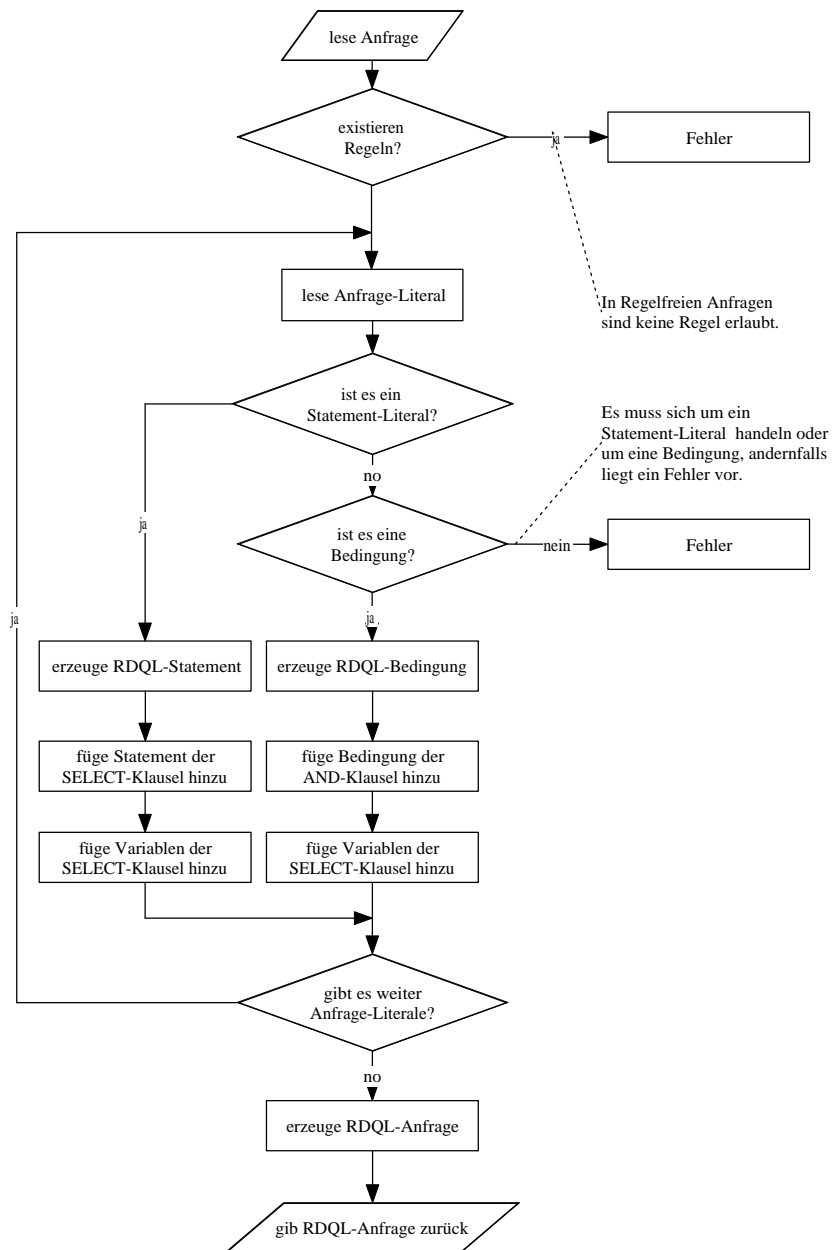
Ähnlich verhält sich die Übersetzung von Bedingungen. Sie bestehen aus zwei Argumenten und einem Operator. Die beiden Argumente werden wie im Fall des Statement-Literal dahingehend überprüft, ob es sich um Variablen, um Ressourcen oder um Literale handelt.

Beispiel:

```
?- http://www.uni-hannover.de/MatrNr/1813885/name/age (person, age), qel:greaterthan(age, 25)
```

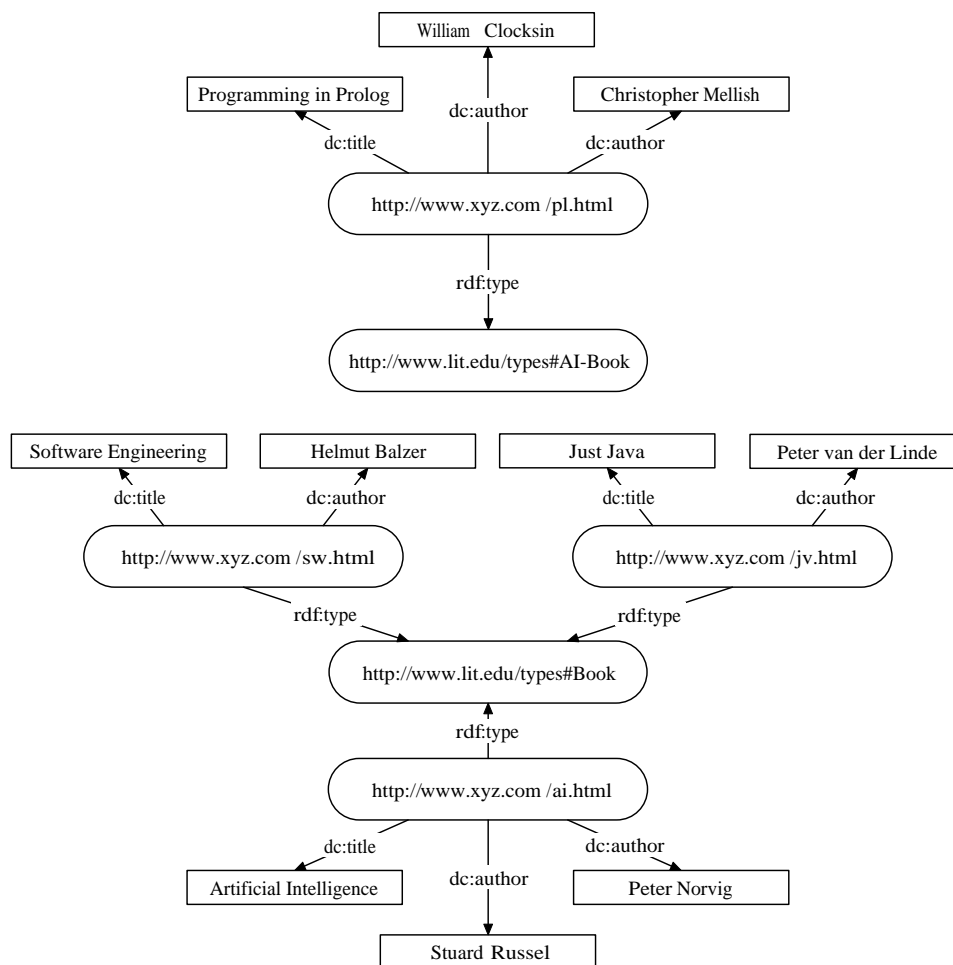
Diese Anfrage sucht nach allen Personen, die älter sind als 25 Jahre. Das erste Anfrage-Literal wird nach dem oben beschriebenen Schema umgewandelt. Die Bedingung wird umgewandelt, indem die beiden Operatoren darauf untersucht werden, ob sie Variable, Ressource oder Literal sind. Abhängig davon werden sie in die jeweilige Darstellung übersetzt und ggf. an die SELECT-Klausel angefügt. Zum Schluss wird noch die AND-Klausel erzeugt, indem die beiden Argumente um den Operator angeordnet werden. Bei mehreren Konditionen werden diese durch Komma getrennt aneinander gereiht. Ebenso verhält es sich mit den Bestandteilen der WHERE-Klausel.

```
SELECT
    ?person, ?age
WHERE
    (?resource, <http://www.uni-hannover.de/MatrNr/1813885/name/age>, ?age)
AND
    (?age >= 25)
```



