IWI Seminar Notes

# Einführung in RDF

RDF in wissenschaftlichen Bibliotheken

2. vollständig überarbeitete und erweiterte Auflage


mit Beiträgen von:

- Elena Demidova
- Shirish Kucheria
- Judith Plümer
- Hartmut Polzer
- Roland Schwänzl
- Thomas Severiens (Ed.)
- Stefan Wenneker

Osnabrück, Juni 2006

# Inhaltsverzeichnis

# Vorwort

Der Traum des Semantic Web ist noch nicht ausgeträumt. Er gewinnt sogar durch Techniken und Dienste wie WebServices, GRID-Netzwerke, Institutional Repositories usw. zunehmend an Realisierbarkeit. Voraussetzung ist jedoch, dass wenigstens jene Struktur, die Information als Ergebnis von Arbeitsprozessen in sich trägt, auch erhalten bleibt, transportiert wird und der entstehende Mehrwert bei Sichtbarkeit und Erhaltung (Stichwort: Langzeitarchivierung) genutzt wird sowie redundante Arbeitsschritte vermieden werden. Nur bei Implementierung aller vorhandener Synergien in die Arbeitsgänge von Publikation und Archivierung wird es mittelfristig möglich – weil ökonomisch lohnend – sein, den Tendenzen der Verflachung der Informationslandschaft und Discounterierung der Publikationskultur effektiv Einhalt zu gebieten.

Das vorliegende Skript entstand aus einer Schulung „RDF in wissenschaftlichen Bibliotheken“, die das Institut für wissenschaftliche Information Osnabrück e.V. im Juni 2000 für Die Deutsche Bibliothek durchgeführt hat. Zielgruppe sind Bibliothekare, die Vorkenntnisse in HTML und Dublin-Core haben, jedoch noch unerfahren in Techniken des Semantic Web, wie RDF und RDFS, sind.

Die zweite Auflage entstand in 2005/2006 im Rahmen des BMBF-Projektes Cashmere-int. Sie wurde an den 2004 überarbeiteten RDF-Standard angepasst und um aktuelle Entwicklungen der Jahre 2000 bis 2005 ergänzt.

Mit Dank an die Autoren für die Bereitstellung der Beiträge, Frau Grigoras und Herrn Roggenbuck für die Fehlersuche und Zusammenstellung der Dateien. In memoriam Prof. Dr. Roland Schwänzl, dem Herausgeber der ersten Auflage.

Thomas Severiens (IWI)

# Das Graphenmodell von RDF

Es ist üblich, MetaDaten mit dem `<Meta>` Tag von HTML im `<Head>`-Bereich von HTML Dokumenten unterzubringen. Diese Art der Übermittlung bringt eine Reihe von Beschränkungen mit sich.

Einige Probleme sehen wir im folgenden kleinen Beispiel:

```
<HTML>
<HEAD>
<META NAME="DC.subject.msc" CONTENT="(SCHEME=msc91) 19-XX">
<META NAME="DC.Format" CONTENT="application/x-dvi">
<META NAME="DC.Identifier" CONTENT="(SCHEME=url)
            http://www.math.uiuc.edu/K-theory/0316/vv.dvi">
<META NAME="DC.Format" CONTENT="application/postscript">
<META NAME="DC.Identifier" CONTENT="(SCHEME=url)
            http://www.math.uiuc.edu/K-theory/0316/vv.ps.gz">
<META NAME="DC.Identifier.mirror" CONTENT="(SCHEME=url)
       http://www.mathematik.uni-osnabrueck.de/K-theory/0316/vv.ps.gz">
</HEAD>
```

Die Probleme der HTML-Kodierung von Metadaten sind im einzelnen:

Fehlende Gruppierungsmöglichkeit: Welcher `DC.Identifier` gehört zu welcher `DC.Format`-Angabe?

Fehlende Typisierungsmöglichkeit: 19-XX ist ein Klassifikationscode. Dies ist durch die `SCHEME`-Angabe nicht darstellbar. Zur Verbesserung der Nachvollziehbarkeit könnte dem Klassifikationscode auch die Klartextfassung: „Algebraic K-Theory“ folgen.

Vermischung von Objekttypisierung und Beziehungsverfeinerung: Die Beziehung `DC.Identifier` wird nicht durch den Umstand enger gefasst, dass „http://www.mathematik.uni-osnabrueck.de/K-theory/0316/vv.ps.gz“ den Standort eines Mirrors wiedergibt.

HTML Meta Probleme:

- Assoziativ
- Kommutativ
- Beschreibungsschwierigkeiten bei Multi-Part und abstrakten Objekten, Autoren und Organisationen werden gleichgesetzt
- Warwick Framework: Koexistenz, aber keine Kooperation von Schemata realisierbar
- Kein Recycling von Semantik
- Beziehungen zwischen Tag-Namen nicht beschreibbar
- Schemata: Link-Tag nicht funktional an Meta-Tag gebunden.
- Namenskollisionen müssen durch zusätzliche Konventionen vermieden werden.
- Versionsproblem
- Vokabular Inflation

Fehlende Strukturierungsmöglichkeiten wohin man blickt.

# RDF Modell und Syntax Spezifikation

## Entstehung

Das Resource Description Framework (RDF) ist ein Ergebnis der Arbeiten des W3 Consortiums an der Web- Infrastruktur für MetaDaten. Standardisiert wurde RDF erstmals mit der W3C Recommendation vom 22. Februar 1999 [RDF-MSS]. Diese wurde am 10. Februar 2004 ersetzt durch eine überarbeitete Recommendation [RDF-XSS].

## Motivation und Zielsetzung

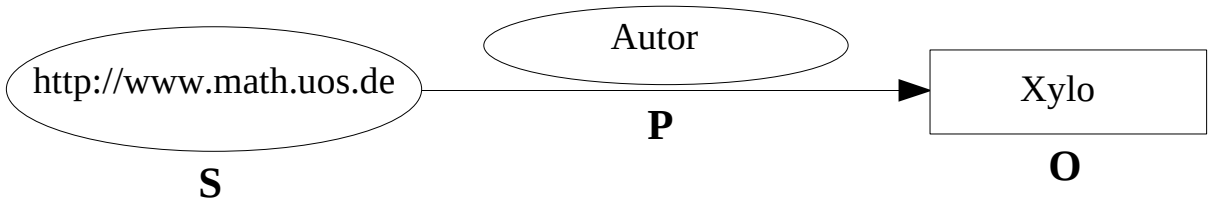
Die Notwendigkeit zu einer solchen Infrastruktur haben wir eingangs an den Defiziten von HTML Meta erläutert.

- Grundbedingung für Interoperabilität mit den übrigen Aktivitäten zur Stärkung der Web-Infrastruktur ist eine Bindung an XML, das die Rolle des physikalischen Übermittlungsmediums einnimmt.
- Gleichzeitig muss die Verwaltung der kodierten Semantik durch Datenbanken im Grundsatz gesichert sein.
- Last but not least: Die Spezifikation muss die Entwicklung graphisch orientierter Design- und Ausgabewerkzeuge ermöglichen.

Im Detail sind diese Ansprüche nicht deckungsgleich. Es resultieren einerseits Einschränkungen andererseits Relationen, die bei einer Beschränkung des Blickwinkels auf einen der drei Aspekte hätten vermieden werden können.

## Ausgangspunkt

Ausgangspunkt der Überlegungen zu RDF sind die Vergleichssätze menschlicher Sprachen: „O ist der P von S“ oder „S hat als P O“.




Bei diesem Ausgangspunkt liegt eine Datenbankverwaltung durch Tripel und eine graphische Darstellung durch gerichtete Graphen mit Bezeichnungen (Labeled Directed Graphs) nahe.

## Aufgabe der Spezifikation

„Konstruiere eine Klasse A von wohlgeformten XML Dokumenten, eine Klasse B von Tripelmengen und eine Klasse C von Labeled Directed Graphs zusammen mit Transformationsregeln zwischen ihnen.“

## Theoretisches

**Definition: RDF Objekt:** Ein 3- Tupel (a, b, c), wobei a aus A, b aus B und c aus C ist, heißt ein RDF Objekt, wenn aus einer der drei Komponenten die beiden anderen durch die Transformationsregeln hervorgehen.

**Definition: RDF Dokument:** Zwei RDF Objekte sind Repräsentanten desselben RDF Dokuments, wenn sie in wenigstens einer Komponente (Repräsentation) gleich sind.

### Ziel dieses Kapitels: Die Graphenrepräsentation von RDF Dokumenten

Die konkrete Beschreibung der XML Dokumente, die zu einem RDF Objekt gehören, ist Inhalt anderer Kapitel.

- Der Wunsch nach Existenz einer XML Darstellung stellt die stärkste Einschränkung bei der Konstruktion von RDF Objekten dar, führt aber auch zu Äquivalenzsetzungen.
- Labeled Directed Graphs führen zu den stärksten Äquivalenzsetzungen. Auch der Wunsch nach Beschreibung durch Tripel hat Äquivalenzen zur Folge. Diese können als weitere Einschränkung für die Graphen beschrieben werden.
- Umformulierung und Konsequenz:
  RDF Dokumente können durch Isomorphieklassen von Labeled Directed Graphs beschrieben werden. Jede solche Isomorphieklasse bestimmt das RDF Dokument eindeutig.

### Notation für Namen

Der Wunsch nach Interoperabilität mit allgemeiner Web Infrastruktur hat - teilweise bereits durch die Bindung an XML - auch Konsequenzen bezüglich der Notation.

Will man einem Gegenstand einen Web-Bezeichner geben, so muss dazu der IETF RFC 3986: Uniform Resource Identifier (URI): Generic Syntax [RFC3986] verwendet werden. RDF „versteht“ also grundsätzlich folgende Notation nicht: „Es bezeichne 'this:that' die Menge der reellen Zahlen.“

Im Grundsatz würde: „Es bezeichne './this:that' die Menge der reellen Zahlen.“ „verstanden“ werden. Das Problem liegt in der besonderen Verwendung des Doppelpunktes in der URI Syntax.

## Was ist eine Resource?

### Benannte Resourcen

- RDF beschränkt sich nicht auf die Beschreibung von Objekten, die es irgendwo im Web gibt. RDF beschränkt lediglich den Zeichenvorrat für Objektnamen auf URI und unterwirft den Zeichenvorrat den in RFC3986 gemachten syntaktischen Bedingungen.
- Die benannten Resourcen eines RDF Objektes kann man aufteilen in die referenzierten und die im Objekt neu benannten Resourcen. Das Verhalten dieser beiden Typen ist in der Graphenrepräsentation gleich. Sie spielt daher in diesem Vortrag - im Gegensatz zur XML-Repräsentation - keine weitere Rolle.

Die Verwendung von Bezeichnern hat eine relevante praktische Konsequenz:

Es ist nicht möglich in einem RDF Objekt einen identischen Bezeichner für verschiedene Dinge zu verwenden.

### Unbenannte Resourcen

In menschlicher Sprache ist es möglich über Dinge zu reden, ohne ihnen einen Namen zu geben.

- A besitzt ein Ding. Dieses Ding ist ein Buch.
- ....Gestern Nacht kam ein Männlein in ihre Kammer und hat alles Flachs für sie gesponnen....

(Wir wissen alle zu welchem Drama sich die Namensfindung ausgeweitet hat.)

Wollen wir also zulassen, dass Aussagen über Dinge mit unbekanntem Namen oder auch Dinge, die wir nicht benennen wollen oder Dinge, die (derzeit) keinen Namen besitzen, gemacht werden, so müssen wir Mechanismen einbauen, die eine Identifizierung und Unterscheidung aus dem Kontext zulassen.

In der RDF Standardisierung [RDF-XSS] wird ein sicherer Standpunkt eingenommen:

- Namenlose (privat gehaltene) Objekte sind zulässig. Hierzu wird ihnen ein Identifier zugeordnet, der aus dem Namespace „_“ stammt. Sofern nicht jenseits aller Zweifel die Identität feststeht, werden namenlose Objekte nicht einander gleichgesetzt, also die Identifier eindeutig (unique) vergeben.

### Literals

- In der RDF/XML Standardisierung [RDF-XSS] werden Aufmacher die Literals des RDF Objektes genannt.

Auch bei der Definition von Literals macht sich die Bindung an XML bemerkbar:

- Die Literals eines RDF Objektes bilden eine Menge. Die Elemente dieser Menge sind selbst beliebiges wohlgeformtes XML. Im weiteren bezeichnen wir mit Literals stets endliche Mengen von wohlgeformtem XML.

## Theoretischer Exkurs

Wir sind nun soweit, die Tripel im Sinne der RDF Standardisierung formal beschreiben zu können. Dazu führen wir zunächst den Begriff eines RDF Modells ein.

**Definition**: RDF Modell: Gegeben sei eine Menge R. Eine Teilmenge P von R, eine Menge L und eine Teilmenge S von R x P x (R $\cup$ L). Das Viertupel M = (R, P, L, S) heißt ein RDF Modell.

Die Menge R wird die Menge der Resourcen von M, P die Menge der Properties (Eigenschaften), L die Menge der Literals in M und S die Menge der Statements (facts) von M genannt.

**Definition**: Feines RDF Modell: Es sei M = (R, P, L, S) ein RDF Modell. Für R sei eine Zerlegung in disjunkte Teilmengen X, Y gegeben und eine Darstellung von X als Teilmenge von (URI - absolut).

Das Sechstupel N = (R, P, L, S, X, Y) heißt ein feines Modell. Die Statements (a, b, c) mit b in X heißen die Tripel des feinen Modells.

**Definition**: Minimales feines Modell: Ein feines Modell N heißt minimal, wenn die Vereinigung des Bildes von erster Projektion, zweiter Projektion und dritter Projektion der Tripel von N die Resourcen R umfasst, die zweite Projektion den Durchschnitt von P mit X und die dritte Projektion L umfasst.

**Bemerkung**: Zu jedem feinen RDF Modell gibt es ein minimales feines Modell.

Minimale Modelle haben die Eigenschaft, dass jede Resource in einem Statement vorkommt und jedes Literal wenigstens einmal adressiert wird.

**Definition**: Es seien $N_1$, $N_2$ zwei feine Modelle. $N_1$, $N_2$ heißen modellisomorph, wenn $X_1 = X_2$ und es eine Bijektion f: $Y_1 \rightarrow Y_2$ gibt.

**Definition**: Modelläquivalenz: Zwei feine Modelle $N_1$ und $N_2$ heißen modelläquivalent, wenn die zugeordneten minimalen feinen Modelle modellisomorph sind.

**Satz**: Durch [RDF-XSS] wird für jedes RDF/XML Dokument ein bis auf Modelläquivalenz eindeutiges feines RDF Modell konstruiert.

**Definition**: Das Tripelmodell von RDF besteht aus den minimalen feinen Modellen in deren Äquivalenzklasse ein durch [RDF-XSS] konstruiertes Modell liegt.

## RDF Graphen

Die formale Beschreibung von Tripelrepräsentationen von RDF Dokumenten hört sich grauslich kompliziert an. Etwas informeller kann man aber zumindest im Fall einer endlichen Resourcenmenge formulieren:

- Eine Tripeldarstellung erhält man durch Auflistung aller im RDF/XML Objekt gemachten Statements. Dabei erhalten die privaten Resourcen interne, eindeutige Identifier. Die Namen benannter Resourcen werden in ihrer vollständig aufgelösten Form angegeben. Literals sind als solche erkennbar.
- Der Übergang zur Graphendarstellung wird uns vom Zwang, interne, eindeutige Identifier finden und hinschreiben zu müssen, die dann aber doch als solche irrelevant sind, befreien.

Mathematisch gesehen können die RDF Resourcen aus diesen gutartigen XML Dokumenten in aufzählender Mengenschreibweise gegeben werden.

Ein Tripel hat dann die Gestalt:

- ([URI(Subject)], [URI(predicate)], [URI(Object)] ) oder
- ([URI(Subject)], [URI(predicate)], Literal)

Es hat sich eingebürgert, die neu definierten Resourcen mit

- URI#gegebenerName

und die unbenannten Resourcen mit

- genidNummer

zu kennzeichnen.

## Vom Tripel zum einfachsten Graphen

Ein Graph besteht aus Knoten und Verbindungsstücken. Bei einem gerichteten Graphen hat das Verbindungsstück eine Richtung. Ein benannter Graph trägt Text- oder andere Merkmale an Knoten und Pfeilen.

- Für ein Tripel ($URI_1$, $URI_2$, $URI_3$) setzen wir je eine Ellipse für $URI_1$ und $URI_3$ aufs Blatt Papier. In die Ellipsen tragen wir $URI_1$ bzw. $URI_3$ ein.
- Wir ziehen einen Pfeil von der $URI_1$ Ellipse zur $URI_3$ Ellipse. Auf den Pfeil schreiben wir $URI_2$.
- Für ein Tripel ($URI_1$, $URI_2$, Literal) verfahren wir fast genauso. Statt der Ellipse für $URI_3$ malen wir ein Rechteck.

## Von Tripelmengen zu Graphen

Wir haben beschrieben wie wir mit der Abarbeitung einer Tripelmenge beginnen. Wir beschreiben jetzt wie wir mit weiteren Tripeln fortfahren.

- Sei ($URI_x$, $URI_y$, $URI_z$) das nächste Tripel. Falls für $URI_x$ und $URI_z$ noch keine Ellipse existiert, verfahren wir wie oben.
- Gibt es für $URI_x$ oder $URI_z$ schon eine Ellipse, so verbinden wir diese mit einem (weiteren) Pfeil auf den wir $URI_y$ schreiben.
- Für Literals verfahren wir analog.

Da wir angenommen haben, dass die Tripelmenge endlich ist, sind wir nach endlich vielen Schritten fertig.

Bislang haben wir uns nur von der zufälligen Reihenfolge bei der Auflistung der Tripel befreit. Das Ergebnis ist von der erzwungenen Wahl von Namen für private Resourcen unabhängig. Natürlich ist die Lage der Ellipsen und Rechtecke, der Größe und Länge und Krümmung der Verbindungsstücke immateriell.

Jedem steht es somit frei, für sich besonders schöne Graphenvisualisierungen zu wählen. Es ergibt sich ein reiches Betätigungsfeld für Werkzeugproduzenten.

## Theoretischer Exkurs

In völliger Strenge ist die Graphenrepräsentation eines RDF Objektes nicht die soeben beschriebene Visualisierung, sondern eine Äquivalenzklasse minimaler feiner RDF Modelle.

Allerdings lässt sich der Graph nicht mehr durch eine Menge in aufzählender Schreibweise notieren. Lassen wir „...“ bei der Visualisierung zu, so verallgemeinern wir die zuvor beschriebene Prozedur.

## Sinn der Graphenrepräsentation / Visualisierung

Zu einem nützlichen Werkzeug und nicht nur zu theoretischen Überlegungen wird die Graphendarstellung, wenn wir Bedingungen angeben können unter denen ein Graph ein RDF Graph ist, d.h. wenn es ein XML Dokument gibt, dessen Graph der gegebene ist.

Solche Bedingungen wollen wir jetzt formulieren. Sie sind beim Design von MetaDaten-Profilen außerordentlich nützlich.

Es macht einen erheblichen Unterschied, ob wir mit benannten Resourcen oder privaten hantieren wollen. Bei den privaten Resourcen muss sich deren Identität aus dem Kontext ergeben.

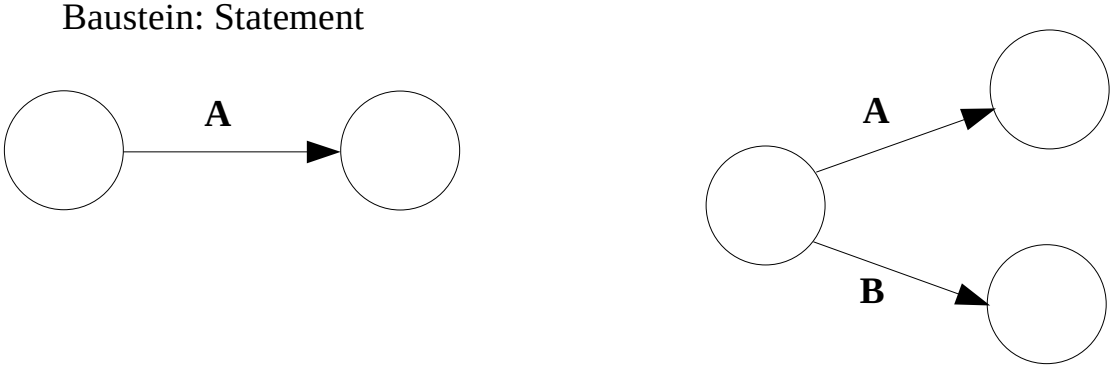
### Vorgehensweise

- RDF Graphenbausteine
- Zusammensetzungsregeln für Bausteine

Insbesondere beim zweiten Punkt unterscheiden wir zwischen benannten und unbenannten Resourcen.