### 5.3.2 Conjunctive Queries

Um die nächsten beiden Anfragestufen besser veranschaulichen zu können, soll an dieser Stelle ein Beispielgraph angegeben werden, auf den sich die Anfragebeispiele beziehen werden.



In RDF wird dieser Graph wie folgt notiert:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:edu="http://www.edutella.org/edutella#"
  xmlns:lit="http://www.lit.edu/types#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description about="http://www.xyz.com/sw.html">
    <rdf:type rdf:resource="http://www.lit.edu/types#Book"/>
    <dc:title>Software Engineering</dc:title>
    <dc:author>Helmut Balzert</dc:author>
  </rdf:Description>

  <rdf:Description about="http://www.xyz.com/ai.html">
    <rdf:type rdf:resource="http://www.lit.edu/types#Book"/>
```

```

    <dc:title>Artificial Intelligence</dc:title>
    <dc:author>Stuart Russel</dc:author>
    <dc:author>Peter Norvig</dc:author>
  </rdf:Description>

  <rdf:Description about="http://www.xyz.com/pl.html">
    <rdf:type rdf:resource="http://www.lit.edu/types#AI-Book" />
    <dc:title>Programming in Prolog</dc:title>
    <dc:author>William Clocksin</dc:author>
    <dc:author>Christopher Mellish</dc:author>
  </rdf:Description>

  <rdf:Description about="http://www.xyz.com/jv.html">
    <rdf:type rdf:resource="http://www.lit.edu/types#Book" />
    <dc:title>Just Java</dc:title>
    <dc:author>Peter van der Linden</dc:author>
  </rdf:Description>

</rdf:RDF>

```

Bei der Übersetzung von Anfragen, die zu Conjunctive Queries konform sind, treten zum ersten Mal Regeln auf. Als Einschränkung gilt jedoch, dass jeder Regelkopf nur ein einziges Mal auftreten darf. Auf diese Weise ist gewährleistet, dass es sich bei der Anfrage tatsächlich um eine Konjunktion handelt.

Bei der Übersetzung werden die Anfrage-Literale der Reihenfolge nach untersucht. Handelt es sich bei dem gefundenen Anfrage-Literal um ein Statement-Literal oder eine Bedingung, so wird dieses, analog zu Rule-Less Queries, in die RDQL-Anfrage übernommen. Handelt es sich dagegen um ein Regelverweis, muss die entsprechende Regel untersucht werden.

Die Anfrage-Literale im Regelinhalt werden nach demselben Schema wie die Anfrage-Literale in der Anfrage rekursiv untersucht. Dabei werden die entstehenden WHERE- und AND-Klauseln aus der Regel an die WHERE- und AND-Klauseln der Anfrage-Literal angehängt. Dies ist möglich, da es sich um eine konjunktive Anfrage handelt.

Ein Beispiel für ein solche Anfrage wäre:

```

ai (y) :- Type (y, AI-Book).
        ?- ai(x), author(x,'Peter van der Linden')

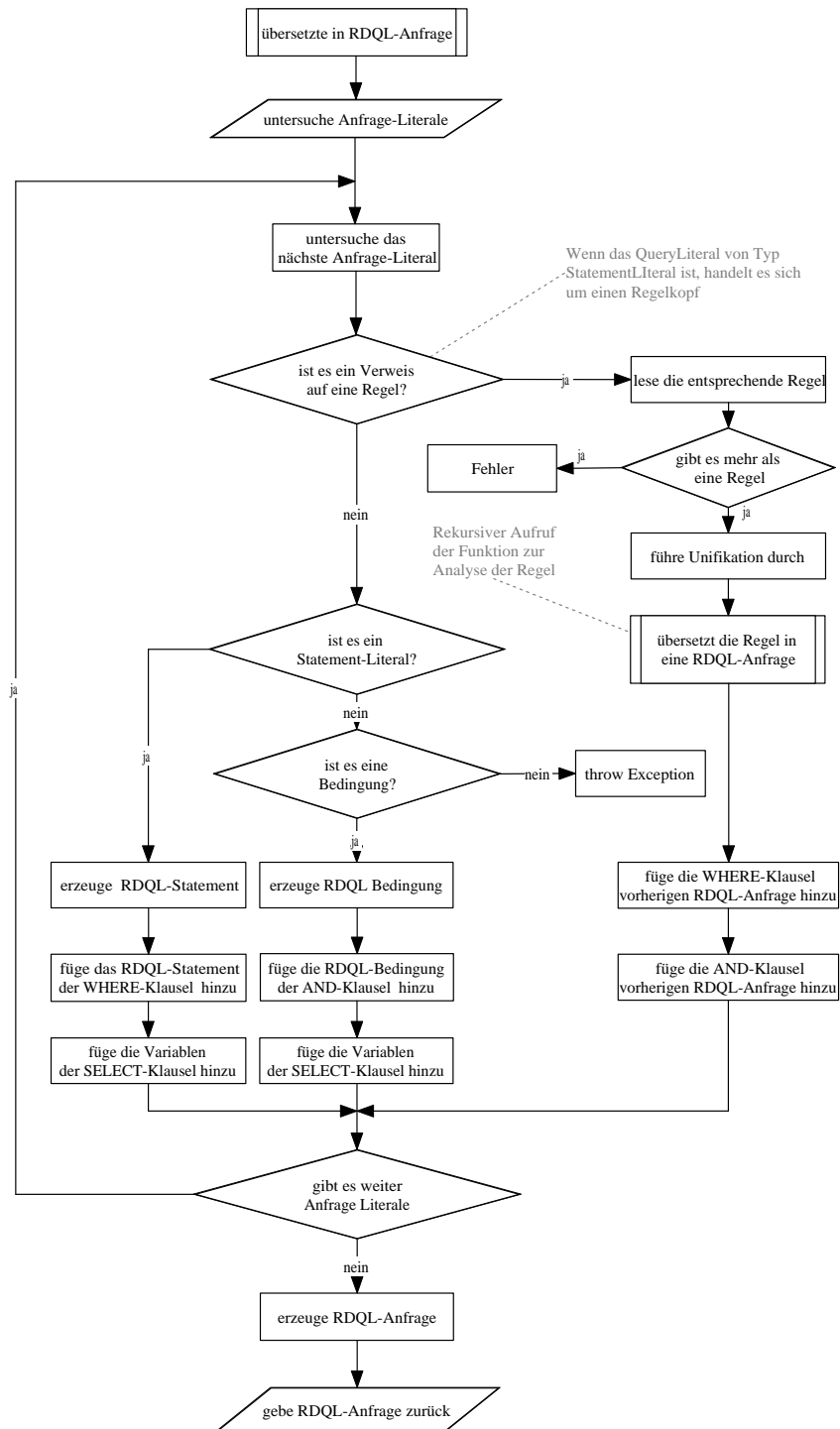
```

Eine Übersetzung dieser Anfrage träge bei der Auswertung auf das Anfrage-Literal ai (x) zu, das auf eine Regel verweist.

Im nächsten Schritt muss geprüft werden, ob es mehrere Regeln zu diesem Regeltyp gibt. Wenn dies der Fall ist, wird die Übersetzung abgebrochen ansonsten wird die Regel unter Verwendung der Unifikation in die richtige Form gebracht und anschließend durch Substitution in die Anfrage übernommen. Das nächste Statement legt fest, dass es sich bei dem Autor um "Peter van der Linden" handeln soll. Da diese Aussage direkt angegeben wurde, wird sie auch direkt bei der Übersetzung übernommen.

Eine Übersetzung käme zu dem Ergebnis:

```
SELECT
  ?x
WHERE
  (?x, <http://www.lit.edu/types#Book>, "AI-Book"),
  (?x, <http://purl.org/dc/elements/1.1/author>, "Peter van der Linden")
```



### 5.3.3 Disjunctive Queries

Bei der Übersetzung von Anfragen, die zu Disjunctive Queries konform sind, ist zu beachten, dass diese im Gegensatz zu Conjunctive Queries nicht nur einen passenden Regelkopf besitzen dürfen, sondern mehrere.

```
ai(x) :- Title(x, "Artificial Intelligence"), Type(x, Book).
?- ai(x), http://purl.org/dc/elements/1.1/author(x, 'Stuart Russel')
```

Dies ist die einfachste Form einer solchen Anfrage. Sie besteht lediglich aus zwei Anfrage-Literalen und einer Regel. Die Regel selbst besteht aus zwei Anfrage-Literalen. Die Anfrage-Literale der Regel sind durch ein logisches UND miteinander verbunden. Das Anfrage-Literal und die Regel in der Anfrage sind ebenfalls durch ein UND verbunden. Um eine spätere Auflösung mehrerer Regeln mit einem ODER zu ermöglichen, muss ein Weg gefunden werden, der es erlaubt, die Aussage der Regel in die AND-Klausel der RDQL-Anfrage zu verlagern, da es nur dort möglich ist, ODER-Verknüpfungen zu verwenden.

Um dies zu erreichen, wird für jedes Anfrage-Literal einer Regel in der WHERE-Klausel ein Platzhalter eingefügt. Die Platzhalter bestehen aus den im Regelkopf festgelegten Variablen sowie aus anonymen Platzhalter-Variablen, die in die Auswertung der Anfrage nicht eingehen. Anschließend werden in der AND-Klausel die anonymen Variablen mit Bedingungen versehen, die die ursprüngliche Aussage wieder herstellen.

Enthält eine Regel eine Bedingung, wird diese mit UND verknüpft an den zu der Regel gehörenden Bedingungsblock in der AND-Klausel angehängt. Die Anfrage-Literale, die in der Anfrage stehen, können wie gewohnt übersetzt werden.

```
SELECT
  ?x
WHERE
  (?x, ?anonym1, ?anonym2), (?x, ?anonym3, ?anonym4),
  (?x, <http://purl.org/dc/elements/1.1/author>, "Stuart Russel")
AND
  ( (?anonym1 eq <dc:title>)
    && (?anonym2 eq "Artificial Intelligence"))
  && ( (?anonym3 eq <rdf:type>)
    && (?anonym4 eq <http://www.lit.edu/types#Book>))
USING
  dc FOR <http://purl.org/dc/elements/1.1/>,
  rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Für die beiden Anfrage-Literale aus der Regel `ai(x)` wurde jeweils ein Statement in die WHERE-Klausel aufgenommen. Innerhalb dieser Statements ist nur die im Kopf der Regel angegebene Variable `?x` erhalten geblieben. Alle anderen Bestandteile wurden durch anonyme Variablen `?anonymi` ersetzt, die anschließend in der AND-Klausel mit entsprechenden Bedingungen belegt werden. Bei der Übersetzung von mehreren gleichartigen Regeln müssen diese durch ein logisches ODER verbunden werden. Ein Beispiel dafür ist:

```
ai(x) :- s(x, Title, "Artificial Intelligence"), s(x, Type, Book).
ai(x) :- s(x, Type, AI-Book).
?- ai(x)
```

Diese Anfrage besteht aus zwei Regeln mit unterschiedlicher Anzahl von Anfrage-Literalen. Bei der Auswertung dieser Anfrage werden die beiden Regeln  $a_i(x)$  wie durch ein logisches ODER verbunden behandelt. Dieses ODER wird durch Verwendung der AND-Klausel verwirklicht.

Zunächst müssen allerdings noch ein paar Vorüberlegungen angestellt werden.