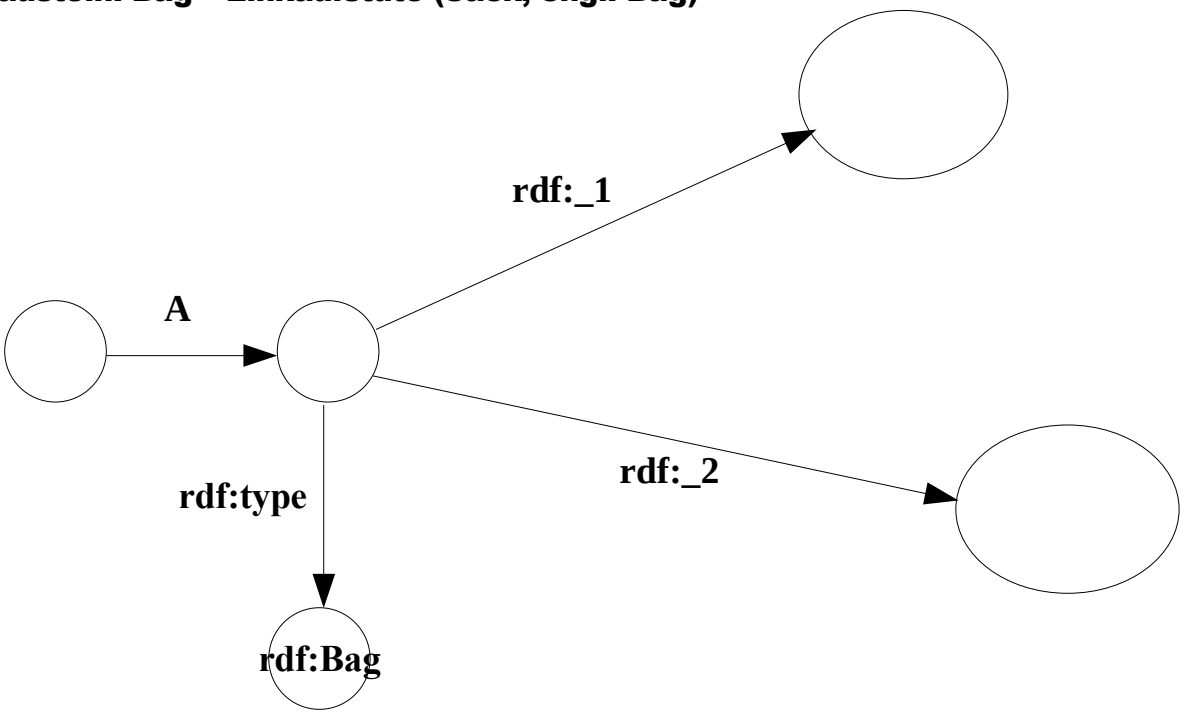
Baustein: Ver-undung

Baustein: Statement




Beide Aussagen können für sich alleine stehen. Volle Information wird nur bei Beachtung beider erhalten. Es kann eine beliebige (endliche) Menge von Statements mit identischem Subjekt erzeugt werden. Unbenannte Objekte können nur von einer Property angesteuert werden. Eine Ausnahme ergibt sich bei der Reifizierung.

### Baustein: Bag – Einkaufstüte (Sack, engl. Bag)



Die Aussagen ergeben nur zusammengenommen einen Sinn. Ihre Reihenfolge ist irrelevant. Es kann eine beliebige (endliche) Anzahl von Aussagen gebündelt werden. Die von den Zählelementen rdf:_i angesteuerten Resourcen heißen die Objekte des Bags. Der Ausgangsknoten eines Bags kann aber muss nicht benannt sein.

### Baustein: Seq - Sequence

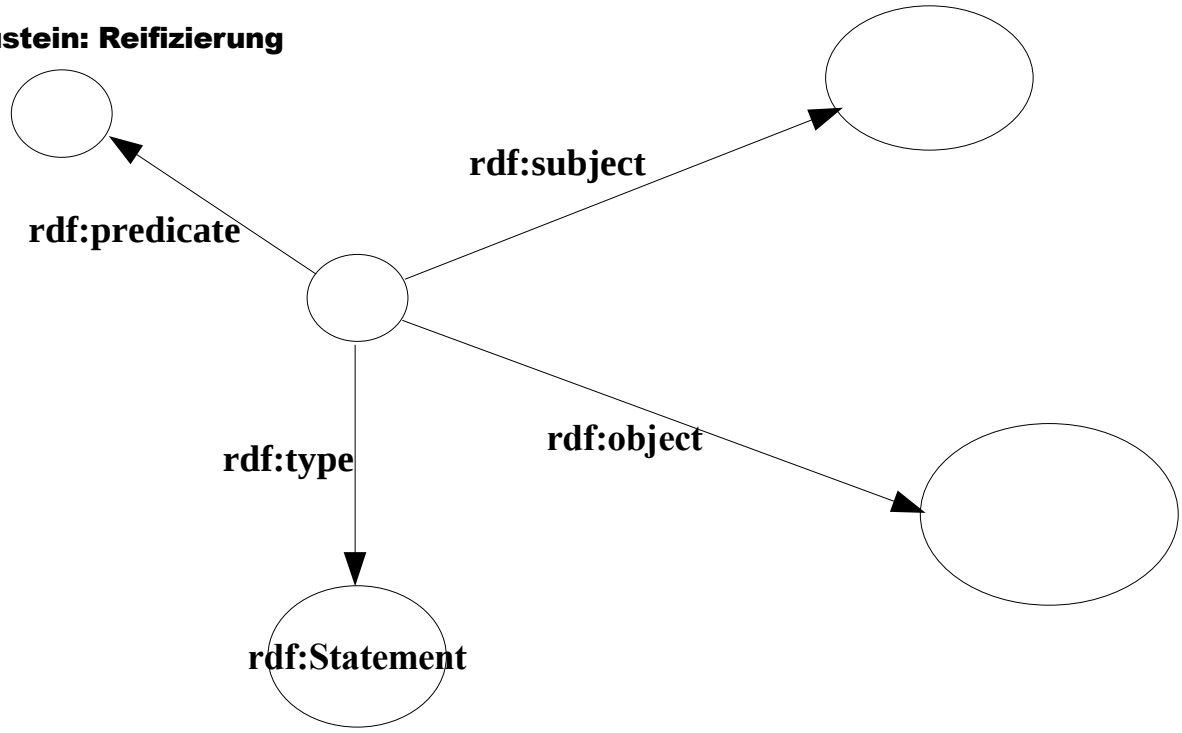
Die Graphendarstellung hat statt des Labels **Bag** das Label **Seq**. Die Reihenfolge ist bei Seq wichtig. Dieses Konstrukt eignet sich besonders für Listen. Nützlich ist es, die Art der Ordnung zu beschreiben.

### Baustein: Alt - Alternative

Zur Erfüllung der eingehenden Eigenschaft genügt ein Zweig einer Alternative. Die Graphendarstellung unterscheidet sich nur im Label **Alt** vom Bag-Graphen.

Alternativen müssen wenigstens ein Element enthalten. Bags und Sequenzen dürfen leer sein.

### Baustein: Reifizierung




Der Baustein Reifizierung verursacht oft begriffliche Schwierigkeiten.

Die Standardanwendung der Reifizierungskonstruktion liegt bei Aussagen über Aussagen. Ihre Handhabung insbesondere im Zusammenhang mit unbenannten Resourcen, kann schwierig sein.

**Definition:** Jede Resource mit `URI#Name` oder eine unbenannte Resource, für die die angegebenen vier Statements gesetzt sind, wird eine Reifizierung genannt.

Subjekt, Prädikat und Objekt Statement müssen dabei genau einmal gesetzt sein.

- Die Objekt Resource von rdf:predicate ist per Definition die Property der Reifizierung.
- Die Semantik von Subject, Predicate, Object, Type sind von RDF fest definiert.
- rdf:type muss immer eine Resource als Ziel haben.
- Ist das Statement (Subjekt, Prädikat, Objekt) ebenfalls im RDF Objekt enthalten, so heißt die Reifizierung eine Reifizierung des gegebenen Statements.

- Handelt es sich bei Subjekt und Objekt um benannte Resourcen, so kann beim Aufbau eines RDF Graphen zuerst eine Reifizierung gesetzt werden und erst anschließend das einfache Statement (S,P,O). Dasselbe gilt falls O ein Literal ist.
- Ist S oder O unbenannt, so muss - falls man (S,P,O) überhaupt in den Graphen einbringen will - zuerst (S,P,O) gesetzt werden.

Die weiteren Punkte zur Reifizierung betreffen Sonderfälle, die im Zusammenhang mit unbenannten Resourcen Kopfzerbrechen bereiten können. Sie können in einem ersten Durchlauf überschlagen werden.

Ist O unbenannt, so kann (S,P,O) nur eine Reifizierung tragen. Ist P eine der Eigenschaften rdf:subject, rdf:predicate, rdf:object oder rdf:type und S vom Typ Statement, so kann es eintreten, dass nicht weiter reifiziert werden kann (siehe unten).

Ist S oder O unbenannt, so kann nur eine benannte Reifizierung unabhängig von weiteren Konstrukten hinzugefügt werden.

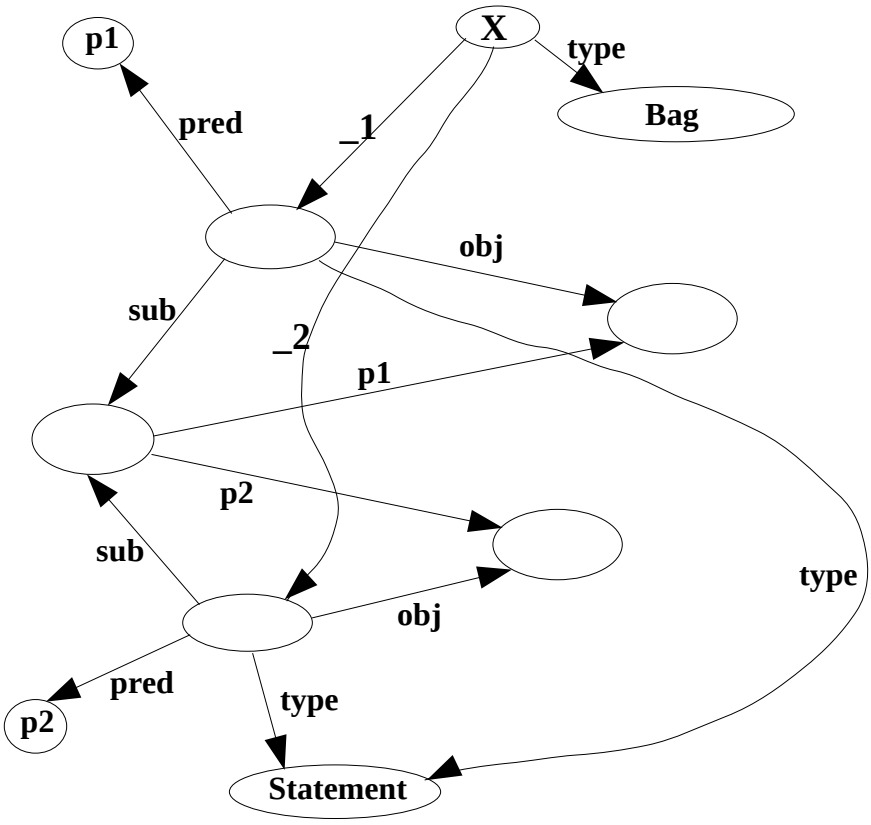
Die von einem Subjekt (unmittelbar) abgehenden Statements kann man gleichzeitig reifizieren. Gegebenenfalls bereits konstruierte Reifizierungen werden respektiert. Die neu hinzukommenden Reifizierungen sind unbenannt. Die Resourcen der auf diese Weise beschriebenen Reifizierungen sind die Objekte einer mit einer `URI#Name` benannten Resource vom rdf:type eines Bag.

Die rdf:subject, rdf:predicate und rdf:object Bögen von unbenannten Reifizierungen aus dem letzten Schritt, können selbst nicht reifiziert werden.

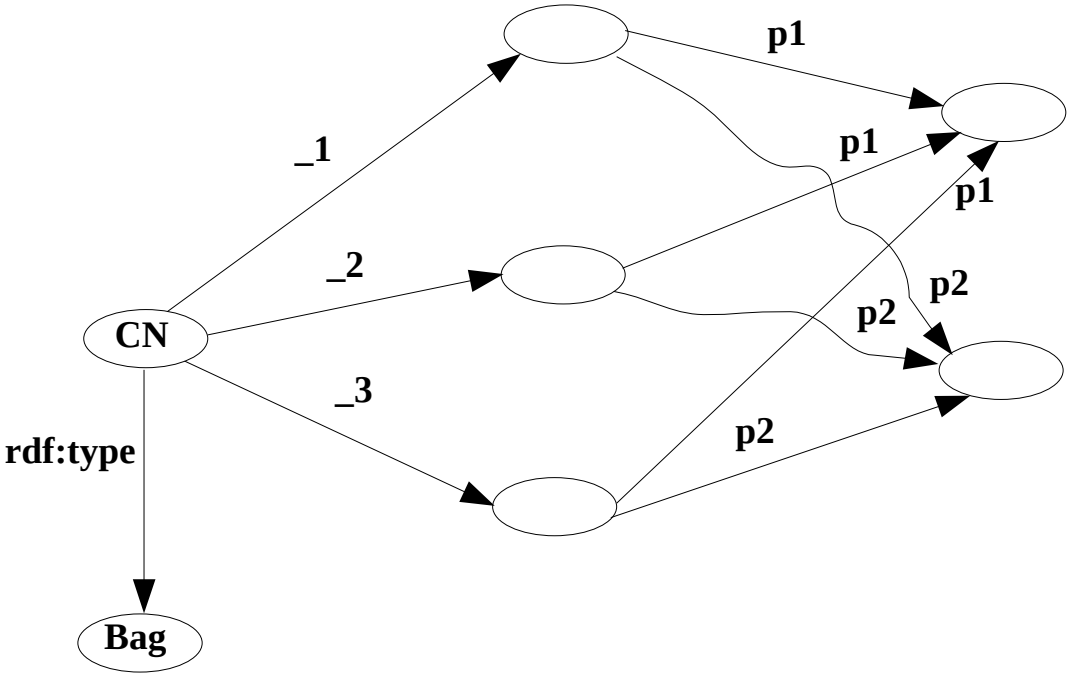
Als Ergebnis wollen wir zusammenfassen:

- Reifizierungen existieren in allen „vernünftigen“ Fällen.
- Reifizierungen sind nicht immer eindeutig.
- In der Regel wird man nur bereits vorhandene Statements reifizieren wollen.

**Baustein: Reifizierung im Sack**



### Baustein: Distributive Referenten




Als Containertypen sind auch Seq und Alt zulässig. Die Anker (Subjekte) beim distributiven Referenzieren sind die durch die Zählelemente ausgewiesenen Resourcen.

Literals können natürlich auch distributiv nicht referenziert werden, also nicht Subjekt eines Statements sein.

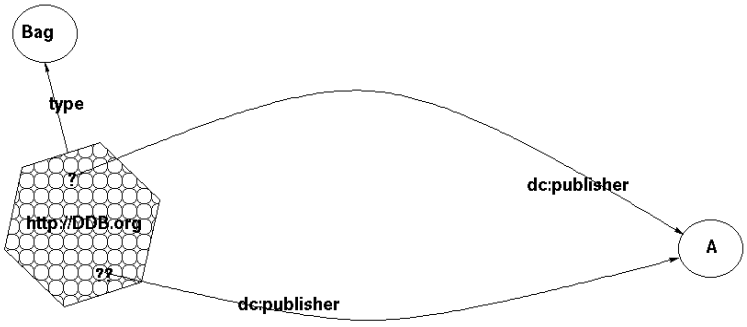
### Baustein: Komponenten von unbenannten Resourcen

Will man Aussagen zu unbenannten Resourcen verketten, so muss beachtet werden, dass abgesehen von der Reifizierung, eine unbenannte Resource stets nur Ziel einer einzigen Eigenschaft sein kann.

### Erweiterter Baustein: URI Präfix Distribution

Ein Präfix stellt man sich als Anfang von URI's vor. Präfixe verhalten sich dual zu Fragmentidentifiern. Es wird bei der Entwicklung etwa des Präfixes http://www.mathematik.uni-osnabrueck.de zur URL http://www.mathematik.uni-osnabrueck.de/gibtEsNicht/ erwartet, dass es sich bei .../gibtEsNicht/ um die URL etwa eines files handelt. Ob - und wenn ja wie - ein Name mit "Etwas" verknüpft ist, ist nicht die Sorge von RDF. Die Meinung bei URL's vom Typ http:// ist: Wenn das http Protokoll in der Lage ist eine Verbindung zu einem Objekt seines Gültigkeitsbereichs herzustellen, dann wird über dieses Objekt in RDF die dann folgende Aussage getroffen.

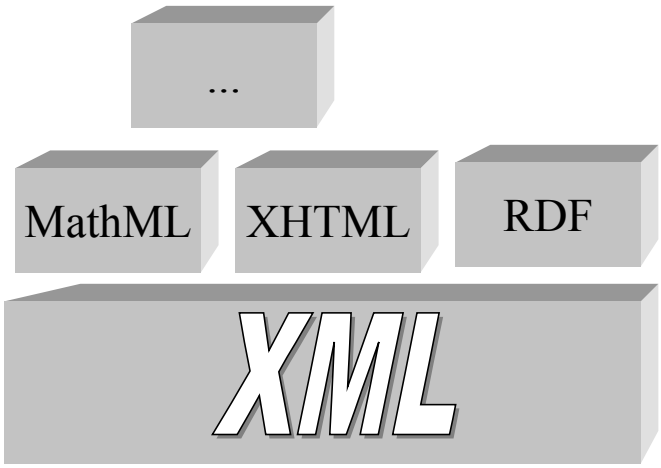
Ein Labeled Directed Graph der diesen Sachverhalt symbolisiert könnte etwa wie folgt aussehen:



# Einführung in XML

Die eXtensible Markup Language [XML] ist eine Metasprache für die Definition von Markup Sprachen. Sie unterscheidet sich durch ihre Fähigkeit, Markup Befehle und Attribute definieren zu können, wesentlich von anderen bekannten Markup Sprachen wie z.B. HTML, das nur die Nutzung einer bestimmten (und unveränderlichen) Menge von Markup Befehlen erlaubt.

Das seit Januar 2000 als Recommendation des W3C vorliegende XHTML [XHTML] ist eine XML konforme Reformulierung von HTML. Inzwischen existieren eine Fülle weiterer XML Anwendungen, wie MathML, CML, SOAP-WebServices, RSS etc.:




XML ist als Teilmenge von SGML nicht ganz so mächtig wie SGML, dafür aber sehr viel einfacher anzuwenden. Im Folgenden werden die grundlegenden Konzepte von XML vorgestellt.

## XML Dokumente

Das folgende Beispiel zeigt ein typisches XML Dokument:

```
<?xml version="1.0" ?>
<document class="H.3.3">
    <author>John Smith</author>
    <title>XML Retrieval</title>
    <chapter>
        <heading>Introduction</heading>
        This text explains all about XML and IR.
    </chapter>
    <chapter>
        <heading>Extensible Style Language</heading>
        <section>
            <heading>Examples</heading>
        </section>
        <section>
            <heading>Syntax</heading>
        </section>
    </chapter>
</document>
```

Im Gegensatz zu HTML wird in XML zwischen Groß- und Kleinschreibung unterschieden. Die erste Zeile gibt an, um welche XML Version es sich bei den vorliegenden Dokument handelt.

```
<?xml version="1.0" ?>
```

Bislang gibt es nur die Version "1.0". Nach dieser sogenannten Processing Instruction beginnt der eigentliche Inhalt des XML Dokumentes, der aus einer Reihe von Markup Befehlen und den Daten besteht, die in einer baumartigen Struktur angeordnet werden. Die eigentlichen Daten stehen zwischen dem Markup, z.B. `<heading>Syntax</heading>`. XML ist in Bezug auf die Syntax strenger als HTML, da jedes geöffnete Tag wieder geschlossen werden muss und keine überlappenden Tags (z.B. `<a><b></a></b>`) erlaubt sind.

Im Folgenden werden die wichtigsten Sprachkonstrukte im Einzelnen vorgestellt.

## Processing Instructions

Die Processing Instructions (PI) geben darüber Aufschluss, wie das folgende XML Dokument verarbeitet werden soll. Die erste Zeile eines XML Dokumentes muss die Processing Instruction `<?xml version="1.0" ?>` enthalten. Der innerhalb eines Dokumentes verwendete Zeichensatz kann über das Schlüsselwort encoding in der PI definiert werden, z.B. `<?xml version="1.0" encoding="UTF-8"?>`. Wird kein Zeichensatz spezifiziert, erwartet der XML Parser UTF-8[1]. Es können beliebige weitere PIs angegeben werden, die Metadaten über das XML Dokument enthalten. Sie dürfen nur nicht mit dem reservierten Wort xml starten.

## Tags

Die Struktur der Dokumente ist durch die Tags gegeben und entspricht einem Baum. Insbesondere muss es ein Wurzelelement geben. Der eigentliche Inhalt der Daten steht innerhalb der Tags. Ein leeres Tag `<a></a>` kann durch den Ausdruck `<a/>` abgekürzt werden. Jedes Tag muss geschlossen werden und es sind keine überlappenden Tags (also beispielsweise nicht: `<a><b></a></b>`, sondern stattdessen: `<a><b></b></a>`) erlaubt.