Da bereits Statements in der WHERE-Klausel aus der Übersetzung der ersten Regel vorhanden sind, können keine weiteren Statements hinzugefügt werden, weil das Hinzufügen weiterer Regeln in der WHERE-Klausel logisch UND bedeutet. Es müssen also die bereits vorhandenen Aussagen verwendet werden.

Die Anzahl der vorhandenen Regeln hängt von der Anzahl der Anfrage-Literale in den Regeln ab. Es muss zunächst die Regel mit der maximalen Anzahl von Anfrage-Literalen bestimmt werden, um herauszufinden, wieviele Statements benötigt werden.

Hat eine Regel, so wie in diesem Beispiel, weniger Anfrage-Literale als die Regel mit der maximalen Anzahl, sollten die überzähligen Statements nicht einfach leer gelassen werden. Natürlich wäre dies theoretisch möglich, weil sich dadurch keine zusätzlichen Ergebnisse ergeben, allerdings versucht die Queryengine, Belegungen für die überzähligen Statements zu finden und erzeugt für jede mögliche Belegung einen neuen Ergebniseintrag. Diese Einträge stimmen alle überein und erzeugen somit eine unerwünschte Redundanz.

Um dies zu verhindern, müssen die überzähligen Statements sinnvoll belegt werden. Dies geschieht, indem die Variablen mit Bedingungen aus bereits zuvor verwendeten Anfrage-Literalen nochmalig belegt werden.

Der Eintrag für die AND-Klausel für die erste Regel ist bereits aus dem vorherigen Beispiel bekannt:

```
(      (?anonym1 eq <dc:title>)
  && (?anonym2 eq "Artificial Intelligence"))
&& (      (?anonym3 eq <rdf:type>)
  && (?anonym4 eq <http://www.lit.edu/types#Book>))
```

Ein Übersetzung der zweiten Regel führt zu:

```
(      (?any1 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?any2 eq <http://www.lit.edu/types#AI-Book>))
&& (      (?any3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?any4 eq <http://www.lit.edu/types#AI-Book>))
```

Es lässt sich erkennen, dass die Bedingungen für die anonymen Bestandteile des ersten Statements für die Belegung des zweiten Statements erneut verwendet wurden. Auf diese Weise findet die Queryengine die gleiche Belegung, wie in dem Fall mit nur einem Statement.

Zusammengefasst ergibt sich die Anfrage zu:

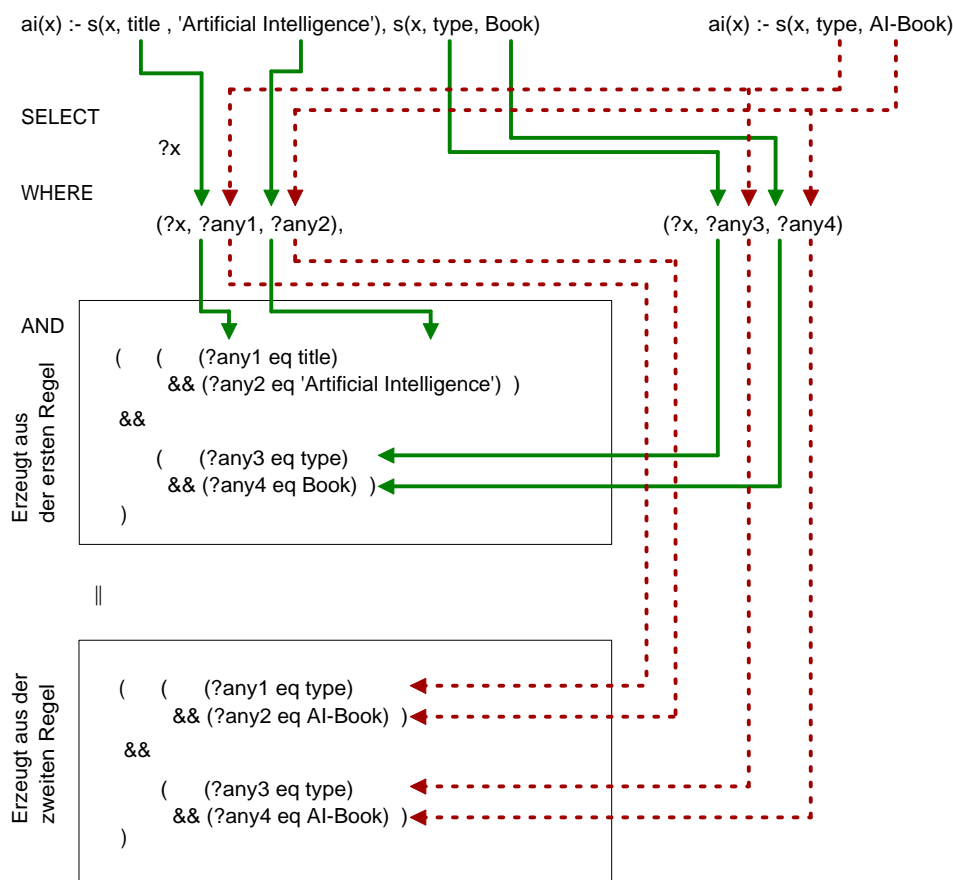
```
SELECT
  ?x
WHERE
  (?x, ?any1, ?any2), (?x, ?any3, ?any4)
AND
```

```

(
  (
    (?any1 eq <http://purl.org/dc/elements/1.1/title>)
    && (?any2 eq "Artificial Intelligence")
  )
  && (
    (?any3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
    && (?any4 eq <http://www.lit.edu/types#Book>)
  )
)
||
(
  (
    (?any1 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
    && (?any2 eq <http://www.lit.edu/types#AI-Book>)
  )
  && (
    (?any3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
    && (?any4 eq <http://www.lit.edu/types#AI-Book>)
  )
)

```

Man erkennt die Anonymisierung der beiden Statements und die mit ODER verknüpfte Belegung der anonymen Variablen.



Im nächsten Schritt sollen verschachtelte Regeln betrachtet werden. Wenn innerhalb einer Regel auf eine andere Regel verwiesen wird, erhöht sich die Anzahl der zu betrachtenden Statements und die Bedingungen werden ineinander verschachtelt.