## Attribute

Tags können Attribute besitzen. Diese spezifizieren meistens die Daten, die das Tag enthält, etwas genauer. Ein Attribut darf höchstens einmal aufreten, d.h. eine Konstruktion wie `<mitarbeiter name="hans" name="emil">` ist verboten. Ob es sinnvoller ist, Daten in Attribute oder in eigenständigen Tags zu speichern, hängt von der jeweiligen Anwendung ab und ist oft nicht einfach zu entscheiden.

## Kommentare

Kommentare `<!-- Dies ist ein Kommentar -->` können wie normale Tags an beliebiger Stelle im XML Dokument eingefügt werden.

## Textknoten

Analog zu HTML kann man auch bei XML Markup und Text vermischen:

```
<chapter>
        <heading>Introduction</heading>
        This text explains all about XML and IR.
</chapter>
```

Diesen Text bezeichnet man als Textknoten.

1 In den Beispielen aufgeführte Umlaute sind wegen der besseren Lesbarkeit nicht in ihrem UTF-8 Encoding, sondern immer durch die betreffenden Zeichen der natürlichen Schriftsprache wiedergegeben.

### CDATA

Unter Character Data (CDATA) lassen sich beliebige Zeichenketten abspeichern, die beim Parsen des XML Dokumentes nicht weiter überprüft werden. Auf diese Weise lässt sich zum Beispiel ein nicht korrekter XML Textabschnitt in einem XML Dokument abspeichern:

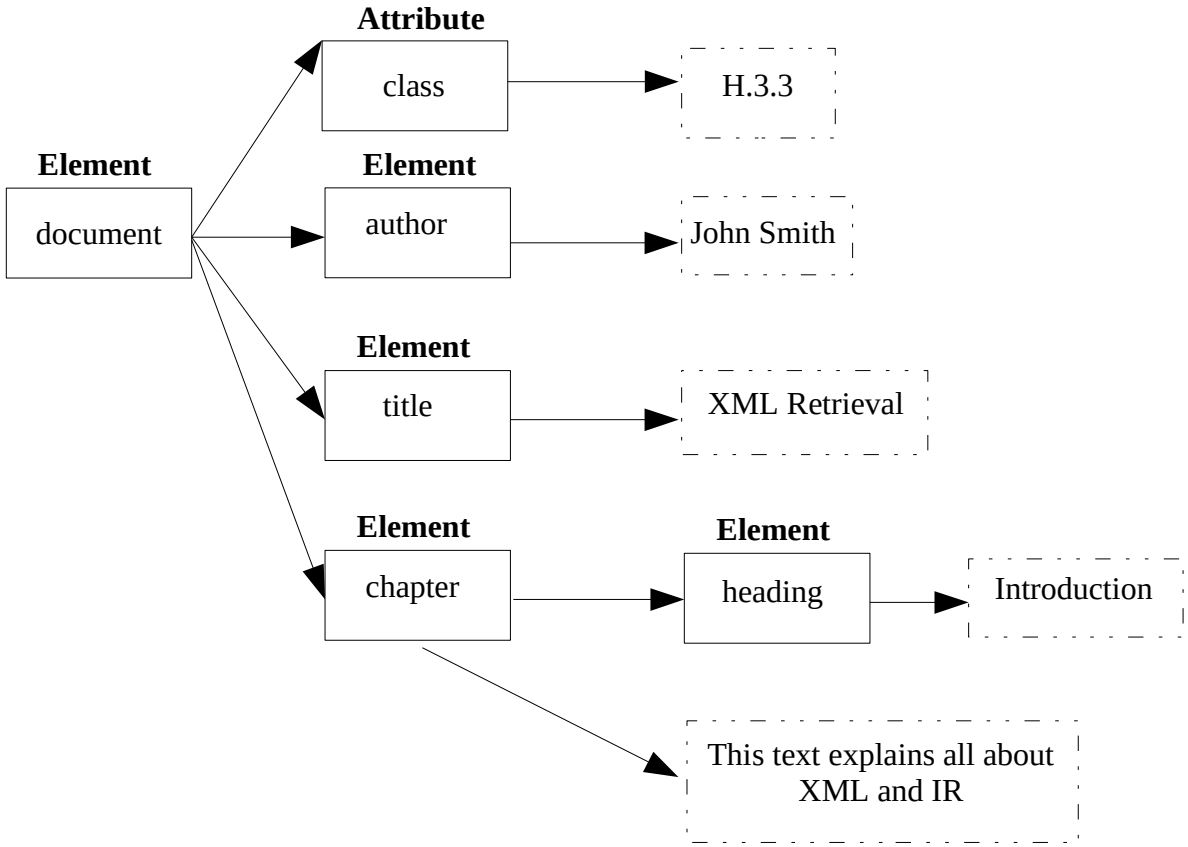
```
<oldHTML >
<![CDATA[
        <html>
        <p>Introduction
        <br>This text explains <BR>
        all about XML and IR.
        <HTML>
]]>
</oldHTML >
```

## Baumstruktur

Jedes XML Dokument lässt sich als Baum darstellen, dessen Knoten den im letzten Kapitel vorgestellten Typen entsprechen.

```
<document class="H.3.3">
    <author>John Smith</author>
    <title>XML Retrieval</title>
    <chapter>
        <heading>Introduction</heading>
        This text explains all about XML and IR.
    </chapter>
</document>
```

Die Baumstruktur dieses XML Dokumentes sieht wie folgt aus:




# Validität

## Well-Formed versus Valid

Bislang ist nur die grobe syntaktische Struktur von XML Dokumenten thematisiert worden. Dazu gehört etwa die Unterscheidung zwischen Groß- und Kleinschreibung, die notwendige Baumstruktur der Tags usw. Entspricht ein XML Dokument den im letzten Abschnitt erläuterten Bedingungen, so bezeichnet man dieses Dokument als wohlgeformt (well-formed).

Dieser Korrektheitsbegriff reicht in vielen Fällen jedoch nicht aus, da er keine Informationen über die erlaubten Tags und ihre Beziehung zueinander repräsentieren kann. Aus diesem Grund gibt es in XML die Möglichkeit, die komplette syntaktische Struktur von XML Dokumenten in einer Document Type Definition (DTD) festzulegen. Für jedes XML Dokument ist es dann entscheidbar, ob es einer bestimmten DTD entspricht oder nicht. Zum Beispiel ist in der DTD von XHTML genau festgelegt, welche Tags für die Erstellung eines XHTML Dokumentes benutzt werden dürfen. Die Gültigkeit (validity) eines XML Dokumentes bezüglich einer DTD schließt die Wohlgeformtheit des Dokumentes ein. Die Umkehrung gilt im Allgemeinen nicht.

## Document Type Definition

Im Folgenden wird die Grundfunktionalität einer DTD anhand eines Beispiels demonstriert. Angenommen, man möchte XML Dokumente für die Speicherung von Adressen verwenden.

```
<?xml version="1.0" ?>
<adresse>
        <name>Karl Mustermann</name>
        <strasse>Musterweg 15</strasse>
        <ort>Berlin</ort>
</adresse>
```

Eine DTD für diesen Dokumenttyp wird wie folgt definiert:

```
<!DOCTYPE adressen [
<!ELEMENT adresse (name, strasse, ort)>
        <!ELEMENT name #PCDATA>
        <!ELEMENT strasse #PCDATA>
        <!ELEMENT ort #PCDATA>
]>
```

Hier wird festgelegt, dass das Wurzelelement `adresse` vom Dokumenttyp `adressen` die Kindelemente `name`, `strasse` und `ort` besitzt. Die DTD lässt sich direkt in das XML Dokument mit einbinden.

```
<?xml version="1.0" ?>
<!DOCTYPE adressen [
<!ELEMENT adresse (name, strasse, ort)>
        <!ELEMENT name #PCDATA>
        <!ELEMENT strasse #PCDATA>
        <!ELEMENT ort #PCDATA>
]>

<adresse>
        <name>Karl Mustermann</name>
        <strasse>Musterweg 15</strasse>
        <ort>Berlin</ort>
</adresse>
```

Alternativ kann die DTD in einer externen Datei abgelegt und in das Dokument über einen Verweis eingebunden werden.

```
<?xml version="1.0" ?>
<!DOCTYPE adressen SYSTEM="adressen.dtd">

<adresse>
        <name>Karl Mustermann</name>
        <strasse>Musterweg 15</strasse>
        <ort>Berlin</ort>
</adresse>
```

Dieses Beispiel sollte nur einen kleinen Einblick in die Erstellung von DTDs geben. Da bezüglich RDF die Wohlgeformtheit von Dokumenten das entscheidende Kriterium ist, wird im Folgenden nicht weiter auf DTDs eingegangen.

## Namespaces

Ein Problem mit Namensräumen ergibt sich immer dann, wenn in einem XML Dokument Tags aus verschiedenen Communities benutzt werden sollen. Zum Beispiel könnte man eine Dublin Core Metadatenbeschreibung sehr einfach als XML Dokument repräsentieren:

```
<?xml version="1.0" ?>
<metadata>
        <creator>Karl Mustermann</creator>
        <title>XML</title>
</metadata>
```

Um auszudrücken, wo diese Tags definiert wurden, kann man einen Namespace angeben. Ein Namespace beschreibt eine Menge von Elementen und Attributen, die einer URI zugeordnet sind.

```
<?xml version="1.0" ?>
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:creator>Karl Mustermann</dc:creator>
        <dc:title>XML</dc:title>
</metadata>
```

Nun lassen sich auch andere Tags für die Beschreibung von Metadaten nutzen. Zum Beispiel könnte man den Namen des Creators in Vor- und Nachname auflösen, indem man vCard Elemente in die Beschreibung aufnimmt.

```
<?xml version="1.0" ?>
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:vCard="http://imc.org/vCard/3.0#">
        <dc:creator>
                <vCard:Given>Karl</vCard:Given>
                <vCard:Family>Mustermann</vCard:Family>
        </dc:creator>
        <dc:title>XML</dc:title>
</metadata>
```

Durch den Einsatz von Namespaces lassen sich Tags benutzen, die in verschiedenen Communities definiert wurden. Offensichtlich führt ein solcher Mix von Tags im Allgemeinen nicht zu einem bezüglich einer DTD gültigen XML Dokument. Die beschreibende Dokumentation der Komposition von Tags aus den verschiedenen Namensräumen, angereichert um deren Verpflichtungsgrad und Kardinalität (Wiederholbarkeit) bezeichnet man als „Metadata Application Profile“.

In den Beispielen dieses Skriptes werden die folgenden URI für die nachfolgend zusammengefassten Namespace-Prefixes und URI verwendet:

| Prefix | URI |
|---|---|
| dc | purl.org/dc/elements/1.1/ |
| dcq | purl.org/dc/qualifiers/1.0/ |
| dct | purl.org/dc/terms/ |
| dctype | purl.org/dc/dcmitype/ |
| rdf | www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | www.w3.org/2000/01/rdf-schema# |
| vCard | imc.org/vCard/3.0# |

# Einführung in die XML Kodierung von RDF

## Einleitung

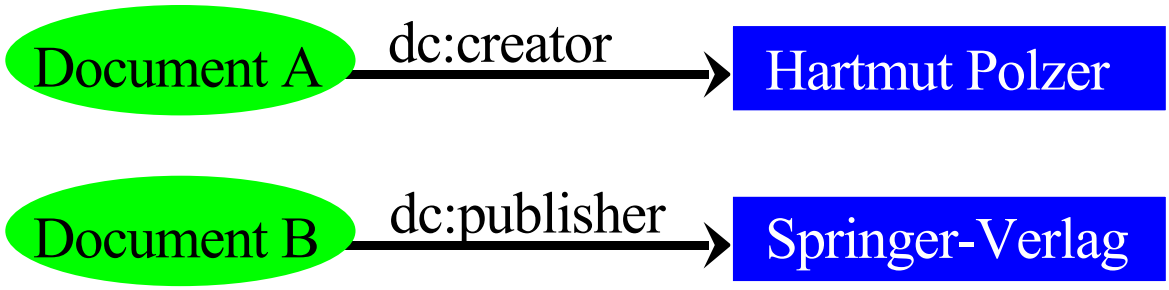
Vorausgesetzt wird im folgenden die Kenntnis, wie selbst komplexe Sachverhalte durch RDF-Graphen repräsentiert werden können. Diese Art der Repräsentation von RDF hat jedoch aufgrund der enthaltenen graphischen Elemente im Vergleich zu textorientierten Formaten deutliche Nachteile. Obwohl sich die Darstellung von Aussagen mit Hilfe von Graphen sehr intuitiv gestaltet, ist sie nur mit vergleichsweise großem Aufwand maschinenverstehbar.

Als Ausweg soll hier nun in eine weitere RDF-Repräsentationsform eingeführt werden: Neben Tripelschreibweise und Graphen-Darstellung lassen sich im RDF formulierte Aussagen auch innerhalb eines XML-Dokuments unterbringen.

Bei dieser Einführung soll anhand von Beispielen ausgehend von einfachen Statements schrittweise hin zu komplexen Strukturen (z.B. Reifizierung) vorgestellt werden, wie einzelne RDF-Konstrukte mit Hilfe der XML-Syntax darstellbar sind. Dabei werden auch einige alternative Darstellungsmöglichkeiten innerhalb der XML-Syntax diskutiert. Die hier gezeigten Möglichkeiten erheben keinen Anspruch auf Vollständigkeit, mit ihnen lassen sich jedoch sämtliche RDF-Konstruktionen realisieren.

## Back to the Roots: Einfache Statements

Um in RDF formulierte Aussagen in ein XML-Dokument einzubinden, muss man sich eines geeigneten Vokabulars bedienen. Zu diesem Zweck stehen jedem RDF-Parser zwei bekannte Namespaces zur Verfügung, die dennoch innerhalb des RDF-Abschnittes des XML-Dokumentes deklariert werden müssen: RDF und RDFS




```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/" >

  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_A">
       <dc:creator>Hartmut Polzer</dc:creator>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_B">
       <dc:publisher>Springer-Verlag</dc:publisher>
  </rdf:Description>

</rdf:RDF>
```

Die ersten drei Zeilen bilden zusammen mit der letzten gewissermaßen den Rahmen für die eigentliche Aussagen

„Hartmut Polzer ist der Autor von http://www.../Document_A."

und „Springer-Verlag ist der Publisher von http://www.../Document_B."

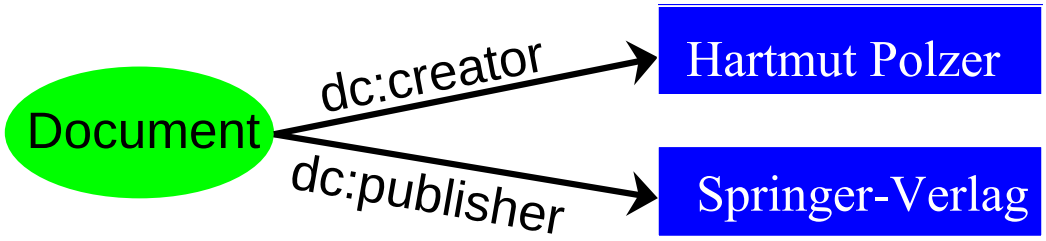
Durch die Anweisung `Description` wird jeweils ein neuer Knoten generiert, der im folgenden näher beschrieben wird. Das Attribut `about` ist optional. Fehlt es, wird eine anonyme Resource generiert, über die dann jedoch außerhalb des aktuellen Description-Blockes keine zusätzlichen Aussagen mehr gemacht werden können. Um sich dagegen auch außerhalb des Description-Blockes auf die neue Resource beziehen zu können, wird ihr durch das Attribut `ID="..."` explizit ein Name zugewiesen.

Die Prädikate `creator` und `publisher` werden dem Dublin-Core-Vokabular [DCMES] entnommen, das vorher im Kopf des RDF-Blockes als XML-Namespace deklariert wurde.

Für ein RDF-Dokument ist die Reihenfolge, in der Resourcen in einem XML-Dokument eingeführt werden, irrelevant. Die einzige Ausnahme von dieser Regel wird später bei der Container-Klasse „Sequence“ besprochen.

## Mehrere Statements beziehen sich auf dieselbe Resource

Beziehen sich mehrere Statements auf dieselbe Resource, gibt es verschiedene äquivalente Arten der XML-Kodierung:




Neben der im ersten Beispiel dargestellten Version, bei der sich nun beide Description-Blöcke durch den Parameter `rdf:about` auf dieselbe Resource beziehen würden, gibt es hier eine weitere Möglichkeit der Beschreibung:

```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
       <dc:creator>Hartmut Polzer</dc:creator>
       <dc:publisher>Springer-Verlag</dc:publisher>
</rdf:Description>
```

Wie oben gezeigt, lassen sich auch mehrere Statements in einem Description-Block unterbringen, wobei auch hier die Reihenfolge der einzelnen Zeilen keine Rolle spielt.

Bezieht man sich nicht auf eine externe Resource, sondern generiert eine eigene, gibt es ebenfalls verschieden äquivalente Schreibweisen:

a)

```
<rdf:Description rdf:ID="mein_Dokument">
       <dc:creator>Hartmut Polzer</dc:creator>
       <dc:publisher>Springer-Verlag</dc:publisher>
</rdf:Description>
```

b)

```
<rdf:Description rdf:ID="mein_Dokument">
       <dc:creator>Hartmut Polzer</dc:creator>
</rdf:Description>

<rdf:Description rdf:about="#mein_Dokument">
       <dc:publisher>Springer-Verlag</dc:publisher>
</rdf:Description>
```

c)

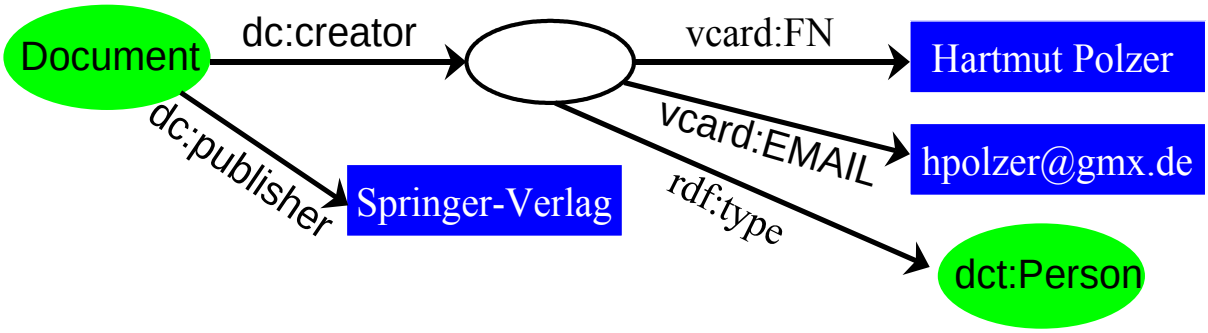
```
<rdf:Description rdf:ID="mein_Dokument"/>

<rdf:Description rdf:about="#mein_Dokument">
       <dc:creator>Hartmut Polzer</dc:creator>
       <dc:publisher>Springer-Verlag</dc:publisher>
</rdf:Description>
```

In a) wird die Verwendung des Attributes ID gezeigt. Hierbei werden über die neu generierte Resource, die den Namen *mein_Dokument* zugewiesen bekommt, zwei Aussagen gemacht. Ist eine Resource erst einmal mit einem Namen versehen worden, so lassen sich wie in b) auch außerhalb dieses Description-Blockes Aussagen über die Resource machen. Bei der Benutzung von `about` wird eine URI-Referenz erwartet, so dass durch das Voranstellen des Lattenkreuzes die nachfolgende ID als relativ zum aktuell verarbeiteten Dokument interpretiert wird und damit zusätzliche Angaben über die im ersten Description-Block mit *mein_Dokument* benannte Resource möglich werden. In c) ist einer Resource sogar zunächst lediglich ein Name zugewiesen worden. Würden anders als im Beispiel im folgenden keine Angaben über diese Resource gemacht, so hätte diese Zeile letztlich keine Wirkung.

## Bäume / Verschachtelungen

Über die Formulierung einfacher Statements hinaus lassen sich Statements auch verknüpfen.



XML-Realisierung:

```
<rdf:Description rdf:ID="Document">
    <dc:publisher>Springer-Verlag</dc:publisher>
    <dc:creator>
        <rdf:Description>
            <vCard:FN>Hartmut Polzer</vCard:FN>
            <vCard:EMAIL>hpolzer@gmx.de</vCard:EMAIL>
            <rdf:type
rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
        </rdf:Description>
    </dc:creator>
</rdf:Description>
```

Im Beispiel wird nach dem Verlag der Autor angegeben, der dann näher beschrieben wird. Hierzu wird mit Description eine neue (anonyme) Resource generiert, die die innerhalb dieses Description-Blockes aufgeführten Eigenschaften hat.

Nachdem bisher als Objekte nur Literals eingesetzt wurden, wird für die Aussage, dass es sich bei dem Autor um eine Person handelt, auf eine andere Resource verwiesen. Hierbei wird dem Prädikat stets das Attribut rdf:resource („rdf“ ist hierbei die Kennung für den im Kopf des RDF-Blockes vereinbarten RDF-Namespace, die auch im folgenden weiter verwendet wird) samt URI der entsprechenden Resource mitgegeben.