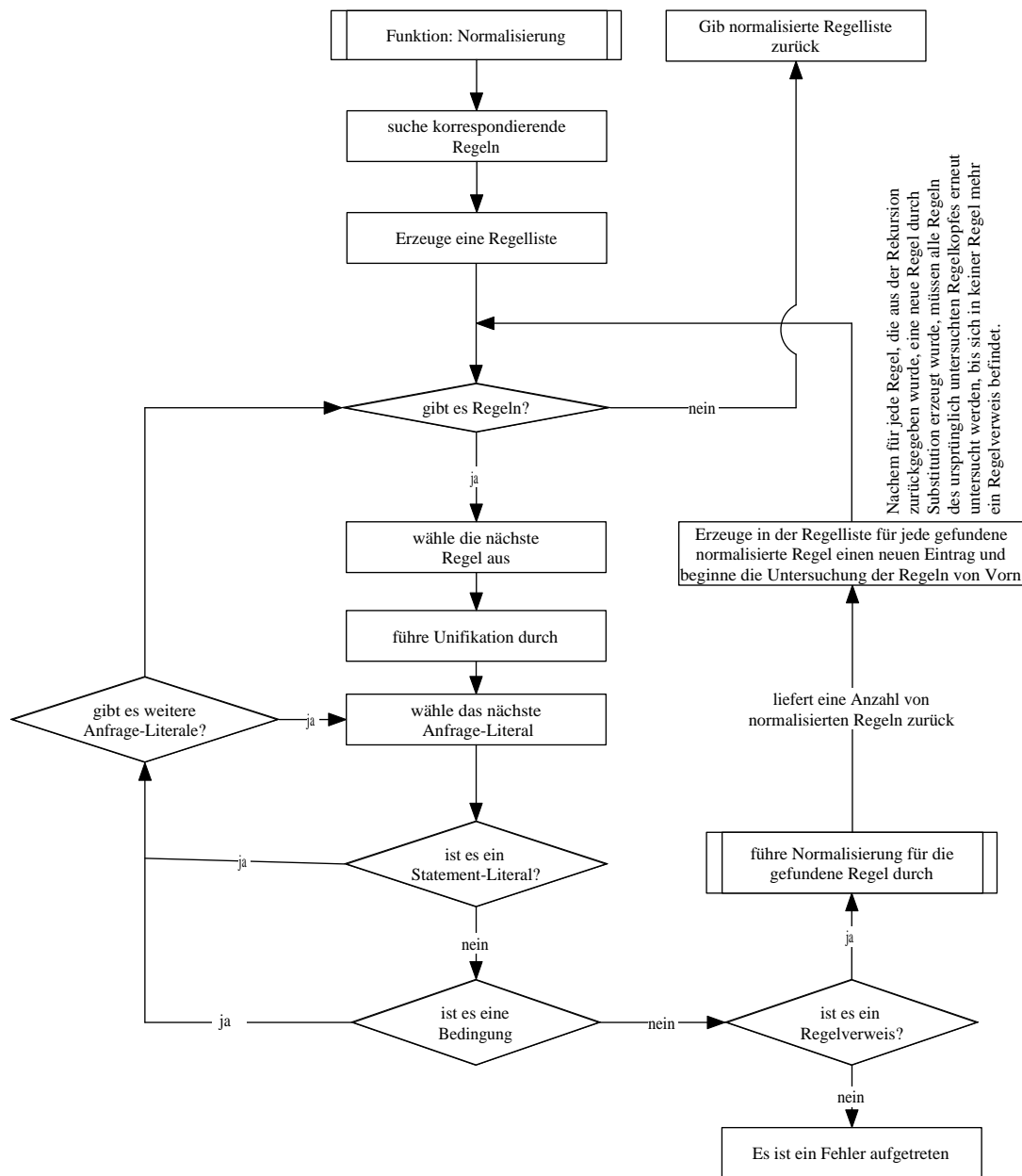
Eine solche Auswertung während der Übersetzung vorzunehmen, ist sehr kompliziert. Es ist daher ratsam, die Regelstruktur in eine angemessenen Form zu bringen, bevor die Anfrage in RDQL übersetzt wird, um die Übersetzung zu vereinfachen.

Die Übersetzung von EQM nach RDQL ist besonders einfach, wenn innerhalb von Regeln keine Verweise auf weitere Regeln auftauchen. Eine solche Darstellung der Regeln soll im Weiteren

als **normalisiert** bezeichnet werden.

Ein Normalisierung erfolgt, indem zunächst zu jedem Anfrage-Literal, das einen Regelverweis darstellt, die dazugehörigen Regeln normalisiert werden. Dazu werden aus dem Regelsatz alle korrespondierenden Regeln herausgefiltert. Diese gefundenen Regeln sollen zur besseren Beschreibung als "Ober"-Regeln bezeichnet werden. Anschließend werden nacheinander die einzelnen "Ober"-Regeln untersucht. Enthält eine "Ober"-Regel einen Verweis auf eine andere Regel, hier als "Unter"-Regel bezeichnet, wird zu jeder existierenden "Unter"-Regel eine neue Instanz der "Ober"-Regel erzeugt. Auf diese Weise werden die ODER-Verknüpfungen der Regeln untereinander aus den Querverweisen in die Vielfachheit der Regeln transformiert. Damit dies gelingt, müssen die "Unter"-Regel rekursiv nach dem gleichen Verfahren ebenfalls normiert werden.

Am Ende dieses Vorganges liegen nur noch querverweisfreie Regeln vor, die auf leichte Weise übersetzt werden können.



Die Übersetzung an einem konkreten Beispiel:

- Regel1:* book (Y) :- Type (Y, Book).  
*Regel2:* russel (Z) :- Author (Z, 'Stuart Russel').  
*Regel3:* aibook (X) :- Type (X, AI-Book).  
*Regel4:* aibook (X) :- book (X), aititle (X).  
*Regel5:* aititle (Y) :- title (Y, 'AI').  
*Regel6:* aititle (Y) :- title (Y, 'Artificial Intelligence').  
 ?- aibook(X), russel(X), Author(X, 'Peter Norvig')

Diese Anfrage besteht aus sechs Regeln. Die erste Regel legt fest, dass *Y* ein *Book* ist, wenn es vom Typ *Book* ist. Die zweite besagt, dass *z* die Regel *russel* erfüllt, wenn '*Stuart Russel*'

der Autor von  $z$  ist. Die anderen beiden Regeln legen fest, dass es sich bei  $X$  um ein `aibook` handelt, wenn entweder  $X$  den Regeln `aititle (Y)` und `book (x)` genügt, oder wenn es vom Typ *AI-Book* ist.