## Variationen in der Schreibweise

Selbst wenn der RDF-Graph eindeutig ist, kann es verschiedene äquivalente XML-Formulierungen für ihn geben. Über die Möglichkeit des Vertauschens einzelner Description-Blöcke und das Vertauschen einzelner Aussagen innerhalb eines Description-Blockes wurde bereits gesprochen. Darüber hinaus lässt sich häufig das folgende Fragment

```
<rdf:Description rdf:ID="Document">
    <dc:publisher>Springer-Verlag</dc:publisher>
    <dc:creator>
      <rdf:Description>
         <vCard:FN>Hartmut Polzer</vCard:FN>
         <vCard:EMAIL>hpolzer@gmx.de</vCard:EMAIL>
         <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
      <rdf:Description>
    </dc:creator>
</rdf:Description>
```

ersetzen durch

```
<rdf:Description rdf:ID="Document">
    <dc:publisher>Springer-Verlag</dc:publisher>
    <dc:creator rdf:parseType="Resource">

         <vCard:FN>Hartmut Polzer</vCard:FN>
         <vCard:EMAIL>hpolzer@gmx.de</vCard:EMAIL>
         <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>

    </dc:creator>
</rdf:Description>
```

Durch die Benutzung von `rdf:parseType="Resource"` als Attribut eines Prädikates wird der Parser explizit angewiesen, für das Objekt des Statements eine anonyme Resource zu erzeugen, für die die innerhalb des zugehörigen Prädikat-Blocks gemachten Aussagen gelten. Damit ergibt sich eine Möglichkeit, einen Knoten zu generieren, ohne `rdf:Description` zu schreiben. Ob das eleganter erscheint, bleibt jedem selbst überlassen. Allerdings ist dieses Konstrukt im Hinblick auf die Existenz von `Description` im Prinzip überflüssig und kann andererseits `Description` nicht überall ersetzen, weil z.B. das Attribut `about` hier nicht zugelassen ist.

## Content-Hiding

Mitunter wird es sinnvoll sein, RDF-Abschnitte innerhalb anderer Einheiten in ein XML-Dokument einzubetten. Insbesondere XHTML-Dokumente sind hierbei interessant.

Diese können dann von einer XML-Anwendung verarbeitet, also z.B. von einem Browser angezeigt werden, wobei der RDF-Abschnitt natürlich nicht dargestellt werden soll.

Hierbei taucht allerdings das Problem auf, dass gemäß der XHTML-Spezifikation eine XHTML-Anwendung den Inhalt von Tags, die sie nicht kennt, anzeigen MUSS. Als Resultat würden also im folgenden Beispiel die markierten Inhalte durch den Browser angezeigt.

```
<rdf:Description rdf:ID="Document">
    <dc:publisher>Springer-Verlag</dc:publisher>
    <dc:creator>
      <rdf:Description>
         <vCard:FN>Hartmut Polzer</vCard:FN>
         <vCard:EMAIL>hpolzer@gmx.de</vCard:EMAIL>
         <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
      <rdf:Description>
    </dc:creator>
</rdf:Description>
```

Um dieses Problem zu umgehen, müsste man sämtliche Inhalte innerhalb der Tags selber unterbringen.

Auch dieses lässt sich innerhalb der XML-Syntax realisieren.

Das folgende Beispiel ist in der sog. „Basic Abbreviated Syntax“ (vgl. [RDF-MSS] 2.2.2.) formuliert und mit dem oben gezeigten semantisch äquivalent:

```
<rdf:Description rdf:ID="Document"
     dc:publisher="Springer-Verlag">
    <dc:creator>
      <rdf:Description
             vCard:FN="Hartmut Polzer"
             vCard:EMAIL="hpolzer@gmx.de">
         <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
      </rdf:Description>
    </dc:creator>
</rdf:Description>
```

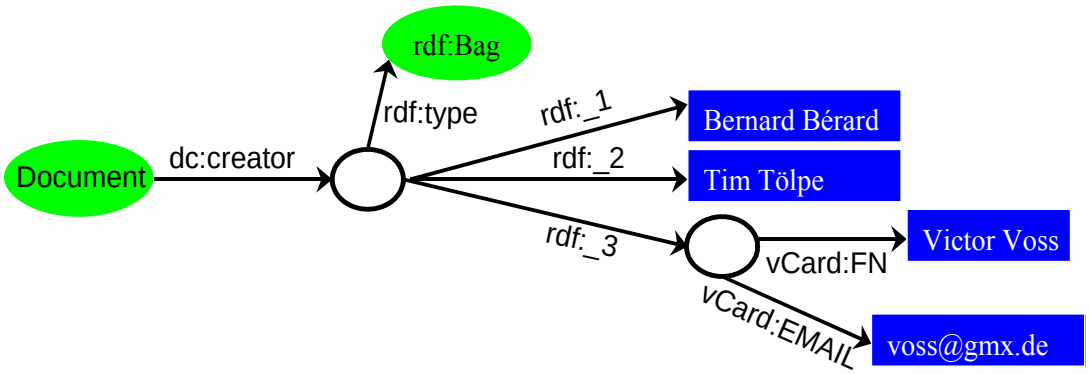
Bei dieser Syntax werden einzelne Prädikate als Attribute in die einzelnen Description-Tags mit hineingezogen. Dabei definiert der Name des Attributs das Prädikat, während das zugehörige Objekt, sofern es sich um ein Literal handelt, in Anführungszeichen angefügt wird. Hierbei ist zu beachten, dass alle Objekte, die Literals sind, bereits innerhalb des einleitenden Description-Tags behandelt werden müssen. Auf Objekte, die keine Literals sind, kann dagegen wie bisher innerhalb von Tags, die für einzelne Prädikate stehen, mit `rdf:resource` verwiesen werden. Andererseits muss man bedenken, dass sich nicht alle Konstruktionen in dieser kompakten „Basic Abbreviated Syntax“ formulieren lassen. Da die wiederholte Auflistung eines Attributes innerhalb eines Tags nicht zulässig ist, versagt diese Darstellung bei Resources, bei deren Beschreibung einzelne Prädikate wiederholt werden.

## Container

Nachdem die Formulierung einfacher und verschachtelter Statements sowie verschiedene mögliche Schreibweisen hierfür behandelt wurden, wird es im folgenden um Container gehen. Dabei werden die Beispiele aus Gründen der Übersichtlichkeit lediglich in der „Basic Serialization Syntax“ (vgl. [RDF-MSS] 2.2.1.) formuliert. Außerdem wird der umgebende RDF-Block weggelassen.

Container-Elemente (Bag, Seq, Alt) kommen z.B. zum Einsatz wenn die gemeinsame Autorenschaft zum Ausdruck gebracht oder auf verschiedene Zugangsmöglichkeiten hingewiesen werden soll.

Im folgenden Graph wird z.B. die gemeinsame Autorenschaft beschrieben, wobei die Reihenfolge der Autoren ausdrücklich keine Rolle spielen soll.




XML-Realisierung:

```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
    <dc:creator>
        <rdf:Bag>
            <rdf:li>Raoul B&#233;rard</rdf:li>
            <rdf:li>Tim T&#246;lpe</rdf:li>
            <rdf:li>
                <rdf:Description>
                  <vCard:FN>Viktor Voss</vCard:FN>
                  <vCard:EMAIL>voss@gmx.de</vCard:EMAIL>
                </rdf:Description>
            </rdf:li>
        </rdf:Bag>
    </dc:creator>
</rdf:Description>
```

Das zum Prädikat `dc:creator` gehörende Objekt ist vom Typ Bag. Durch den von `<rdf:Bag>` und `</rdf:Bag>` umschlossenen Block wird eine neue Resource vom Typ `Bag` generiert. Die Objekte dieses Bags werden dann nur noch aufgeführt. Die Aussage, die man über sie mittels `rdf:li` macht ist: Sie sind Objekte eines Containers (in diesem Fall eines Bags). Bei `rdf:li` handelt es sich wie bei `rdf:Description` um ein XML-Beschreibungselement für RDF und NICHT um ein Prädikat. Die „zugehörigen“ Prädikate `rdf:_1`, `rdf:_2` usw. finden sich in diesem Fall erst nach dem Parsen in den Tripeln. Durch die Reihenfolge der Auflistung der einzelnen Objekte werden dabei die konkreten Prädikate vergeben.

Bei der Benutzung von Containern ist die Reihenfolge der einzelnen `<rdf:li>`-Abschnitte also für die Vergabe der Prädikate sehr wohl von Bedeutung, auch wenn diese Reihenfolge semantisch letztlich nicht bei allen Containern eine Rolle spielt.

Im Zusammenhang mit der z.B. für Content-Hiding benutzten kompakten Schreibweise, bei der die Prädikate als Attribute in einzelne Tags mit hinein gezogen werden, tritt hier eine Besonderheit auf: Da `rdf:li` kein Prädikat ist und daher nicht aufgelistet werden kann, müssen die Attribute `rdf:_1="..." rdf:_2="..."` usw. ggf. wie im folgenden Beispiel explizit aufgeführt werden.

```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
    <dc:creator>
        <rdf:Bag rdf:_1="Raoul B&#233;rard" rdf:_2="Tim T&#246;lpe"/>
    </dc:creator>
</rdf:Description>
```

Allerdings ist in diesen Fällen, ein zusätzliches Aufführen von weiteren Objekten mittels `rdf:li` nicht zulässig. Aus diesem Grund lässt sich der obige Graph auf diese Weise nicht durch die kompakte Schreibweise ausdrücken, weil das dritte Objekt dann nicht beschrieben werden kann.

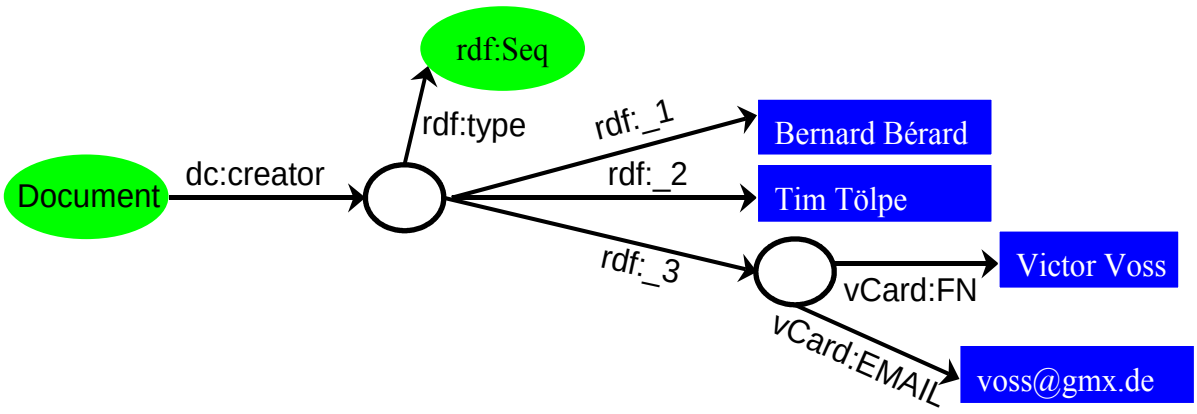
Die im Beispiel gezeigte Schreibweise `<rdf:Bag>` hat über die Container hinaus generelle Bedeutung. In RDF können in externen Schemata (vgl. [RDFS]) Klassen definiert werden, die gewissermaßen bestimmte Objekttypen repräsentieren. Die einzelnen Container stellen z.B. Klassen dar. Durch `<ns:KLASSE>` wird stets ein neuer Knoten generiert, von dem mittels `rdf:type` auf eine durch den Namespace `ns` definierte Resource verwiesen wird, die die jeweilige Klasse beschreibt.

Genau genommen handelt es sich bei dem oben besprochenen XML-Ausschnitt also um eine Kurzschreibweise für folgenden Ausschnitt, der den RDF-Graphen unmittelbarer widerspiegelt, und bei dem die Prädikate `rdf:_1`, `rdf:_2` usw. explizit vergeben werden. Darüber hinaus lassen sich damit auch die oben beschriebenen Eigenheiten bei der Benutzung der kompakten Schreibweise umgehen.

```
<rdf:Description about="http://www.iwi-iuk.org/Document">
  <dc:creator>
   <rdf:Description>
     <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Bag"
     <rdf:_1>Raoul B&#233;rard</rdf:_1>
     <rdf:_2>Tim T&#246;lpe</rdf:_2>
     <rdf:_3 rdf:Resource="http://www.voss.de"/>
   </rdf:Description>
  </dc:creator>
</rdf:Description>
```

An diesem Beispiel ist außerdem zu sehen, wie mit Sonderzeichen verfahren wird, die – wie auch in HTML möglich – als Entity ausgedrückt werden.

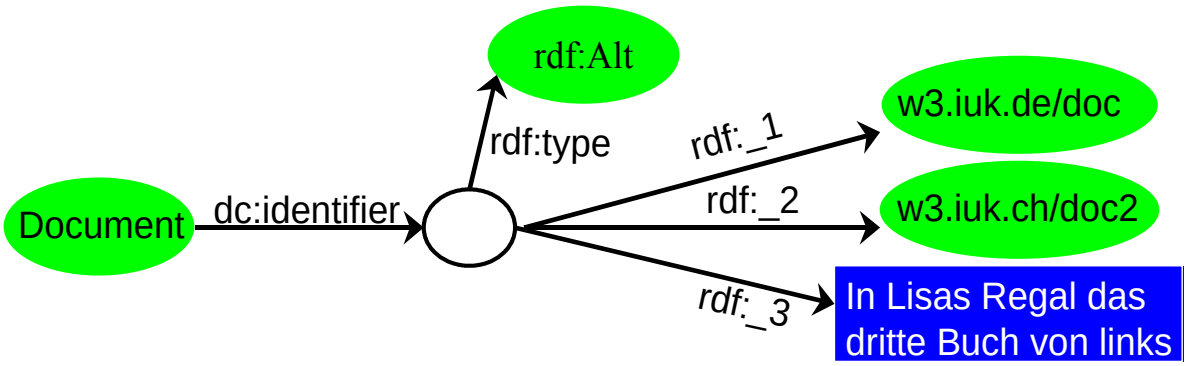
Die Beschreibung eines Containers vom Typ Sequence verläuft völlig analog. Falls die Reihenfolge der einzelnen Autoren ein Rolle spielt, ergibt sich z.B. folgender RDF-Graph:



Auch bei der XML-Syntax ändert sich nur die Container-Klasse:

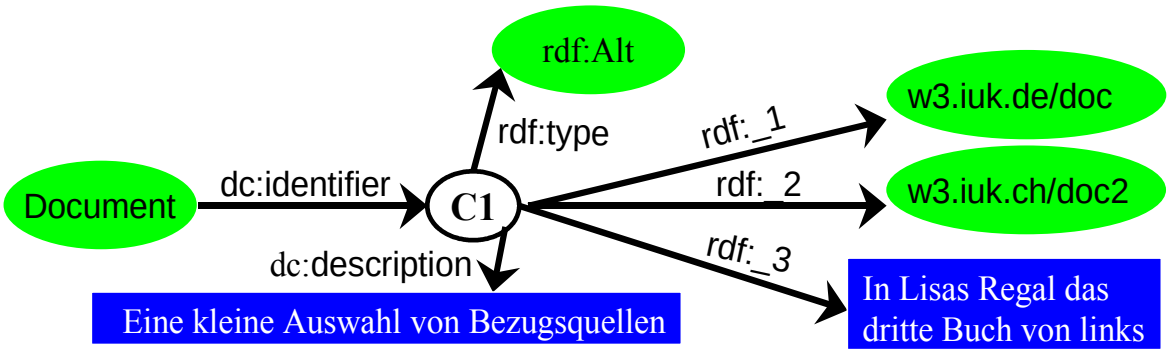
```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
    <dc:creator>
        <rdf:Seq>
            <rdf:li>Raoul B&#233;rard</rdf:li>
            <rdf:li>Tim T&#246;lpe</rdf:li>
            <rdf:li>
                <rdf:Description>
                  <vCard:FN>Viktor Voss</vCard:FN>
                  <vCard:EMAIL>voss@gmx.de</vCard:EMAIL>
                </rdf:Description>
            </rdf:li>
        </rdf:Seq>
    </dc:creator>
</rdf:Description>
```

Für die dritte und letzte Container-Klasse Alternative ändert sich syntaktisch ebenfalls nur die bereits angesprochene Kleinigkeit:




```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
    <dc:identifier>
        <rdf:Alt>
            <rdf:li rdf:resource="w3.iuk.de/doc"/>
            <rdf:li rdf:resource="w3.iuk.ch/doc2"/>
            <rdf:li>In Lisas Regal das dritte Buch von links</rdf:li>
        </rdf:Alt>
    </dc:identifier>
</rdf:Description>
```

Über die bisher besprochenen Möglichkeiten hinaus, lassen sich für Container auch IDs vergeben, um auch außerhalb des jeweils aktuellen Description-Blockes Aussagen über ihn bzw. die in ihm enthaltenen Objekte machen zu können. Den folgenden RDF-Graph erhält man beispielsweise mit dem darunter stehenden XML-Abschnitt.




```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document">
    <dc:identifier>
        <rdf:Alt rdf:ID="C1">
             <rdf:li rdf:resource="w3.iuk.de/doc"/>
             <rdf:li rdf:resource="w3.iuk.ch/doc2"/>
             <rdf:li>In Lisas Regal das dritte Buch von links</rdf:li>
        </rdf:Alt>
    </dc:identifier>
</rdf:Description>
<rdf:Description rdf:about="#C1">
    <dc:description>Eine kleine Auswahl von
Bezugsquellen</dc:description>
</rdf:Description>
```

Durch die Benutzung von `about` mit dieser Konstruktion werden keinerlei Aussagen über die Objekte in diesem Container gemacht; lediglich der Container bekommt eine weitere Eigenschaft.

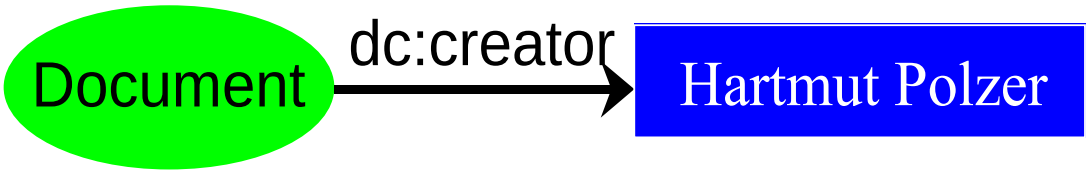
Mit diesem Beispiel soll das Kapitel Container abgeschlossen werden.

## Reifizierung

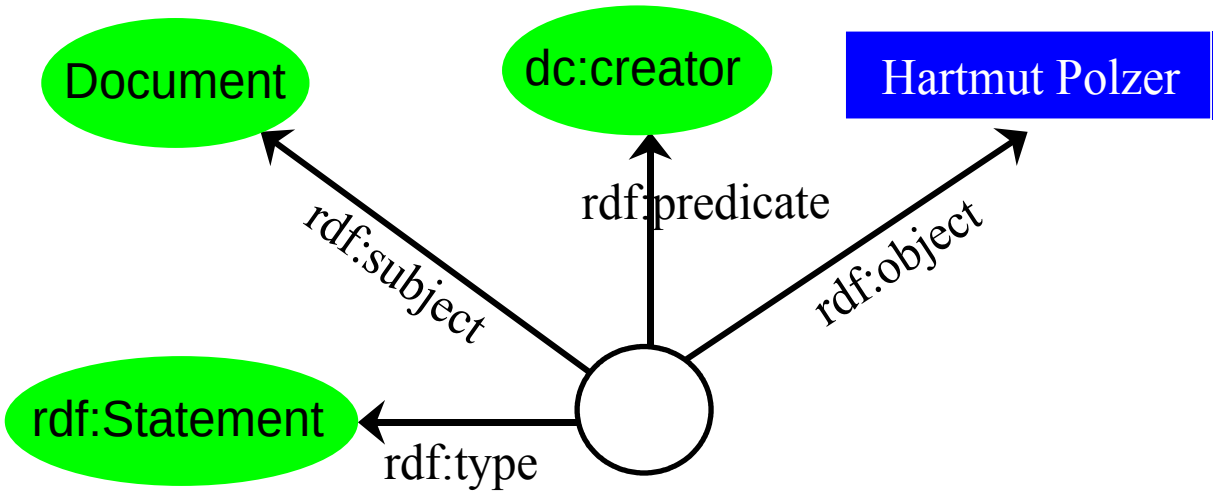
Eine Aussage lässt sich in RDF zum einen auf die herkömmliche Art mit einem Tripel beschreiben. In XML-Syntax könnte dies z.B. wie folgt aussehen.

```
<rdf:Description rdf:about="http://www.iwi-iuk.org/Document_B">
   <dc:creator>Hartmut Polzer</dc:creator>
</rdf:Description>
```

Diese Formulierung führt zu einem einzelnen Statement.




Unter Reifizierung eines Statements wird in RDF die Generierung einer neuen Resource vom Typ *Statement* und der die explizite Benennung von Subjekt, Prädikat und Objekt des Statements als solche verstanden. Dies würde zu folgendem RDF-Graph führen.




Außer den drei Statements, die die Bestandteile des ursprünglichen Statements bezeichnen, muss die neue Resource typisiert werden.

Hierzu wird eine Resource generiert die vom Typ Statement ist, und die außerdem noch folgende Eigenschaften hat: Subjekt ist Dokument, Prädikat ist dc:creator und Objekt ist das Literal „Hartmut Polzer“.