Die Regeln `aititle(x)` besagen, dass  $X$  genau dann dieser Regel genügt, wenn es entweder den Titel "AI" oder den Titel "Artificial Intelligence" besitzt. Bei der Übersetzung werden im ersten Schritt die Statement-Literale und die Bedingungen aus der Anfrage übernommen. Das ist in diesem Fall `Author(X, 'Peter Norvig')`. Dieses Statement kann, wie bei Rule-Less-konformen Anfragen, direkt in die `WHERE`-Klausel übersetzt werden. Anschließend wird mit der Übersetzung der Regeln begonnen.

Dabei wird jeder Regelverweis aus der Anfrage separat betrachtet.

Der erste Regelverweis (in diesem Fall der einzige), der gefunden wird, ist `aibook(X)`.

Es treffen die Regeln 3 und 4 zu. Regel 3 besitzt ein Statement und keinen Verweis auf eine andere Regel. Aus diesem Grund braucht diese Regel nicht normalisiert zu werden.

Die Regel 4 besitzt zwei Querverweise auf andere Regeln. Ein evtl. vorkommendes Statement bliebe von der Normalisierung unberührt, da es eine `UND`-Verknüpfung mit den anderen Statements darstellt. Der Verweis auf eine andere Regel hingegen beschreibt eine `ODER`-Verknüpfung.

In der Normalform werden die `ODER`-Verknüpfungen durch mehrere Regeln dargestellt. Um das bei durch Verweise verknüpften Regeln zu erreichen, muss von der Regel 4 zu jedem vorkommen einer Regel vom Typ `book(Y)` und von Typ `aititle(X)` eine Instanz erzeugt werden.

Die einzige Regel vom Typ `book (Y)` ist Regel 1. Es könnte sein, dass in der Regel 1 ebenfalls Verweise auf andere Regeln auftreten. Diese Verweise müssten, bevor die Auswertung von Regel 4 weitergeführt werden kann, aufgelöst und evtl weitere Instanzen der Regel 1 erzeugt werden.

Da keine weiteren Instanzen der Regel 1 erzeugt werden müssen, kann mit der Untersuchung des Inhaltes der Regel 4 fortgefahren werden.

Die Regel 1 `book (Y)` besitzt ein Statement, das in die Regel 4 eingesetzt werden muss. Bevor dies gemacht werden kann, muss noch die Notation durch Unifikation angeglichen werden. Die Regel 1 lautet damit zunächst

`book (X) :- Type(X, Book)`

Nun kann der Verweis in Regel 4 substituiert werden. Dies führt zu:

`aibook (X) :- Type(X, Book), aititle (X)`

Im nächsten Schritt wird der zweite Verweis – `aititle (X)` – untersucht. Auf diesen Regelkopf treffen die Regeln 5 und 6 zu. Wie zuvor müssen zunächst die einzelnen Regeln untersucht werden. Da in beiden Regeln nur Statements und keine Querverweise auftreten, müssen keine weiteren Instanzen dieser Regeln erzeugt werden.

Anders verhält es sich bei der Regel 4, da hier zwei Regeln auf den Regelkopf `aititle (X)` zutreffen. Die beiden Instanzen der Regel lauten:

```
aibook (X) :- Type(X, Book), title (Y, 'AI')
```

```
aibook (X) :- Type(X, Book), title (Y, 'Artificial Intelligence')
```

Damit ist die Untersuchung des ersten Verweises der Abfrage abgeschlossen, und es wird der nächste Verweis untersucht. Es handelt sich um `russe1(X)`. Die einzige Regel, die mit `russe1(X)` übereinstimmt, ist Regel 2. Sie verweist auf keine andere Regel. Damit könnte sie indirekt als Anfrage-Literal substituiert werden.

Das Regelsystem liegt jetzt in der Normalform vor:

*Regel1:*    `book (Y) :- Type (Y, Book).`

*Regel2:*    `russe1 (Z) :- Author (Z, 'Stuart Russel').`

*Regel3:*    `aibook (X) :- Type (X, AI-Book).`

*Regel4a:*   `aibook (X) :- Title (X, 'AI'), Type (X, Book) .`

*Regel4b:*   `aibook (X) :- Title (X, 'Artificial Intelligence'), Type (X, Book) .`

*Regel5:*    `aititle (Y) :- title (Y, 'AI').`

*Regel6:*    `aititle (Y) :- title (Y, 'Artificial Intelligence').`

`?- aibook(X), russe1(X), Author(X, 'Peter Norvig')`

Da sämtliche Verweise aus den in Frage kommenden Regeln entfernt wurden, kann die maximale Anzahl der Statements bestimmt werden. Dies geschieht pro Verweis innerhalb der Anfrage. Es ergibt sich, dass für `aibook(X)` zwei Statements benötigt werden und für `russe1(X)` ein Statement. Damit ergeben sich 3 Statements mit anonymen Variablen.

```
SELECT
  ?X
WHERE
  (?X, <http://purl.org/dc/elements/1.1/author>, "Peter Norvig"),
  (?X, ?anonym1, ?anonym2), (?x, ?anonym3, ?anonym4), (?x, ?anonym5, ?anonym6)
```

Im dritten Schritt müssen die anonymen Variablen in der `WHERE`-Klausel mit Bedingungen versehen werden.

Es soll mit der Regel 3 `aibook (X) :- Type (X, AI-Book)` begonnen werden. Diese Regel besitzt nur ein Statement.

Daraus ergibt sich als Bedingung

```
( (?anonym1 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?anonym2 eq <http://www.lit.edu/types#AI-Book>))
```

Die weiteren Statements müssen sinnvoll ergänzt werden, um Redundanz zu vermeiden. Dabei ist zu beachten dass nur Statements, die zu `aibook(X)` gehören, für die Ergänzung relevant sind. Der Einfachheit halber werden hier die Belegungen weitergeführt. Damit ergibt sich als Bedingungsblock für die erste Regel:

```
( (?anonym1 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?anonym2 eq <http://www.lit.edu/types#AI-Book>))
&& ( (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?anonym4 eq <http://www.lit.edu/types#AI-Book>))
```

Bei der Regel 4 verhält es sich wie zuvor. Die anonymen Variablen werden mit den entsprechenden Bedingungen versehen. Dabei werden die einzelnen Regeln mit einem logischen ODER verbunden. Das Ergebnis ist:

```
( ( (?anonym1 eq <http://purl.org/dc/elements/1.1/title>)
  && (?anonym2 eq "Artificial Intelligence"))
  && ( (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?anonym4 eq <http://www.lit.edu/types#Book>)))
||
( ( (?anonym1 eq <http://purl.org/dc/elements/1.1/title>)
  && (?anonym2 eq "AI"))
  && ( (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
  && (?anonym4 eq <http://www.lit.edu/types#Book>)))
```

Für `russel(X)` existiert nur die Regel 2, die nur aus einem Statement besteht. Der Bedingungsblock hat damit die Form:

```
( (?anonym5 eq <http://purl.org/dc/elements/1.1/author>)
  && (?anonym6 eq "Stuart Russel"))
```

Nachdem alle Bedingungsblöcke bestimmt wurden, kann die Anfrage zusammengesetzt werden. Die Bedingungsblöcke, die zu einem Verweis gehören, werden durch ein logisches ODER verbunden, die Bedingungsblöcke der einzelnen Verweise untereinander mit einem logischen UND.