Ein XML-Ausschnitt, der den obigen Graphen ausdrückt, könnte z.B. wie folgt aussehen, wobei für die mit Description neu generierte Resource mit `ID=` natürlich auch ein Name vergeben werden kann:

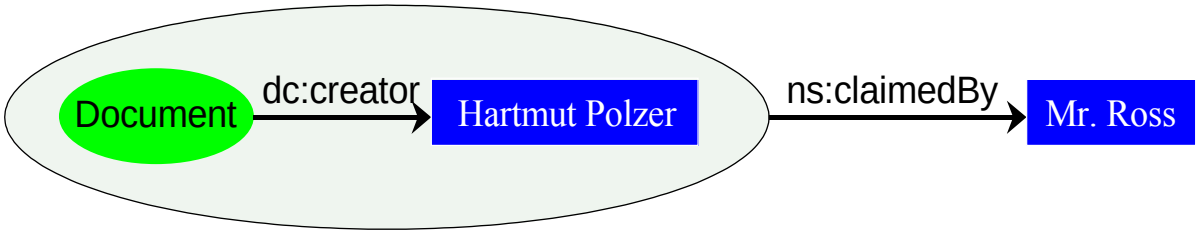
```
<rdf:Description>
 <rdf:type rdf:resource="http://www.w3.org/.../22-rdf-syntax-
ns#Statement"/>
 <rdf:subject rdf:resource="http://www.iuk.org/Dokument"/>
 <rdf:predicate
rdf:resource="http://purl.org/dc/elements/1.0/creator"/>
 <rdf:object>Hartmut Polzer</rdf:object>
</rdf:Description>
```

Da es sich bei `rdf:Statement` um eine Klasse handelt, kann man hier, wie bereits bei den Container-Klassen gesehen, auch eine kürzere Schreibweise benutzen:

```
<rdf:Statement>
  <rdf:subject rdf:resource="http://www.iuk.org/Dokument"/>
  <rdf:predicate
rdf:resource="http://purl.org/dc/elements/1.0/creator"/>
  <rdf:object>Hartmut Polzer</rdf:object>
</rdf:Statement>
```

Allerdings sieht die RDF-Spezifikation lediglich eine Kontrolle auf syntaktischer Ebene vor, so dass unvollständige Statements hier ebenso erlaubt sind wie Subjekt oder Prädikat vom Typ Literal oder sogar mehrere Prädikate, die von einer Resource ausgehen.

Aussagen höherer Ordnung: (Statements über Statements) lassen sich machen, indem man das ursprüngliche Statement als Subjekt oder Objekt weiterer Statements begreift.




In diesem Fall wird über die Resource vom Typ Statement also noch eine zusätzliche Aussage gemacht, die somit einfach zusätzlich in den Description-Block eingefügt werden kann.

```
<rdf:Description rdf:about="http://www.iuk.org/Dokument"/>
 <dc:creator>Hartmut Polzer</dc:creator>
</rdf:Description>

<rdf:Description ID="stat1">
 <rdf:type rdf:resource="http://www.w3.org/.../22-rdf-syntax-
ns#Statement"/>
 <rdf:subject rdf:resource="http://www.iuk.org/Dokument"/>
 <rdf:predicate
rdf:resource="http://purl.org/dc/elements/1.0/creator"/>
 <rdf:object>Hartmut Polzer</rdf:object>

  <ns:claimedBy>Mr. Ross</ns:claimedBy>
</rdf:Description>
```

Will man in einem anderen Description-Block Aussagen hinsichtlich des reifizierten Statements machen, muss man den für das Statement neu generierten Knoten durch Vergabe einer ID benennen. Im obigen Beispiel wäre das aber nicht nötig gewesen. Wichtig ist es dabei, zu unterscheiden, ob das ursprüngliche Statement, bereits eine Tatsache ist. In diesem Fall muss das zugehörige Tripel wie im Beispiel separat generiert werden.

# RDF Schema und Dublin Core

RDF [RDF-XSS] ermöglicht die gleichzeitige Nutzung unterschiedlicher Vokabulare für die Beschreibung von Metadaten.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="purl.org/dc/elements/1.1/"
         xmlns:vCard:="http://imc.org/vCard/3.0#">
  <rdf:Description rdf:ID="Document">
    <dc:creator>
      <rdf:Description>
        <vCard:FN>Hartmut Polzer</vCard:FN>
        <vCard:EMAIL>hartmut@gmx.de</vCard:EMAIL>
      </rdf:Description>
    </dc:creator>
  </rdf:Description>
</rdf:RDF>
```

Mit Hilfe von RDF Schema [RDFS] lassen sich

- Vokabulare in wohl definierter Weise spezifizieren und
- Relationen zwischen Vokabularen definieren.

Das RDF Schema des Dublin Core wird im folgenden ausführlich vorgestellt und es werden mögliche Erweiterungen bezüglich der Darstellung von Qualified Dublin Core thematisiert. Anhand dieser konkreten Beispiele werden die grundlegenden Konzeptionen von RDF Schema erläutert.

Als weiterführende und ergänzende Literatur empfehlen wir [DCRDF].

## Ein RDF Schema für Dublin Core

Der folgende Ausschnitt zeigt einen Teil des RDF Schema für Dublin Core:

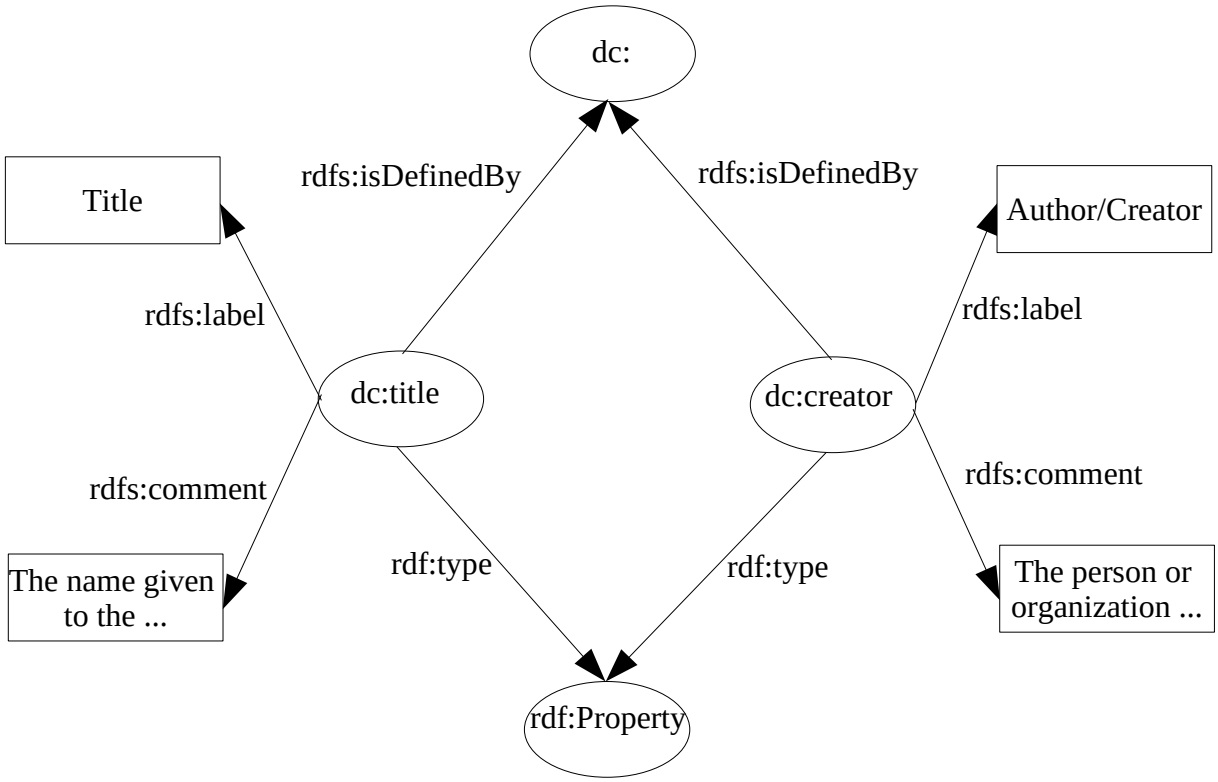
```
<? xml version='1.0'?>
  <rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:ID="title">
    <rdf:type rdf:resource="http://www.w3.org/TR/REC-rdf-
syntax#Property"/>
    <rdfs:label>Title</rdfs:label>
    <rdfs:comment>The name given to the resource, usually by the
Creator
    or Publisher.</rdfs:comment>
    <rdfs:isDefinedBy rdf:resource=""/>
  </rdf:Description>

  <rdf:Description rdf:ID="creator">
    <rdf:type rdf:resource="http://www.w3.org/TR/REC-rdf-
syntax#Property"/>
    <rdfs:label>Author/Creator</rdfs:label>
    <rdfs:comment>The person or organization primarily responsible for
    creating the intellectual content of the resource. For example,
    authors in the case of written documents, artists, photographers,
or
```

```
      illustrators in the case of visual resources.</rdfs:comment>
      <rdfs:isDefinedBy rdf:resource=""/>
    </rdf:Description>
...
</rdf:RDF>
```

(Hierbei bezeichnet `""` die URI des Dokumentes, in dem dieses RDF enthalten ist. Im obigen Fall ist das die URL `http://purl.org/dc/documents/rec-dces-19990702.htm`.)

Ein RDF Schema ist ein RDF Dokument, in dem spezielle Eigenschaften (aus dem `rdfs` Namespace) benutzt werden! Syntaktisch unterscheidet es sich damit überhaupt nicht von den RDF Dokumenten, die bislang vorgestellt wurden. Das RDF Graphenmodell zur obigen XML Repräsentation sieht wie folgt aus:




Bei der Spezifikation eines RDF Schema werden im wesentlichen Aussagen über Resources gemacht, die vom Typ `rdf:Property` und `rdfs:Class` sind. Die so definierten Properties können später zur (semantischen) Beschreibung von Daten verwendet werden. Welche speziellen Properties für die Spezifikation von neuen Vokabularen zur Verfügung stehen, ist im `rdfs` Namespace definiert. In dem RDF Schema des Dublin Core werden folgende Properties aus dem `rdfs` Namespace für die Beschreibung der Dublin Core Elemente benutzt:

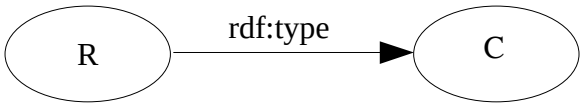
- `rdfs:label.` Ein Name bzw. eine Bezeichnung für die Resource.
- `rdfs:comment.` Eine menschenlesbare Beschreibung der Resource.
- `rdfs:isDefinedBy.` Ein Verweis auf eine Resource, die die zu beschreibende Resource definiert.

Für eine vollständige Auflistung der im `rdfs` Namespace definierten Properties sei auf die Spezifikation [RDFS] verwiesen. Im nächsten Abschnitt werden die grundlegenden Konzepte der RDF Schema Spezifikation vorgestellt.

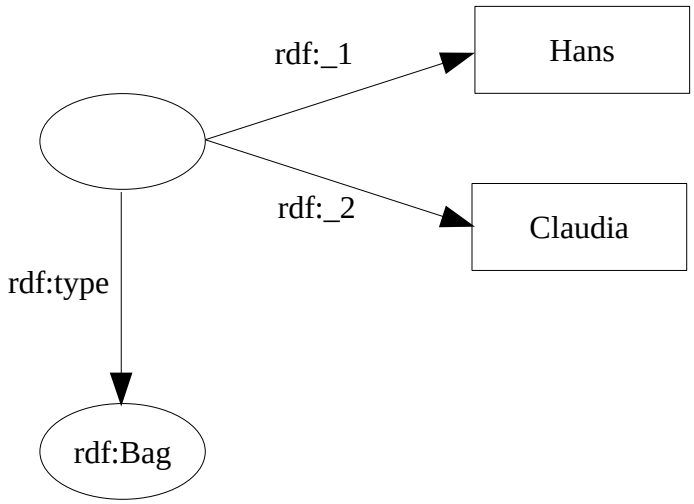
# Klassen und Eigenschaften

## Klassen

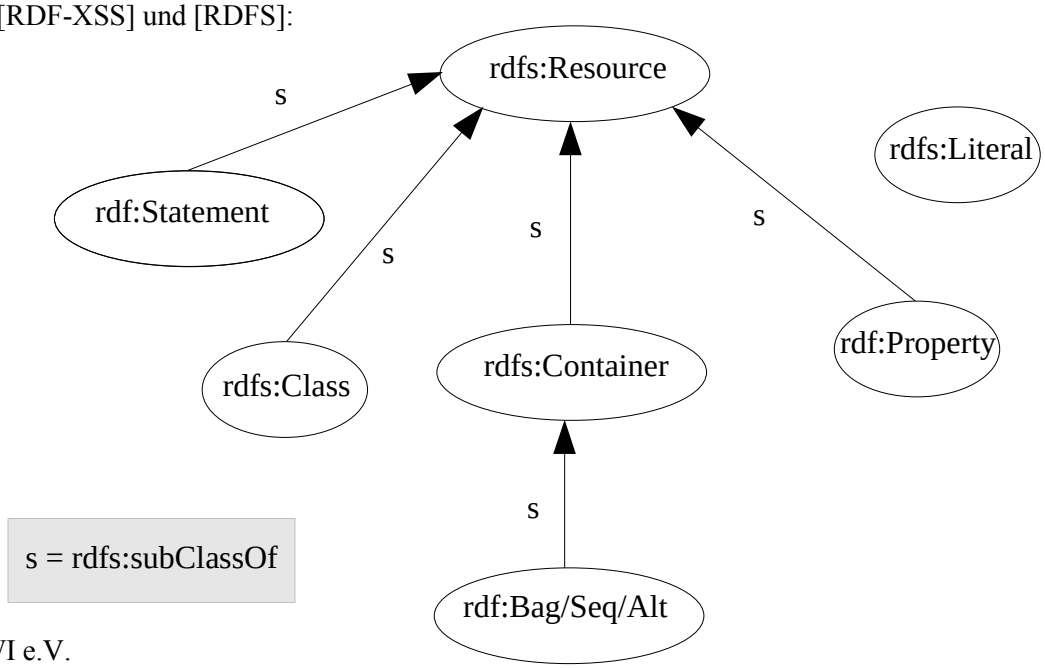
RDF Schema unterstützt die Definition von Klassen analog zu objektorientierten Programmiersprachen. Eine Klasse beschreibt eine Menge von Resources, die bestimmte Eigenschaften haben. Ist eine Resource R Element (oder Instanz) einer Klasse C, so wird dies durch den RDF Graphen




repräsentiert. Dabei ist die Klasse C selbst wieder durch eine Resource gegeben. Einige Klassen sind schon in der RDF Spezifikation definiert. Zum Beispiel ist ein Bag definiert als eine Resource, von der aus n Pfeile rdf:_1, ..., rdf:_n ($n \geq 1$) starten (natürlich können auch noch weitere Prädikate an dieser Resource starten):

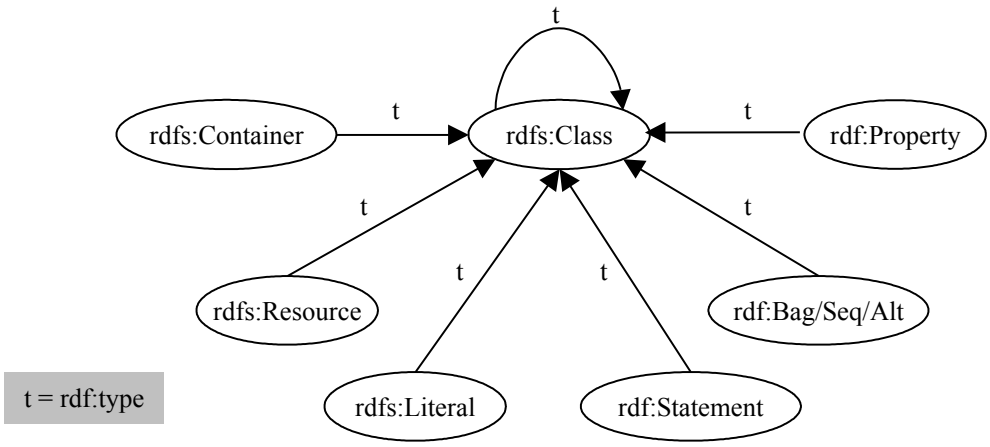



Eine Klasse A heißt Unterklasse einer Klasse B, wenn die Menge der durch A definierten Resources eine Teilmenge der Menge der durch B definierten Resources ist. Diese Relation zwischen Klassen wird durch die Property rdfs:subClassOf ausgedrückt. Der folgende RDF Graph zeigt einen Ausschnitt dieser Relation für Klassen aus [RDF-XSS] und [RDFS]:



- `rdfs:Literals.` Die Klasse der „Literals“, die in [RDF-MSS] definiert wurde. Da über Literals keine Aussagen gemacht werden dürfen, kann die Klasse der Literals keine Unterklasse einer der anderen Klassen sein. (Sonst wäre sie auch eine Unterklasse der Resources woraus sich ein Widerspruch ergeben würde.)
- `rdfs:Resource.` Die Klasse aller Objekte, über die mit Hilfe von RDF Aussagen gemacht werden [RDF-MSS]. Die Klasse der Resources ist Oberklasse von allen anderen Klassen, außer der Klasse der Literals. Das heißt jedes Statement, jede Klasse, jeder Container, jede Property ist eine Resource!
- `rdf:Statement.` Dies ist die Klasse der Statements, die in [RDF-MSS] definiert wurde. Eine Resource aus der Klasse `rdf:Statement` besitzt jeweils genau eine der folgenden drei Properties: `rdf:predicate`, `rdf:object` oder `rdf:subject`.
- `rdfs:Class.` Das Konzept von Klassen entspricht dem aus objektorientierten Programmiersprachen bekanntem Paradigma. Eine Resource aus der Klasse `rdfs:Class` ist dadurch charakterisiert, dass sie eine Property `rdf:type` besitzt, deren Wert `rdfs:Class` ist.
- `rdf:Property.` Die Klasse der Properties beschreibt diejenigen Resources, die zur Beschreibung von Resources benutzt werden.
- `rdfs:Container`, `rdf:Bag/Seq/Alt`. Die Klasse `rdfs:Container` ist die Oberklasse aller Container Objekte. Die unterschiedliche Semantik der drei Unterklassen Bag, Seq oder Alt ist in der RDF Spezifikation beschrieben.

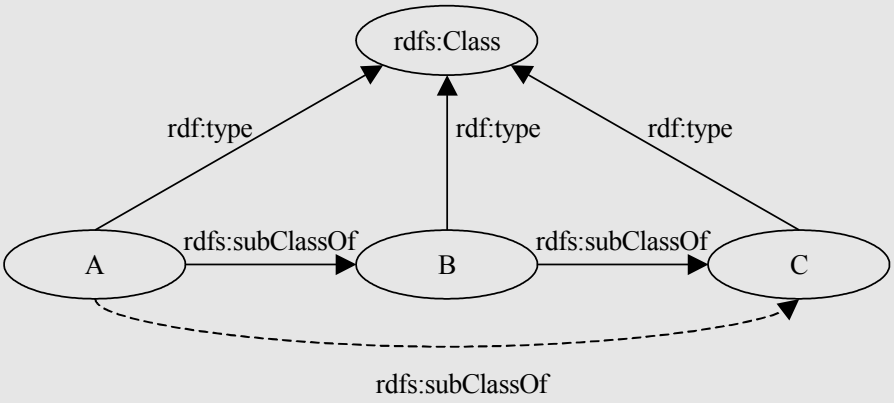
Bislang ist die durch die Property `rdfs:subClassOf` („Teilmenge von“) definierte Relation zwischen Klassen betrachtet worden. In der folgenden Grafik wird die Relation `rdf:type` („ist Element von“) repräsentiert:



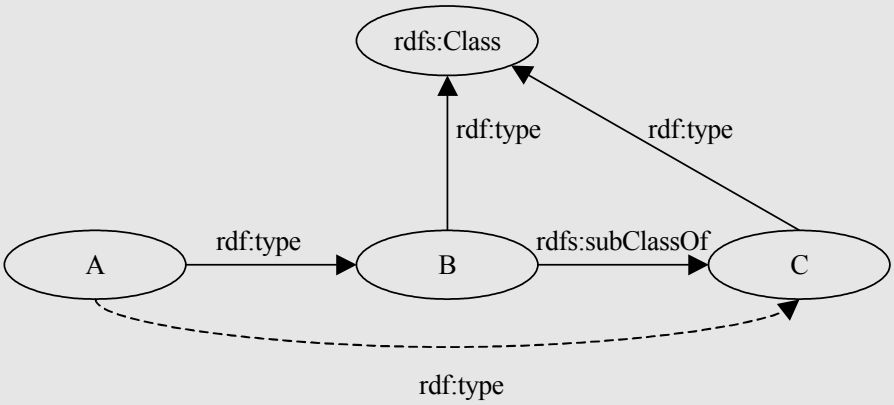

Das bedeutet: Jede Klasse ist Element der Klasse aller RDF Klassen! Insbesondere ist die Klasse aller Klassen eine Klasse.

### Theoretischer Exkurs

Konstruktionen dieser Art sind in der Mathematik wohlbekannt, auch wenn sie nicht immer einfach zu interpretieren sind. Man sollte sie als Axiome auffassen. Wichtiger sind die Schlussfolgerungen, die man aus ihnen ziehen kann. Zum Beispiel wird die Property `rdfs:subClassOf` als transitive Relation definiert, d.h. ist A Unterklasse von B und B Unterklasse von C, so ist A Unterklasse von C:




Auf diese Weise gewinnt man neue Relationen und somit neue Information. Ein weiteres Axiom ist das folgende: Ist A Element der Klasse B und B Unterklasse von C, so ist auch A Element aus C:




### Beispiel 1: Erstellung und Anwendung eines RDF-Schemas für Qualified Dublin Core

Die Dublin Core Initiative hat eine Liste von *Qualifiern* für die 15 Hauptelemente verabschiedet. Die *Qualifier* werden grob in zwei Klassen aufgeteilt: In die *Element Refinements* und die *Encoding Schemes*. Die *Element Refinements* beschreiben semantische Verfeinerungen der Hauptelemente, während die *Encoding Schemes* Vokabulare für die Beschreibung der Elementwerte zur Verfügung stellen (z.B. Klassifikationssysteme). Um diese *Qualifier* in RDF nutzen zu können, müssen sie (analog zu den Hauptelementen) innerhalb eines RDF Schema definiert sein. Im folgenden wird an dem Beispiel des Hauptelementes `dc:subject` gezeigt, wie eine solche Definition aussehen könnte. Dazu betrachte man die ursprünglich vorgesehenen *Qualifier* von `dc:subject`:

- Element Refinement: classification
- Encoding Schemes: LCSH, MESH, DDC, LCC, UDC.

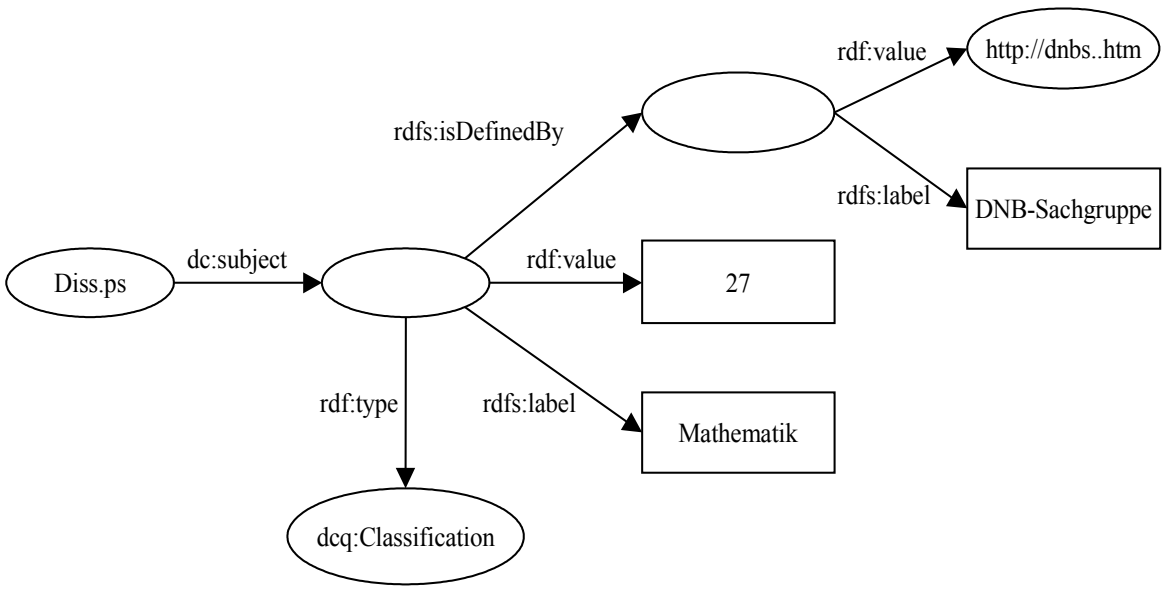
Es darf angezweifelt werden, ob `classification` wirklich eine semantische Verfeinerung von `dc:subject` ist. Schließlich handelt es sich ja immer noch um das „subject" des zu beschreibenden Objektes, nur die Art der Beschreibung hat sich geändert. Während vorher Schlüsselwörter verwendet wurden, wird nun ein Klassifikationssystem für die Beschreibung von „subject" benutzt. Somit hat sich nur der Typ der Beschreibung geändert, nicht die Beziehung zu dem zu beschreibenden Objekt.

Wahrscheinlich ist auf Grund dieser Problematik `classification` als Qualifier von `dc:subject` abgelehnt worden. Im folgenden wird gezeigt, wie sich dieses Problem mit Hilfe von RDF elegant lösen lässt.

Es liegt nahe, eine Klasse für den *Qualifier* `classification` zu definieren. Diese Klasse sollte folgende Werte repräsentieren können:

- Name der verwendeten Klassifikation
- URL der Klassifikation (falls vorhanden)
- (Aktueller) Wert
- Menschenverstehbare Formulierung des Wertes

Nehmen wir an, wir wollen eine Dissertation mit Hilfe der DNB-Sachgruppe klassifizieren. Der entsprechende RDF Graph könnte folgendermaßen aussehen:



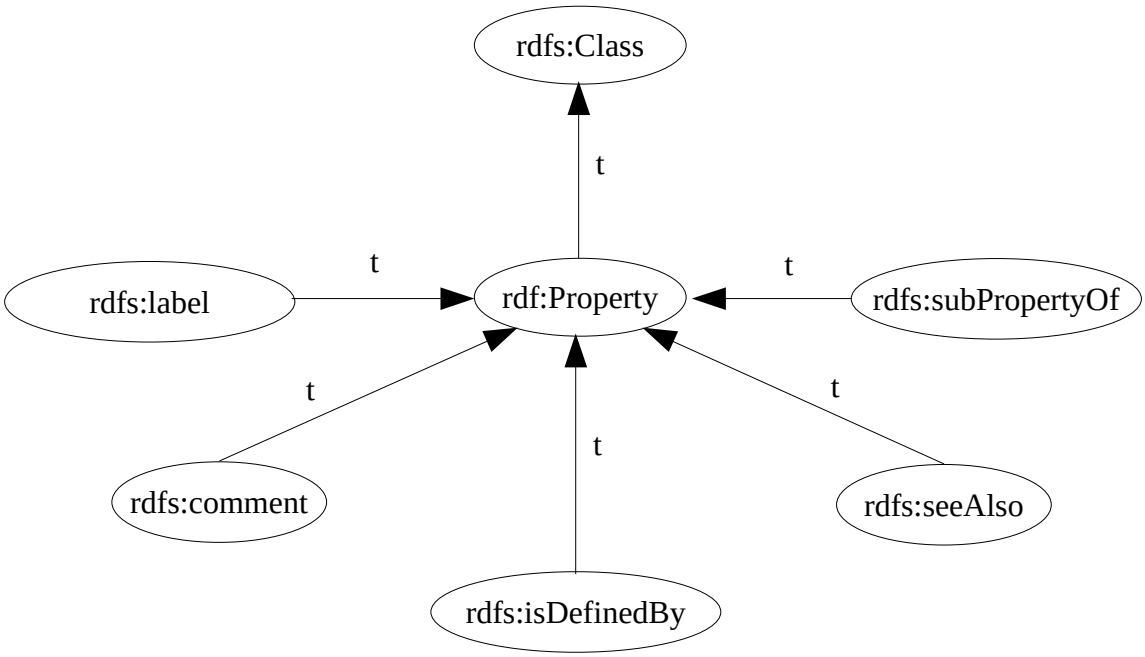

Dass eine Resource vom Typ `Classification` unabhängig von `dc:subject` ist (und somit keine semantische Verfeinerung ist), kann man sich folgendermaßen verdeutlichen. Angenommen, in dem (fiktiven) Namespace `ns` gibt es eine Property `ns:interests`, die die Interessengebiete eines Wissenschaftlers beschreiben soll. Ersetzt man nun im obigen Graphen die Dissertation Diss.ps durch eine Resource, die einen Wissenschaftler repräsentiert und die Property `dc:subject` durch `ns:interests`, so ergibt sich ein semantisch sinnvoller RDF Graph.

Der folgende RDF Ausschnitt skizziert eine mögliche Definition der Klasse `Classification` innerhalb des `dcq` (Dublin Core Qualifier) Namespaces:

```
<rdf:Description rdf:ID="Classification">
 <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
       <rdfs:label>Classification</rdfs:label>
       <rdfs:comment>Eine Resource vom Typ Klassifikation ist
       definiert durch ...</rdfs:comment>
       <rdfs:isDefinedBy rdf:resource=""/>
</rdf:Description>
```

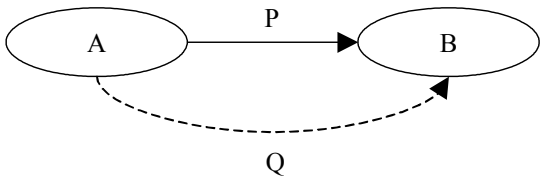
### Properties

Neben der Definition von Klassen ermöglicht RDF Schema natürlich die Definition von neuen Properties. Die folgende Grafik zeigt die wichtigsten Properties, die zur Definition von neuen Vokabularen benutzt werden können:




t = rdf:type

Mit `rdfs:isSubPropertyOf` wird (analog zu `rdfs:subClassOf` bei Klassen) eine transitive Relation definiert. Ist eine Property P `rdfs:subPropertyOf` einer Property Q, folgt aus dem Statement (A,P,B) das Statement (A,Q,B):



### Beispiel 2: Properties und Qualified Dublin Core

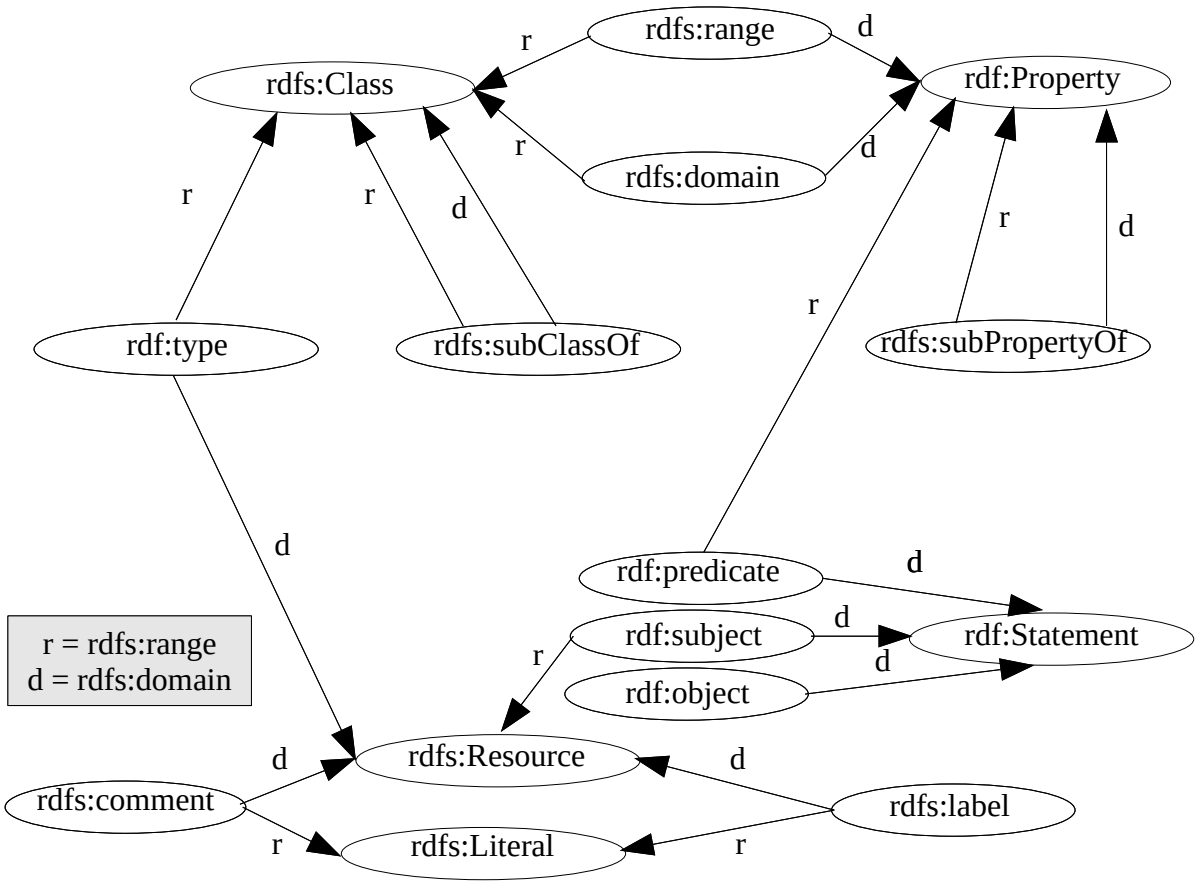
Mit Hilfe der Relation `rdfs:subPropertyOf` lässt sich genau die semantische Verfeinerung realisieren, die bei Qualifiern vom Typ *Element Refinement* benötigt wird. Man betrachte z.B. den Qualifier `created` vom Hauptelement `dc:date`. Der Qualifier `created` verfeinert die Semantik des Elementes `dc:date` in Bezug auf das zu beschreibende Dokument und kann deshalb in diesem Fall nicht durch eine neue Klasse angemessen repräsentiert werden. Das folgende RDF Fragment definiert die neue Property `dcq:created` als Subproperty von `dc:date`.

```
<rdf:Description rdf:ID="created">
    <rdf:type rdf:resource="http://www.w3.org/TR/REC-rdf
-syntax#Property"/>
    <rdfs:subPropertyOf rdf:resource="http://purl.org/dc/documents/
                rec-dces-19990702.htm#Date"/>
    <rdfs:label>Created</rdfs:label>
    <rdfs:comment>The date of creation of the resource.</rdfs:comment>
    <rdfs:isDefinedBy rdf:resource=""/>
</rdf:Description>
```

Alle Qualifier, die eine echte Verfeinerung eines Dublin Core Elementes darstellen, können auf diese Weise repräsentiert werden.

### Constraints

Bei der Definition von Properties innerhalb eines RDF Schema gibt es die Möglichkeit, den Werte- und den Zielbereich der Properties einzuschränken. Dazu stehen die beiden Properties `rdfs:domain` und `rdfs:range` zur Verfügung, die auf Resources vom Typ `rdfs:Class` zeigen können. Eine Property kann entweder keine oder genau eine `rdfs:range` Einschränkung besitzen. Eine Property kann beliebig viele `rdfs:domain` Einschränkungen besitzen. Sind überhaupt keine Einschränkungen angegeben, so bedeutet dies, dass die Property alle Resources als Quelle und alle Resources und Literals als Ziel haben kann. Die folgende Grafik (Quelle: [RDFSS]) zeigt die Einschränkungen der wichtigsten Properties aus der RDF und RDF Schema Spezifikation:



# RDF in XHTML

Zuerst stellen wir wieder Daten aus der RDF Spezifikation zusammen.

**RDF-MSS: Formal Grammar for RDF 1.0**

> *"When an RDF processor encounters an XML element or attribute name that is declared to be from a namespace whose name begins with the string 'http://www.w3.org/TR/rdf-primer/' and the processor does not recognize the semantics of that name then the processor is required to skip (i.e., generate no tuples for the entire XML element, including its content, whose name is unrecognizabed or that has an attribute whose name is unrecognized)"*

Nach Auskunft von J. Saarela, dem Autor von Sirpac und des Pro-solution Parsers, ist das Namespaceprefix ein Druckfehler und zu ersetzen durch „http://www.w3.org/1999/02/22-rdf-syntax-ns#". Dieser Namensraum ist bekannt und enthält z.B. das Element <html> nicht. Eine Konstruktion <rdf:html>HTML Text</rdf:html> in einem wohlgeformten RDF/XML Dokument ist somit für RDF unsichtbar.

**RDF-MSS 2.2.1 Basic Serialization Syntax**

> *"The RDF element is a simple wrapper that marks the boundaries in an XML document between which the content is explicitely intended to be mappable into an RDF data model instance. The RDF element is optional if the content can be known to be RDF from the application context."*

Der einzige uns bekannte Mechanismus den Kontext maschinenverstehbar klarzumachen ist die Signalisierung des Mimetypes. Für RDF ist dieser gemäß RFC 3023 „XML Media Types", z.B. unter http://www.rfc-editor.org/rfc/rfc3023.txt

- application/rdf+xml

**RDF-MSS 7.7 Content Hiding For RDF inside HTML**

> *"RDF, being wellformed XML, is suitable for direct inclusion in an HTML document when the user agent follows the HTML recommendations for handling in invalid documents. When a fragment of RDF is incorporated into an HTML document some browsers will render any exposed string content. Exposed string content is anything that appears between the '>' that ends one tag and the '<' that begins the next tag."*

Die Bemerkung „some browsers" verwandelt sich in „alle Browser", wenn man die XHTML Spezifikation zu Rate zieht:

> XHTML: 4. User Agent Conformance: *"If a user agent encounters an element it does not recognize, it must render the element's content."*

RDF und XHTML Ignoranz-Regeln sind also gegensätzlich zueinander.

Behauptet man gegenüber RDF falsche Dinge über den Inhalt seines Namensraumes, so wird das RDF nicht valide auswertbar sein. Dagegen kann man in XHTML sagen, was man will, jeder XHTML-Browser wird wenigstens den textuellen Inhalt darstellen.

- Es tritt hinzu, dass weder Netscape noch MS Internet-Explorer sich in jeder Hinsicht Spezifikations-konform verhalten.

Wir diskutieren nun am Beispiel mathematischer Preprints, wie sich unter dem Gesichtspunkt der Minimierung von Datenduplikation XHTML Dokumente erstellen lassen, die beiden Zwecken – Akzeptable Darstellung im Browser und korrekte Übermittlung von RDF MetaDaten – dienen.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Da die Standardkodierung gewählt wird, könnte die Angabe der Kodierung entfallen

```
<!-- created by MMM Version 3.0 -->
```

Kommentarzeile

```
<html xmlns="http://www.w3.org/1999/xhtml/">
```

Deklaration des XHTML Namespaces als default Namespace

```
<head><title>DC MetaData for: K-theory and generalized free products of
S-algebras: Localization methods</title>
</head><body bgcolor="#e7e7e7">
<h4><a href="ftp://ftp.mathematik.uni-
                osnabrueck.de/pub/sfb343/pr99XXX.ps.gz">
<em>K</em>-theory and generalized free products of S-algebras:
Localization methods</a></h4>
<p><a href="mailto:roland@mathematik.uni-osnabrueck.de">Roland
Schw&#228;nzl</a>,
<a href="mailto:ross@nmsu.edu">Ross Staffeldt</a>,
<a href="mailto:Friedhelm.Waldhausen@mathematik.uni-
bielefeld.de">Friedhelm Waldhausen</a></p>
<h4>Preprint Series</h4>
Diskrete Strukturen in der Mathematik, P99, SFB 343 Bielefeld
<h4>MSC2000</h4>
19D10 Algebraic $K$-Theory of spaces
<h4>Abstract</h4>
```

Bis zu dieser Stelle haben wir RDF keinen Anlass gegeben an ein RDF/XML Objekt zu denken.

```
<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:dc="http://purl.org/dc/elements/1.1/"
       xmlns:dct="http://purl.org/dc/terms/"
       xmlns:dcq="http://purl.org/dc/qualifier/1.0/"
       xmlns:mathnet="http://MathNet.preprints.org/SCHEME1.0#"
       xmlns:vCard="http://www.imc.org/vcard/3.0#">
```

Die URI für den MathNet-Namespace ist vorläufig, also noch nicht standardisiert.

```
<rdf:Description
       dc:subject="Stable K-Theory, Topological Hochschild Homology,
                Generalized free products, S-Algebras, Localization"
       dc:rights="Copyright: The author(s) agree, that this abstract
may be          stored as full text and distributed as such by
abstracting              services"
       dc:title="&lt;html
xmlns=&quot;http://www.w3.org/1999/xhtml&quot;
&gt;&lt;body&gt;&lt;em&gt;K&lt;/em&gt;-theory and
       generalized free products of S-algebras: Localization
                methods&lt;/body&gt;&lt;/html&gt;">
```

Der Titel wird als hidden content wiederholt. Da keine `ID/about` gesetzt wurde - in unserem Falle wird über ein Preprint als Werk geredet - kann diese Wiederholung nicht vermieden werden. XML RDF kann sonst keinen Kontext an den Parser weitergeben, der die Zuordnung für die anonyme Resource liefert.

Wir sind in einen großen Baum anonymer Resourcen eingetreten.

```
<dc:creator>
  <rdf:Bag>
```

Die Autoren werden in einen Bag (oder deutsch: Sack) gesteckt.

```
<rdf:li>
  <rdf:Description
    vCard:FN="Vorname Name"
    rdfs:label="Vorname Name"
    vCard:EMAIL="emailAnschrift">
```

Literal values werden vor XHTML versteckt. Die teilweise Wiederholung ist wieder dem Umstand zu verdanken, dass die genannte Person nicht über eine URI identifizierbar ist.

```
<rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
```

Lösung des Person / Organisation Problem aus dem ersten Kapitel: Die betrachtete Resource ist ein Element der Menge der Personen. In der nächsten Version des Profiles wird ein `<dcq:isReferencedBy rdf:resource="URL der (professional) homePage"/>` eingeführt.

Mit der in seiner die, in der Graphendarstellung die Pfeilrichtung invertierenden Bedeutung kann man die Schwierigkeit umgehen, dass anonyme Resourcen nur nach Benennung Objekte mehrerer Eigenschaften sein können.

```
<vCard:N>
  <rdf:Description vCard:Family="Nachname" vCard:Given="Vorname"/>
</vCard:N>
```

An dieser Stelle könnte die Erzeugung einer unbenannten Resource nur vermieden werden, wenn es eine URI für den unter vCard:N in Vor- und Zunamen zerlegten Namen gäbe.

```
</rdf:Description>
```

Ende des Kontexts für die unbenannte Resource, die die Person repräsentiert.

```
</rdf:li>
```

Ende des Kontexts für das erste Listenobjekt.

```
    <rdf:li>
      <rdf:Description
           vCard:FN="Ross Staffeldt"
           rdfs:label="Ross Staffeldt"
           vCard:EMAIL="ross@nmsu.edu">
 <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
           <vCard:N>
           <rdf:Description vCard:Family="Staffeldt"
                           vCard:Given="Ross"/>
           </vCard:N>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description
           vCard:FN="Friedhelm Waldhausen"
```

```
            rdfs:label="Friedhelm Waldhausen"
            vCard:EMAIL="Friedhelm.Waldhausen@mathematik.uni-
                                                    bielefeld.de">
 <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
      <vCard:N>
        <rdf:Description vCard:Family="Waldhausen"
                         vCard:Given="Friedhelm"/>
      </vCard:N>
      </rdf:Description>
    </rdf:li>
  </rdf:Bag>
```

Ende des Kontexts für den unbenannten Container, der die Koautoren hält.

```
</dc:creator>
```

Ende des creator Kontexts.

```
<dcq:created>
  <rdf:Description rdf:value="1999-07-22" rdfs:label="22 July 1999">
    <rdfs:isDefinedBy
rdf:resource="http://www.iso.ch/markete/8601.pdf"/>
    <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Point"/>
  </rdf:Description>
</dcq:created>
```

Implizit wird hier `dcq` als RDF Scheme angenommen, das `dcq:created` als subProperty von `dc:date` deklariert.

```
<dc:language>
  <rdf:Description rdf:value="ENG" rdfs:label="English">
         <rdfs:isDefinedBy
             rdf:resource="http://lcweb.loc.gov/standards/iso639-2/B"/>
  </rdf:Description>
</dc:language>
```

Hier wird die Sprache des beschriebenen Objektes angegeben, nicht hingegen die Sprache der Metadatenbeschreibung. Diese würde hingegen in dem `xml:lang` Attribut beschrieben.

```
<dcq:modified>
  <rdf:Description rdf:value="2000-02-29" rdfs:label="29 February
1999">
    <rdfs:isDefinedBy rdf:resource="http://iso.org/ISO8601"/>
    <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Point"/>
  </rdf:Description>
</dcq:modified>
```

Das Zielobjekt von `dc:identifier` wird als Alt Container angesteuert. Bezüglich der beschriebenen Resource von der Art eines Werkes sind diese Identifier alle gleich gut geeignet. Das bedeutet nicht, dass sich die angesteuerten Resourcen nicht in anderer Hinsicht unterscheiden könnten.

```
<dc:identifier>
   <rdf:Alt>
     <rdf:li rdf:resource="ftp://ftp.mathematik.uni-
osnabrueck.de/pub/sfb343/pr99XXX.ps.gz" />
```

```
      <rdf:li>
       <rdf:Description rdfs:label="Diskrete Strukturen in der
Mathematik, P99,SFB 343 Bielefeld"/>
      </rdf:li>
   </rdf:Alt>
</dc:identifier>
```

Am Werkknoten wird eine weitere `dc:subject` Property angehängt. Diesmal eine sehr qualifizierte Charakterisierung des Gegenstandes.

```
<dc:subject>
    <rdf:Description rdf:value="19D10"
                    rdfs:label="Algebraic $K$-theory of spaces">
       <rdf:type
           rdf:resource="http://purl.org/dc/terms/1.0/Classification"/>
```

Wir können klar zwischen dem Code in `rdf:value` und der menschenlesbaren Version in `rdfs:label` unterscheiden.

```
      <rdfs:isDefinedBy>
         <rdf:Description rdfs:label="Mathematical Subject Classification
                          Scheme (MSC2000)">
           <rdf:value rdf:resource="http://www.ams.org/msc/"/>
         </rdf:Description>
      </rdfs:isDefinedBy>
```

Ursprünglich haben wir an dieser Stelle `dcq:SubjectScheme` benutzt. In der Tat ist dieses Sprachelement von `dcq` jedoch überflüssig. Seine Rolle übernimmt mit klarerer Definition das RDF Schema Element `rdfs:isDefinedBy`

```
    </rdf:Description>
</dc:subject>
```

Beginn des Abstracts

```
<dcq:abstract>
  <rdf:Description>
    <dc:language>
   <rdf:Description rdf:value="ENG" rdfs:label="English">
       <rdfs:isDefinedBy
             rdf:resource="http://lcweb.loc.gov/standards/iso639-2/B"/>
      </rdf:Description>
   </dc:language>
   <rdfs:label rdf:parseType="Literal">
```

XHTML bekommt wieder etwas zu tun. Für XHTML wäre es nicht nötig die Deklaration zu wiederholen. Damit eine Anwendung, die einem RDF Parser nachgeschaltet ist, noch in der Lage ist, die Tags korrekt zu interpretieren, ist es sinnvoll, Processing Instructions zu wiederholen.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body><p align="justify">A generalized free product diagram of S-
algebras is a generalization and stabilization of the diagram of group
rings arising from a Seifert-van Kampen situation.
Our eventual goal is to obtain a description of the algebraic K-theory
```

```
of the ``large'' algebra in a generalized free product diagram in terms
of the K-theories of the three smaller algebras. We first provide
foundational material on generalized free product diagrams of S-
algebras and associated categories of Mayer-Vietoris presentations. We
show that the categories of Mayer-Vietoris presentations are categories
with cofibrations, weak equivalences, and mapping cylinders. In
particular, the hypotheses of the ``generic fibration theorem'' of
Waldhausen (<em>Algebraic K-theory of spaces, Lecture Notes in Math.
1126(1985), 318-419</em>) are satisfied for two fundamental notions of
weak equivalence, and there is, therefore, a three-term fibration
sequence up to homotopy in which the K-theory of Mayer-Vietoris
presentations with respect to the fine notion of equivalence is
compared with the K-theory with respect to the coarse notion. The rest
of the paper is concerned with interpreting these K-theories in terms
of the K-theories of the algebras in the generalized free product
diagram.<p> The first interpretative result of the paper uses the
additivity theorem to identify the K-theory of Mayer-Vietoris
presentations with respect to fine equivalences with the product of the
K-theories of the three smaller algebras in a given generalized free
product diagram.
</p><p align="justify">The second interpretative result uses the
approximation theorem to identify the K-theory of Mayer-Vietoris
presentations with respect to coarse equivalences with the K-theory of
the generalized free product algebra.
In order to confirm the hypotheses of the approximation theorem, we
develop a localization tool for Mayer-Vietoris presentations which
resembles Bousfield's theory of localization of spectra with respect to
generalized homology theories.</p>
<p align="justify">Investigation of the third term in the fibration
sequence will be the subject of another work.</p>
</body>
</html>
```

Die Duplizierung der gesamten Daten für den Abstract wurde vermieden ohne zusätzliche ID's vergeben zu müssen. Die Verwaltung jeder zusätzlichen ID kann ein Problem darstellen. Will man ein gegebenes XML/RDF Objekt mit anderen zusammenfassen muss jedesmal geprüft werden, ob die in beiden Objekten vergebenen ID's verschieden sind. Ist dies nicht der Fall, müssen sie geändert werden, da sonst inkorrektes RDF entsteht. Von der Vergabe, von ID's der Art "creator" oder "abstract" muss dringend abgeraten werden.

```
    </rdfs:label>
  </rdf:Description>
</dcq:abstract>
```

Die Überschrift Notes wird vor RDF versteckt.

Außerhalb von `<rdf:RDF>` ist nach wie vor die XHTML Deklaration aktiv. Das `<h4>` Element kann also ohne weitere Erläuterung genutzt werden. XHTML ignoriert nur den `<rdf:html>` Tag, den es nicht kennen kann.

Dieser Trick war für das Wort „Abstract“ nicht nötig, da es vor Erreichen des `<rdf:RDF>` Blocks gefallen ist.

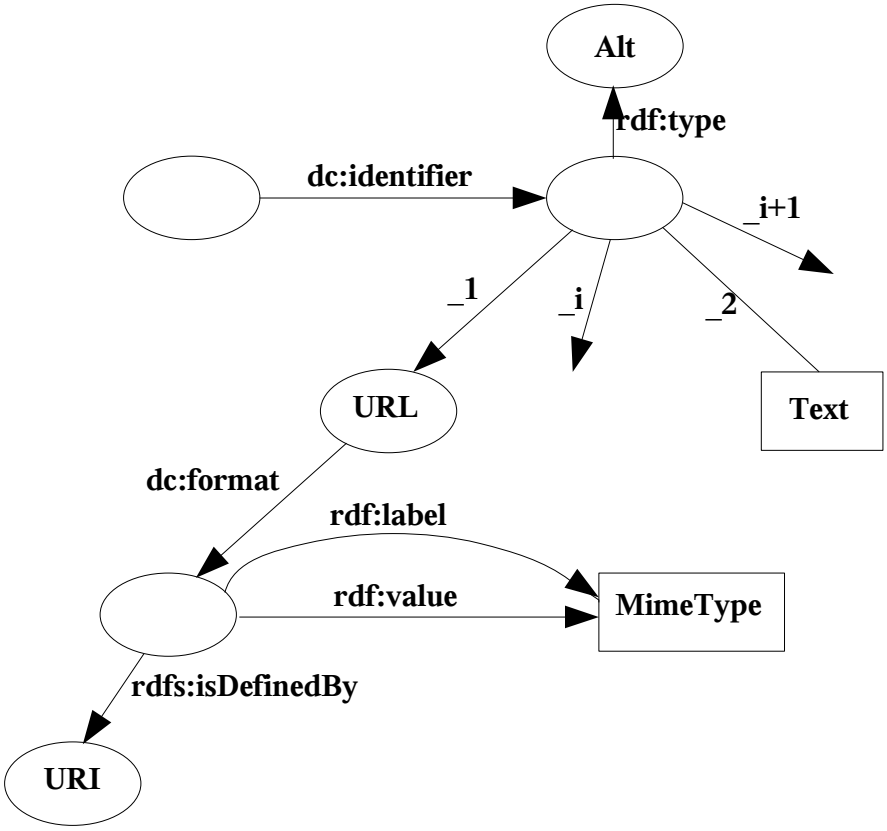
```
<rdf:html><h4>Notes</h4></rdf:html>
<dcq:references>
  <rdf:Description>
    <dc:language>
      <rdf:Description rdf:value="ENG"
```

```
                rdfs:label="English">
                <rdfs:isDefinedBy
            rdf:resource="http://lcweb.loc.gov/standards/iso639-2/B"/>
       </rdf:Description>
    </dc:language>
    <rdfs:label rdf:parseType="Literal">
      <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body>Öfter mal was neues!!
</body>
     </html>
    </rdfs:label>
   </rdf:Description>
</dcq:references>
</rdf:Description>
```

Den nachfolgenden Description Block hätte man auch in den Bereich der Identifier schreiben können. Da eine URI mit „about“ referenziert wird, ändert sich dabei nur die XML/RDF Darstellung und nicht das RDF Dokument. In der Graphendarstellung werden die nun folgenden Statements an der richtigen Stelle eingehängt.

```
<rdf:Description rdf:about="ftp://ftp.mathematik.uni-
         osnabrueck.de/pub/sfb343/pr99XXX.ps.gz">
  <dc:format>
    <rdf:Description rdf:value="application/postscript"
                     rdfs:label="Postscript Document">
        <rdfs:isDefinedBy
           rdf:resource="http://sunsite.auc.dk/RFC/rfc/rfc2046.html"/>
   </rdf:Description>
  </dc:format>
```

Ausschnitt aus der Graphendarstellung mit eingehängter Formatangabe:



```
</rdf:Description>
</rdf:RDF>
```

Der RDF Bereich wird formal geschlossen.

```
</body>
</html>
```

Der HTML Bereich wird formal geschlossen. Insgesamt haben wir wellformed XML: Alle geöffneten Tags sind wieder geschlossen worden. Keine Tags überkreuzten sich und das erste `<html>` Element fungiert als Wurzel für den XML Baum des Gesamtdokuments. Insgesamt erhalten wir wohlgeformtes XML. Eine gegen eine XHTML DTD validierende Form ist nicht möglich, da XML/RDF keine DTD besitzt und damit die DTD Mechanismen der XML1.0 Spezifikation versagen.

### Bibliothekarische Anwendungen:

**1:** Das für das DFG Projekt Dissertationen Online entwickelte RDF Profile lehnt sich eng an das für Preprints an. Bei Dissertationen gehen wir davon aus, dass sie anlässlich ihrer Archivierung eine URN oder einen anderen geeigneten Persistent Identifier erhalten werden. Diese kann dann sowohl als ID des rahmenden Description Blocks fungieren als auch als weiterer alternativer Identifier. Der Einbau als weiterer `dc:identifier` erlaubt es unter anderem, auf die Policy des URN Schemas zu verweisen.

**2:** Für bibliothekarisch archivierte Dissertationen bietet es sich an, die anonymen Resourcen, die Personen bezeichnen mit der URN des Name Authority Files in einer internen Datenbank zu füllen. Beim XML Datenexport in den außerbibliothekarischen Bereich brauchte dann lediglich dieser Eintrag gelöscht zu werden, um alle Personendatenschutzrechte zu respektieren. Administrative Daten können über das Setzen von bagID's angebunden werden.

**3:** Eine technisch fortgeschrittenere Lösung als die hier vorgestellte, wäre eine interne Datenhaltung in generischem XML, das über Content Negotiation dynamisch unter Nutzung von XSL (XML Style Language) XHTML bzw. reines RDF sendet. Unseres Wissens sind in der Praxis einsetzbare Lösungen dieser Art in der für RDF erforderlichen Flexibilität noch nicht verfügbar.

## Für einen RDF Parser relevanter Teil des obigen Dokuments

```
<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:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dct="http://purl.org/dc/terms/"
         xmlns:dcq="http://purl.org/dc/qualifier/1.0/"
         xmlns:mathnet="http://MathNet.preprints.org/SCHEME1.0#"
         xmlns:vCard="http://www.imc.org/vcard/3.0#">
<rdf:Description dc:subject="Stable K-Theory, Topological Hochschild
       Homology,Generalized free products, S-Algebras, Localization"
   dc:rights="Copyright: The author(s) agree, that this abstract may be
       stored as full text and distributed as such by abstracting
services"
   dc:title="&lt;html
        xmlns=&quot;http://www.w3.org/1999/xhtml&quot;&gt;&lt;body&gt;&
lt;em
       &gt;K&lt;/em&gt;-theory and generalized free products of S-
algebras:
       Localization methods&lt;/body&gt;&lt;/html&gt;">
<dc:creator>
  <rdf:Bag>
```

```
      <rdf:li>
        <rdf:Description
             vCard:FN="Vorname Nachname"
             rdfs:label="Vorname Nachname"
             vCard:EMAIL="emailAnschrift">
             <rdf:type
rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
             <vCard:N>
               <rdf:Description vCard:Family="Nachname"
                                                vCard:Given="Vorname"/>
             </vCard:N>
        </rdf:Description>
      </rdf:li>
      <rdf:li>
        <rdf:Description
             vCard:FN="Ross Staffeldt"
             rdfs:label="Ross Staffeldt"
             vCard:EMAIL="ross@nmsu.edu">
             <rdf:type
rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
             <vCard:N>
             <rdf:Description vCard:Family="Staffeldt"
vCard:Given="Ross"/>
             </vCard:N>
       </rdf:Description>
      </rdf:li>
      <rdf:li>
        <rdf:Description
             vCard:FN="Friedhelm Waldhausen"
             rdfs:label="Friedhelm Waldhausen"
             vCard:EMAIL="Friedhelm.Waldhausen@mathematik.uni-
bielefeld.de">
        <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Person"/>
        <vCard:N>
          <rdf:Description vCard:Family="Waldhausen"
                           vCard:Given="Friedhelm"/>
        </vCard:N>
        </rdf:Description>
      </rdf:li>
    </rdf:Bag>
</dc:creator>
<dc:language>
  <rdf:Description rdf:value="ENG" rdfs:label="English">
    <rdfs:isDefinedBy
               rdf:resource="http://lcweb.loc.gov/standards/iso639-
2/B"/>
  </rdf:Description>
</dc:language>
<dcq:created>
  <rdf:Description rdf:value="1999-07-22" rdfs:label="22 July 1999">
    <rdfs:isDefinedBy
rdf:resource="http://www.iso.ch/markete/8601.pdf"/>
    <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Point"/>
  </rdf:Description>
</dcq:created>
<dcq:modified>
  <rdf:Description rdf:value="2000-02-29" rdfs:label="29 February
1999">
    <rdfs:isDefinedBy rdf:resource="http://iso.org/ISO8601"/>
    <rdf:type rdf:resource="http://purl.org/dc/terms/1.0/Point"/>
```

```
  </rdf:Description>
</dcq:modified>
<dc:identifier>
   <rdf:Alt>
     <rdf:li rdf:resource="ftp://ftp.mathematik.uni-

osnabrueck.de/pub/sfb343/pr99XXX.ps.gz">
     </rdf:li>
     <rdf:li>
      <rdf:Description rdfs:label="Diskrete Strukturen in der
Mathematik,               P99, SFB 343 Bielefeld"/>
      </rdf:li>
   </rdf:Alt>
</dc:identifier>
<dc:subject>
  <rdf:Description rdf:value="19D10"
                   rdfs:label="Algebraic $K$-theory of spaces">
    <rdf:type
rdf:resource="http://purl.org/dc/terms/1.0/Classification"/>
    <rdfs:isDefinedBy>
       <rdf:Description rdfs:label="Mathematical Subject Classification
                                                                  Scheme
(MSC2000)">
         <rdf:value rdf:resource="weiss die URL nicht"/>
       </rdf:Description>
    </dcq:subjectScheme>
  </rdf:Description>
</dc:subject>
<dcq:abstract>
  <rdf:Description>
    <dc:language>
      <rdf:Description rdf:value="ENG" rdfs:label="English">
         <rdfs:isDefinedBy
               rdf:resource="http://lcweb.loc.gov/standards/iso639-
2/B"/>
      </rdf:Description>
   </dc:language>
   <rdfs:label rdf:parseType="Literal">
     <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body>A generalized free product diagram of S-algebras is a
generalization and stabilization of the diagram of group rings arising
from a Seifert-van Kampen situation.
Our eventual goal is to obtain a description of the algebraic K-theory
of the ``large'' algebra in a generalized free product diagram in terms
of the K-theories of the three smaller algebras. We first provide
foundational material on generalized free product diagrams of S-
algebras and associated categories of Mayer-Vietoris presentations. We
show that the categories of Mayer-Vietoris presentations are categories
with cofibrations, weak equivalences, and mapping cylinders. In
particular, the hypotheses of the ``generic fibration theorem'' of
Waldhausen (<em>Algebraic K-theory of spaces, Lecture Notes in Math.
1126(1985), 318-419)</em> are satisfied for two fundamental notions of
weak equivalence, and there is, therefore, a three-term fibration
sequence up to homotopy in which the K-theory of Mayer-Vietoris
presentations with respect to the fine notion of equivalence is
compared with the K-theory with respect to the coarse notion. The rest
of the paper is concerned with interpreting these K-theories in terms
of the K-theories of the algebras in the generalized free product
diagram.<p> The first interpretative result of the paper uses the
additivity theorem to identify the K-theory of Mayer-Vietoris
```

```
presentations with respect to fine equivalences with the product of the
K-theories of the three smaller algebras in a given generalized free
product diagram.
</p><p align="justify">The second interpretative result uses the
approximation theorem to identify the K-theory of Mayer-Vietoris
presentations with respect to coarse equivalences with the K-theory of
the generalized free product algebra. In order to confirm the
hypotheses of the approximation theorem, we develop a localization tool
for Mayer-Vietoris presentations which resembles Bousfield's theory of
localization of spectra with respect to generalized homology theories.
</p><p align="justify">Investigation of the third term in the fibration
sequence will be the subject of another work.</p>
</body>
      </html>
    </rdfs:label>
  </rdf:Description>
</dcq:abstract>
<dcq:notes>
   <rdf:Description>
    <dc:language>
      <rdf:Description rdf:value="ENG" rdfs:label="English">
         <rdfs:isDefinedBy
                 rdf:resource="http://lcweb.loc.gov/standards/iso639-
2/B"/>
      </rdf:Description>
   </dc:language>
   <rdfs:label rdf:parseType="Literal">
     <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body>Öfter mal was neues!!
</body>
     </html>
   </rdfs:label>
  </rdf:Description>
</dcq:notes>
</rdf:Description>
<rdf:Description
   rdf:about="ftp://ftp.mathematik.uni-
                              osnabrueck.de/pub/sfb343/pr99XXX.ps.gz">
  <dc:format>
    <rdf:Description rdf:value="application/postscript"
                     rdfs:label="Postscript Document">
    <rdfs:isDefinedBy
            rdf:resource="http://sunsite.auc.dk/RFC/rfc/rfc2046.html"/>
   </rdf:Description>
  </dc:format>
</rdf:Description>
</rdf:RDF>
```

## Visuelle Darstellung des XML Dokuments in einem Browser

**_K_-theory and generalized free products of S-algebras: Localization methods**

Vorname Nachname, Ross Staffeldt, Friedhelm Waldhausen

**Preprint Series**

Diskrete Strukturen in der Mathematik, P99, SFB 343 Bielefeld

**MSC2000**

19D10 Algebraic $K$-Theory of spaces

**Abstract**

A generalized free product diagram of S-algebras is a generalization and stabilization of the diagram of group rings arising from a Seifert-van Kampen situation. Our eventual goal is to obtain a description of the algebraic K-theory of the ``large'' algebra in a generalized free product diagram in terms of the K-theories of the three smaller algebras. We first provide foundational material on generalized free product diagrams of S-algebras and associated categories of Mayer-Vietoris presentations. We show that the categories of Mayer-Vietoris presentations are categories with cofibrations, weak equivalences, and mapping cylinders. In particular, the hypotheses of the ``generic fibration theorem'' of Waldhausen (*Algebraic K-theory of spaces, Lecture Notes in Math. 1126(1985), 318-419*) are satisfied for two fundamental notions of weak equivalence, and there is, therefore, a three-term fibration sequence up to homotopy in which the K-theory of Mayer-Vietoris presentations with respect to the fine notion of equivalence is compared with the K-theory with respect to the coarse notion. The rest of the paper is concerned with interpreting these K-theories in terms of the K-theories of the algebras in the generalized free product diagram.