Wenn nun die einzelnen Blöcke zur gesamten Anfrage zusammengesetzt werden, ergibt sich:

```

SELECT
  ?X
WHERE
  (?X, ?anonym1, ?anonym2), (?X, ?anonym3, ?anonym4), (?X, ?anonym5, ?anonym6),
  (?X, <http://purl.org/dc/elements/1.1/author>, "Peter Norvig")
AND
  (
    (
      (?anonym1 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
      && (?anonym2 eq <http://www.lit.edu/types#AI-Book>))
    && (
      (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
      && (?anonym4 eq <http://www.lit.edu/types#AI-Book>))
  )
  || (
    (
      (?anonym1 eq <http://purl.org/dc/elements/1.1/title>)
      && (?anonym2 eq "Artificial Intelligence"))
    && (
      (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
      && (?anonym4 eq <http://www.lit.edu/types#Book>))
  )
  ||
  (
    (
      (?anonym1 eq <http://purl.org/dc/elements/1.1/title>)
      && (?anonym2 eq "AI"))
    && (
      (?anonym3 eq <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>)
      && (?anonym4 eq <http://www.lit.edu/types#Book>))
  )
)
&& (
  (?anonym5 eq <http://purl.org/dc/elements/1.1/author>)
  && (?anonym6 eq "Stuart Russel"))

```

## 5.4 Übersetzung der Anfrageergebnisse von RDQL in das Edutella Resultset

Die Übersetzung der Ergebnisse der RDQL-Abfrage in das EQM erfolgt auf einfache Weise. Das Ergebnis der Anfrage wird als Variablenbindung zurückgegeben. Diese Variablen werden in der Reihenfolge, die in der EQM-Anfrage vorgegeben ist, in das EQM übertragen und anschließend das Ergebnis zurückgegeben.

# A BNF

## A.1 BNF-Datalog

```

datalog ::= edurule* eduquery
eduquery ::= "?"- eduliteral ( "," eduliteral )* "."
edurule ::= edustatementliteral ":"-
          eduliteral ( "," eduliteral )* "."

eduliteral ::= educonditionstatement
            | negated_edustatementliteral
            | edustatementliteral

educonditionstatement ::= atom op atom
atom ::= variable_or_predicate | constant
edustatementliteral ::= variable_or_predicate "(" atom ( "," atom )* ")"
negated_edustatementliteral ::= ( "NOT" | "not" ) "(" edustatementliteral ")"
op ::= "!=" | "=" | ">" | "<" | "<=" | ">="
variable_or_predicate ::= ["A"- "Z", "a"- "z", "0"- "9"]
                        ( ["a"- "z", "A"- "Z", "0"- "9", ":", " ", "/", "?", "-",
                          "$", "!", "%", "#",
                          " ", "_", ";", "." ] )*

constant ::= " " ( ["a"- "z", "A"- "Z", "0"- "9",
                  ":", " ", "/", "?", "-", "$", "!", "%", "#",
                  " ", "_", ";", "." ] )* " "

```

Quelle: [8]

## A.2 BFN-RDF

RDF	::= ['<rdf:RDF>'] description* ['</rdf:RDF>']
description	::= '<rdf:Description' idAboutAttr? '>' propertyElt* '</rdf:Description>'
idAboutAttr	::= idAttr   aboutAttr
aboutAttr	::= 'about=" URI-reference "'
idAttr	::= 'ID=" IDsymbol "'
propertyElt	::= '<' propName '>' value '</' propName '>'   '<' propName resourceAttr '>'
propName	::= QName
value	::= description   string
resourceAttr	::= 'resource=" URI-reference "'
Qname	::= [ NSprefix ':' ] name
URI-reference	::= string, interpreted per [URI]
IDSymbol	::= (any legal XML name symbol)
name	::= (any legal XML name symbol)
Sprefix	::= (any legal XML namespace prefix)
string	::= (any XML text, with "<", ">", and "&" escaped)

## Ergänzungen für die abkürzende Schreibweise:

description	::= '<rdf:Description' idAboutAttr? propAttr* '>'   '<rdf:Description' idAboutAttr? propAttr* '>' propertyElt* '</rdf:Description>'
propertyElt	::= '<' propName '>' value '</' propName '>'   '<' propName resourceAttr? propAttr* '>'
propAttr	::= propName '=' string '"' (with embedded quotes escaped)
typedNode	::= '<' typeName idAboutAttr? propAttr* '>'   '<' typeName idAboutAttr? propAttr* '>' property* '</' typeName '>'

Quelle: [14]

## A.3 BNF-RDQL

CompilationUnit	::= Query <EOF>
Query	::= SelectClause ( SourceClause )? TriplePatternClause ( ConstraintClause )? ( PrefixesClause )?
SelectClause	::= ( <SELECT> Var ( "," Var )*   <SELECT> "*" )
SourceClause	::= ( <SOURCE>   <FROM> ) SourceSelector
SourceSelector	::= URL
TriplePatternClause	::= <WHERE> TriplePattern ( "," TriplePattern )*
ConstraintClause	::= <SUCHTHAT> Expression ( ( ","   <SUCHTHAT> ) Expression )*
TriplePattern	::= <LPAREN> VarOrURI "," VarOrURI "," VarOrLiteral <RPAREN>
VarOrURI	::= Var   URI
VarOrLiteral	::= Var   Literal
Var	::= "?" Identifier
PrefixesClause	::= <PREFIXES> PrefixDecl ( "," PrefixDecl )*
PrefixDecl	::= Identifier <FOR> URI
Expression	::= ConditionalOrExpression
ConditionalOrExpression	::= ConditionalXorExpression ( <SC_OR> ConditionalXorExpression )*
ConditionalXorExpression	::= ConditionalAndExpression
ConditionalAndExpression	::= ValueLogical ( <SC_AND> ValueLogical )*
ValueLogical	::= StringEqualityExpression
StringEqualityExpression	::= NumericalLogical ( <STR_EQ> NumericalLogical   <STR_NE> NumericalLogical )*
NumericalLogical	::= InclusiveOrExpression
InclusiveOrExpression	::= ExclusiveOrExpression ( <BIT_OR> ExclusiveOrExpression )*
ExclusiveOrExpression	::= AndExpression ( <BIT_XOR> AndExpression )*
AndExpression	::= ArithmeticCondition ( <BIT_AND> ArithmeticCondition )*
ArithmeticCondition	::= EqualityExpression
EqualityExpression	::= RelationalExpression ( <EQ> RelationalExpression   <NEQ> RelationalExpression )?
RelationalExpression	::= NumericExpression ( <LT> NumericExpression   <GT> NumericExpression   <LE> NumericExpression   <GE> NumericExpression )?
NumericExpression	::= ShiftExpression
ShiftExpression	::= AdditiveExpression ( <LSHIFT> AdditiveExpression   <RSIGNEDSHIFT> AdditiveExpression   <RUNSIGNEDSHIFT> AdditiveExpression )*
AdditiveExpression	::= MultiplicativeExpression ( <PLUS> MultiplicativeExpression   <MINUS> MultiplicativeExpression )*
MultiplicativeExpression	::= UnaryExpression ( <STAR> UnaryExpression   <SLASH> UnaryExpression   <REM> UnaryExpression )*
UnaryExpression	::= UnaryExpressionNotPlusMinus   ( <PLUS> UnaryExpression   <MINUS> UnaryExpression )
UnaryExpressionNotPlusMinus	::= ( <TILDE>   <BANG> ) UnaryExpression   PrimaryExpression
PrimaryExpression	::= Var   Literal   FunctionCall   <LPAREN> Expression <RPAREN>
FunctionCall	::= Identifier <LPAREN> ArgList <RPAREN>
ArgList	::= VarOrLiteral ( "," VarOrLiteral )*
Literal	::= URI   NumericLiteral   TextLiteral   BooleanLiteral   NullLiteral
NumericLiteral	::= ( <INTEGER_LITERAL>   <FLOATING_POINT_LITERAL> )
TextLiteral	::= <STRING_LITERAL>
BooleanLiteral	::= <BOOLEAN_LITERAL>
NullLiteral	::= <NULL_LITERAL>
URL	::= URI
URI	::= "<" <URI> ">"
Identifier	::= <IDENTIFIER>

Quelle: [12]

## A.4 BNF-SqishQL

```

CompilationUnit ::= Query <EOF>
Query ::= SelectClause ( FromClause )?
      TriplePatternClause ( ConstraintClause )?
      ( UsingClause )?

SelectClause ::= 'SELECT' VarList
FromClause ::= 'FROM' UriList
TriplePatternClause ::= 'WHERE' TriplePatternList
ConstraintClause ::= 'AND' ConstraintList
UsingClause ::= 'USING' ( ForList )+
TriplePatternList ::= TriplePattern ( TriplePattern )*
TriplePattern ::= '( VarOrLiteral VarOrLiteral VarOrLiteral )'
VarOrLiteral ::= Var | Literal
Var ::= '?' Identifier
VarList ::= Var ( ( ',' )? Var )*
UriList ::= UriLiteral ( ( ',' )? UriLiteral )*
ConstraintList ::= Expression ( 'AND' Expression )*
ForList ::= Identifier 'FOR' UriLiteral
Expression ::= Var SomeFunction
SomeFunction ::= ( NumExpression | StringExpression )+
NumExpression ::= ( '>' | '<' | '==' | '=' | '!=' | '<=' | '>=' )
      NumericLiteral
StringExpression ::= ( 'like' | 'ne' | 'eq' | ' ' ) Literal
Literal ::= TextLiteral | UriLiteral | NumericLiteral
NumericLiteral ::= An integer | A floating point number
UriLiteral ::= A letter followed by none or more letters, numbers or
      other characters allowed in RFC 2396
TextLiteral ::= One or more letters or numbers enclosed in
      inverted commas
Identifier ::= A letter followed by optional numbers and letters.

```

Quelle: [13]

## Literatur

- [1] Collab.Net, Inc; Develop Open Source Software
- [2] W3C; RDF Model and Syntax Specification, <http://www.w3.org/TR/REC-rdf-syntax/>
- [3] Sun Microsystems, Inc.; Project JXTA: An Open, Innovative Collaboration
- [4] Sun Microsystems, Inc.; JXTA Search: Distributed Search for Distributed Networks
- [5] Li Gong, Sun Microsystems, Inc.; [Project JXTA: A Technology Overview](#) ; 2002-10-29
- [6] Sun Microsystems, Inc.; [JXTA v2.0 Protocols Specification](#) ; 2003-02-27
- [7] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, Tore Risch; [EDUTELLA: A P2P Networking Infrastructure Based on RDF](#) ; November 14, 2001
- [8] Boris Wolf; [Peer-to-Peer Networking for Distributed Learning Repositories](#) ; Dezember 19, 2001
- [9] Wolfgang Nejdl, Boris Wolf L3S and Knowledge Based Systems Steffen Staab, Julien Tane L3S and Institute AIFB; "EDUTELLA: Searching and Annotating Resources within an RDFbased P2P Network"
- [10] Mikael Nilsson <mini@nada.kth.se>, Royal Institute of Technology, Stockholm, Wolf Siberski <siberski@learninglab.de>, Learning Lab Lower Saxony, Hannover; [RDF Query Exchange Language \(QEL\) - concepts, semantics and RDF syntax](#)
- [11] Andy Seaborne ; [A Programmer's Introduction to RDQL](#) ; April 2002
- [12] HEWLETT-PACKARD COMPANY WEBSITE; [BNF for RDQL](#) ;
- [13] Libby Miller; [BNF for SquishQL](#) ; ILRT, Bristol University, UK; 2002-07-09
- [14] Ora Lassila (Nokia Research Center), Ralph R. Swick (World Wide Web Consortium ); [RDF Model and Syntax Specification](#) ; 1999-02-22
- [15] Rainer Böhme; [Resource Description Framework](#) ; 1999-06-15
- [16] Dan Brickley (W3C/ILRT), R.V. Guha (IBM) ; [RDF Schema Specification 1.0](#) ; 1999-06-15