The first interpretative result of the paper uses the additivity theorem to identify the K-theory of Mayer-Vietoris presentations with respect to fine equivalences with the product of the K-theories of the three smaller algebras in a given generalized free product diagram.

The second interpretative result uses the approximation theorem to identify the K-theory of Mayer-Vietoris presentations with respect to coarse equivalences with the K-theory of the generalized free product algebra. In order to confirm the hypotheses of the approximation theorem, we develop a localization tool for Mayer-Vietoris present ations which resembles Bousfield's theory of localization of spectra with respect to generalized homology theories.

Investigation of the third term in the fibration sequence will be the subject of another work.

**Notes**

Öfter mal was neues!!

# Speicherung von RDF in Datenbanken

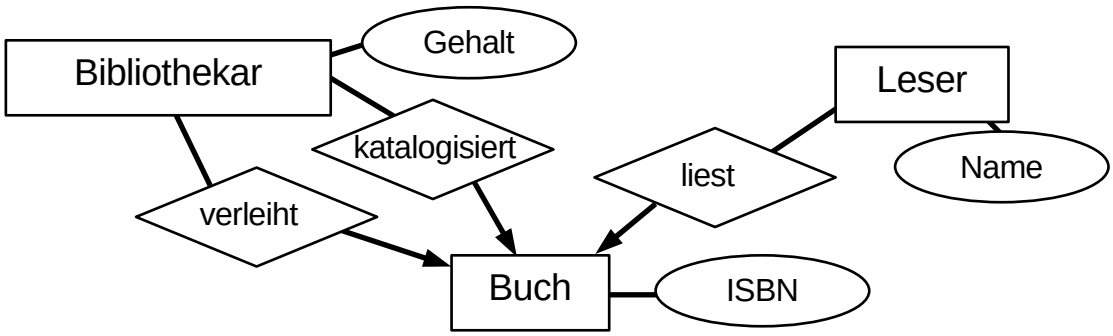
Um die spezielle Problematik, die RDF im Zusammenhang mit der Speicherung in Datenbanken verursacht, zu diskutieren, sollen zunächst die Grundzüge der Idee von Datenbanken wiederholt werden:

Es gibt viele Gründe Daten nicht in Dateien sondern in Datenbanken zu speichern. Einige davon sind die Vermeidung von Redundanz und damit verbundener Inkonsistenz. Werden z.B. Namen von Autoren zusammen mit jedem Buch, das sie geschrieben haben, gespeichert, wird einerseits unnötig Speicherplatz verbraucht. Andererseits kann es zu Unstimmigkeiten kommen, weil eine Korrektur des Namens nur in einigen aber nicht in allen Datensätzen vorgenommen wird.

Sollen Daten miteinander verknüpft werden, so müssen alle Dateien unter hohem Rechenaufwand durchsucht werden. Ein Beispiel dafür ist die Erstellung einer Liste von Büchern eines Autors.

Aus diesen komplizierten Zugriffen auf die Daten folgen dann hohe Entwicklungskosten für relativ einfache Operationen.

## Abstrakte Sicht auf Daten




Ein Modell zur abstrakten Darstellung von Daten ist das Entity-Relationship Modell. Einheiten oder Entitäten werden dabei als Rechtecke, Eigenschaften der Entitäten durch Ellipsen dargestellt.

Die Entitäten stehen zueinander in Beziehung. Diese Beziehungen werden durch Rauten gekennzeichnet. Das Modell erinnert sehr stark an RDF, obwohl es sehr viel älter ist.

Beziehungen oder Relationen können verschiedene Stelligkeit haben. Wir unterscheiden die 1:1-, 1:N-, N:1-, N:M-Relationen. Die 1:1-Relation begegnet uns in der Beziehung „verheiratet mit“, denn jede Ehefrau ist mit genau einem Ehemann verheiratet. Ein Beispiel für eine 1:N Relation ist, dass eine Bibliothek ein Buch aufstellt. Die Bibliothek hat diese Beziehung natürlich zu vielen Büchern, aber jedes Buch kann nur einer Bibliothek gehören. Damit ist auch intuitiv klar, was eine N:1 Beziehung ist. Darüber hinaus gibt es noch die N:M Beziehung, die im Bild in der Relation „lesen“ auftaucht. Die meisten Leser lesen viele Bücher und jedes Buch wird auch von vielen verschiedenen Lesern gelesen.

Aufbauend auf dem Entity-Relationship Modell sollen nun zwei Datenbank Architekturen vorgestellt werden, die Daten unter den Kriterien Redundanzfreiheit, Konsistenz, Sicherheit und Effizienz bei Speicherung und Zugriff aufbereiten. Es handelt sich dabei um relationale und objektorientierte Datenbanken.

# Relationale Datenbanken

Relationale Datenbanken sind die derzeit gängigste Datenbank Architektur (Oracle, DB2, Microsoft Access, mySQL,...), die durch SQL (Standard Query Language) weitgehend standardisiert sind. SQL geht zurück auf den von IBM Anfang der 70er Jahre entwickelten Prototyp System R mit der Anfragesprache Sequel. Theoretisch ist SQL eine Mischung aus einem prozeduralen (relationale Algebra) und deklarativen Ansatz (Relationenkalkül).

Die Idee bei den relationalen Datenbanken ist die Speicherung der Entitäten, Attribute und Relationen in verschiedenen Tabellen. Der Zugriff erlaubt die gleichzeitige Nutzung mehrerer Tabellen mit verschachtelten Bedingungen. Umgesetzt werden die Anfragen und die Eingabe der Daten durch SQL.

Anhand von Beispielen soll im folgenden erklärt werden, wie eine effektive Nutzung der Architektur möglich ist. Wir denken uns dabei die ISBN als einen eindeutigen Kennwert für Bücher (auch wenn dies in der Realität nicht immer der Fall ist).

### 1. Beispiel (schlecht)

| **Bücher** | | |
|---|---|---|
| **ISBN** | **Titel** | **Autor** |
| 12345 | Datenbanken | Erika Muster |
| 34567 | XML Spec | Erwin Mai |
| 78910 | Graf und Welt | Erika Muster, Erwin Mai |

Dieses Beispiel ist schlecht, weil die Anfrage nach `„Muster, Erwin“` ein Ergebnis liefert, obwohl es keinen Autor mit diesem Namen gibt. Die **erste Normierung** fordert deshalb, dass es keine mengenwertigen Attribute gibt. Attribute müssen immer atomar sein.

### 2. Beispiel (schlecht)

| **Bücher** | | |
|---|---|---|
| **ISBN** | **Titel** | **Autor** |
| 12345 | Datenbanken | Erika Muster |
| 34567 | XML Spec | Erwin Mai |
| 78910 | Graf und Welt | Erika Muster |
| 78910 | Graf und Welt | Erwin Mai |

Jetzt sind alle Attribute in der Tabelle atomar. Allerdings ist der Titel des Buches „Graf und Welt“ doppelt abgespeichert. Es entsteht der Eindruck, dass der Titel eines Buches von der ISBN und dem Autor abhängt. Der Titel ist aber durch die ISBN schon eindeutig bestimmt. Das Paar (ISBN, Autor) liefert einen sogenannten Schlüssel für die Tabelle. D.h. alle weiteren Attribute der Tabelle sind durch diese Schlüsselattribute eindeutig bestimmt. Die **zweite Normierung** fordert nun, dass es keine Attribute gibt, die nicht zur Menge der Schlüsselattribute gehören, aber bereits durch einen Teil des Schlüssels bestimmt sind, wie im Beispiel der Titel bereits durch die ISBN bestimmt ist. Um das Problem zu umgehen, wird die Tabelle gespalten.

### Theoretischer Exkurs

Der Vollständigkeit halber, sei hier auch die **dritte Normierung** erwähnt. Sie verlangt, dass Attribute, die keine Schlüsselattribute sind, nicht transitiv von den Schlüsselattributen abhängen. Ein Beispiel dafür wäre eine Tabelle von Personen mit Name, Geburtsdatum, Postleitzahl und Wohnort. Der Name liefert zusammen mit dem Geburtsdatum den Schlüssel. Von diesem Schlüssel hängt die Postleitzahl ab und davon wiederum der Wohnort. Das heißt, der Wohnort ist transitiv abhängig vom Schlüssel. Dies soll durch weiteres Aufspalten der Tabellen vermieden werden.

Auf die Normierungen soll an dieser Stelle nicht weiter eingegangen werden, die schlechten Beispiele bekommen nur noch eine Überarbeitung:

### 3. Beispiel (gut)

| **Bücher** | |
|---|---|
| **ISBN** | **Titel** |
| 12345 | Datenbanken |
| 34567 | XML Spec |
| 78910 | Graf und Welt |

| **Autoren** | |
|---|---|
| **Id** | **Name** |
| 1 | Erika Muster |
| 2 | Erwin Mai |

| **geschrieben von** | |
|---|---|
| **ISBN** | **AutorenId** |
| 12345 | 1 |
| 34567 | 2 |
| 78910 | 1 |
| 78910 | 2 |

## SQL Anfragen

Der Aufbau eines relationalen Datenbankschemas ist nun detailliert besprochen. Es sollen hier noch zwei kurze Beispiele gegeben werden, wie Anfragen an ein solches Schema gestellt werden können. Als erstes soll eine Liste aller Titel von Büchern aus dem oben konstruierten Datenbankschema ausgegeben werden:

```
select Titel from Bücher
```

Ein komplexeres Beispiel liefert die Frage nach den Autoren des Buches mit dem Titel „Graf und Welt“. Um die Spalten der einzelnen Tabellen nicht zu verwechseln, werden den Tabellen in der Anfrage Namen (a,g,b) gegeben. Die Spaltennamen werden dann mit einem Punkt an die Tabellennamen angehängt. Verwechselungen sind dadurch ausgeschlossen.

```
select a.Name from Autoren a, geschrieben_von g, Bücher b where
b.Titel=„Graf und Welt“ and g.ISBN=b.ISBN and g.AutorenId=a.Id
```

Relationale Datenbanken können an dieser Stelle nicht umfassend behandelt werden, die obige Beschreibung reicht jedoch aus, um die Probleme und Möglichkeiten zu verstehen, die bei der Speicherung von RDF entstehen.

# Exkurs: Objektorientierte Datenbanken

Ein weiteres Beispiel einer Datenbankarchitektur stellt der objektorientierte Ansatz dar. Ein Objekt wird hier durch Attribute und Relationen zu anderen Objekten beschrieben. Objekte können aus unseren obigen Beispielen wieder Bücher, Autoren, Leser oder einfach Personen sein.

Der erste Unterschied zu dem relationalen Ansatz ist also das Fehlen der Tabellen. Dies ist vielleicht ein Verlust an Übersichtlichkeit. Allerdings entspricht es eher dem Entity-Relationship Modell, dass die Relationen zusammen mit den Entitäten gespeichert werden.

Der große Vorteil des objektorientierten Ansatzes ist allerdings die Vererbung. Vererbung meint, dass es allgemeinere Objekte gibt und deren Spezialisierungen. Ein Beispiel ist die Speicherung von Autoren, Lesern und Bibliothekaren in einer Bibliothek. Alle drei sind Personen, haben einen Namen, ein Geburtsdatum, eine Adresse. Zusätzlich müssen aber weitere Daten gespeichert werden: Der Autor hat vielleicht einen Künstlernamen und Bücher geschrieben. Vom Leser muss gespeichert werden, welche Bücher er ausgeliehen hat. Eine Kodierung dieser Objekte könnte wie folgt aussehen.

```
class Person {
       attribute string name; }
class Autor extends Person {
       attribute string Künstlername;
       relationship set (Buch) schreibt inverse Buch::geschrieben_von;}
class Leser extends Person {
       relationship set (Buch) entleiht inverse Buch::entliehen von;}
```

Will man Vererbung oder Spezialisierung im relationalen Modell realisieren, so muss eine weitere Relation „ist ein“ eingeführt werden. Dies erscheint unnatürlich und ist auch sehr aufwendig.

Zusammenfassend erscheint uns das objektorientierte Modell als treuere Realisierung unseres Entity-Relationship Modells, auf dem unsere Datenwelt basiert. Allerdings gibt es bisher keine Standardisierung und keine so weite Verbreitung wie bei den relationalen Datenbanken.

Die Probleme, die bei der Speicherung von RDF in Datenbanken auftauchen, sind in beiden Architekturen analog. Aus diesem Grund beschränken wir uns im weiteren auf die relationale Architektur.

## RDF in Datenbanken

Das große Problem bei der Speicherung von RDF in Datenbanken besteht darin, dass vor dem Lesen des RDF Datensatzes im Allgemeinen nicht bekannt ist, welche Attribute und Objekte in diesem Datensatz auftauchen.

Im relationalen Datenbankschema würde das bedeuten, dass eine Tabelle sich verändern oder neu entstehen muss, wenn ein Dokument mit bisher nicht verwendeten Attributen gespeichert werden soll. Dies ist aber nicht effektiv möglich, da dann ja auch die Schnittstellen zur Eingabe und Abfrage der Datenbank geändert werden müssten.

Im Internet wurden in der W3C RDF-special-interest-group [RDFIG], seit 2004 als Semantic Web Interest Group [SWIG] verschiedene Ansätze zur Speicherung von RDF in relationalen Datenbanken diskutiert. Zwei dieser Ansätze werden im folgenden vorgestellt und diskutiert. Eine technische Zusammenfassung der Diskussion findet sich unter [SM].

### Der naive Zugang (Eric Miller)

Der erste Vorschlag basiert auf der Idee, dass RDF sich grundsätzlich in Tripeln darstellen lässt. Alle Tripel werden in einer Tabelle gespeichert. Die Konfiguration sieht folgendermaßen aus:

```
CREATE TABLE triple (
                property          varchar(255),
                resource          varchar(255),
                value    blob
                hint              char(1));
```

Properties und Ressourcen werden in Strings abgespeichert. Die Objekte werden als binary large objects (`blob`) gespeichert. Dies ermöglicht auch die Speicherung von binären Daten. Durch das Attribute hint, das aus einem Buchstaben besteht, ist es möglich kenntlich zu machen, ob in einem der Felder genetic ids gespeichert sind, oder ob es sich bei dem Objekt um eine Resource oder ein Literal handelt.

Dieser Zugang liefert eine korrekte Lösung des Problems, allerdings ist es in dieser Konfiguration nicht möglich nach oder in Objekten zu suchen: Ist beispielsweise der Autor „Erwin“ eines Dokumentes gespeichert, so ist sein Name ein Objekt, das als `blob` abgelegt ist. In `blob` Daten kann aber nicht gesucht werden, die Anfrage „welche Bücher hat Erwin geschrieben“ kann von der Datenbank nicht beantwortet werden. Dies ist eine große Einschränkung. Das Problem lässt sich aber leicht umgehen, indem ein weiteres Attribut eingeführt wird, das binäre Literals abspeichert, in denen dann nicht gesucht werden kann, und das Feld `value` ebenfalls als String deklariert wird, um so eine Suche zuzulassen.

Diesen Schritt geht PhysNet [PN], hier wurde RDF als Datenmodell für eine Faktensammlung gewählt. Die Ontologie ist in OWL-Lite formuliert und die Tripel bestehen aus Strings, binäre Werte gibt es in dieser Sammlung nicht. Insofern handelt es sich um eine Implementation, die nicht vollkommen allgemein ist. Aus der Datenbank werden jeweils nachts statische HTML-Seiten generiert, so dass die Performance zur Runtime hier wenig kritisch ist. [PNCCQ]

Ein weiteres Problem bei der Suche taucht auf, das viel erheblicher und nicht so einfach zu lösen ist: Wird nach einem konkreten Statement gesucht, müssen im schlimmsten Fall alle Datensätze angeschaut werden. Dies ist sicherlich wenig effektiv und im Vergleich zur Speicherung von RDF im Dateisystem nicht wesentlich schneller.

## Der spezifikationstreue Ansatz (Jonas Liljegren)

Ein weiterer Schwachpunkt im Vorschlag von Eric Miller liegt bei der Reifizierung. Wird auf ein Statement verwiesen, so muss das Statement in vier Tripel aufgelöst werden, damit der Verweis auf dieses Statement realisiert werden kann. Dies entspricht zwar der RDF Spezifikation, ist aber für eine effiziente Speicherung der Daten völlig ungeeignet.

Der große Unterschied liegt jetzt in der Trennung von Statements und Resourcen:

```
CREATE TABLE statements (
       id, pred, subj, obj, fact            int);

CREATE TABLE resources (
       id, isprefix                         int,
       uri, value, lang                     text);
```

Die Tabelle `statements` enthält jetzt nicht mehr die eingentlichen Daten, sondern nur noch die ids der entsprechenden Resourcen. Subject, Predicate und Object eines RDF Statements werden jeweils als Resource abgespeichert. Zusätzlich wird jedes Statement durch seine id zu einer Resource, in dem ein lokaler Namespace genutzt und die id angehängt wird (URI=`local#id`). Im Feld `fact` kann durch eine Ziffer gekennzeichnet werden, ob das jeweilige Statement als Ressource verwendet wird (f`act=0`).

Die ids der Statements und der Ressourcen werden eindeutig generiert, d.h. wenn eine id bereits für eine Ressource vergeben wurde, kann es kein Statement mehr mit derselben id geben.

In der Tabelle `resources` werden die eigentlichen Daten gespeichert. Handelt es sich dabei um einen Literal, so wird nicht das Feld `uri` belegt, sondern das Feld `value`. Zusätzlich wird das Language Attribut abgespeichert.

Durch die effektivere Speicherung von reifizierten Statements liegt der Suchaufwand bei diesem Ansatz unter dem Suchaufwand im Vorschlag von Eric Miller, allerdings ist er noch immer sehr hoch.

## Zusammenfassung

Die Speicherung von RDF in relationalen Datenbanken ist ohne Verluste möglich. Allerdings ist die Suche nach vorgegebenen Attributen, z.B. nach einem Autorennamen sehr rechenzeitaufwendig. Dies liegt daran, dass es keine ausgezeichneten Tabellen für spezielle Attribute gibt.

Allgemeine Strategie sollte es in einem konkreten Datenbankschema daher sein, die Tripel wie oben beschrieben abzuspeichern, allerdings für häufig genutzte Attribute (Autor, Titel) spezielle Tabellen anzulegen. Das führt zu einer erheblichen Beschleunigung der Suchabfragen, wobei die Information noch immer vollständig gespeichert ist.

Alternativ ist es auch möglich, die RDF Daten im Filesystem zu speichern und nur für ausgewählte Attribute Tabellen in einem Datenbankschema anzulegen. Das jeweilige Vorgehen hängt dabei wesentlich von der konkreten Anwendung ab.

Zu berücksichtigen ist dabei natürlich auch die Art und Weise in der Nutzer auf die Datenbank zugreifen können sollen.

## Weitere Ansätze zur Speicherung von RDF

Neben den Ansätzen zur Speicherung von RDF in Datenbanken, gibt es Modelle zur Speicherung, die auf der Idee basieren, dass RDF in XML darstellbar ist. XML selbst hat eine Baumstruktur, auf der Suche effektiv möglich ist.

Leider gibt es bei diesem Vorgehen zwei Probleme: Erstens setzen viele der Systeme voraus, dass eine DTD für die zu verwaltenden XML Dokumente vorliegt. Dadurch können beliebige XML Dokumente nicht angemessen verarbeitet werden. Zweitens wird durch die Reduktion auf XML die Graphen-Struktur von RDF nicht vollständig widergespiegelt.

Ein Beispiel für diesen Ansatz ist HyREX [HyREX] der Universität Duisburg-Essen.

Ein weiteres in diesem Zusammenhang interessantes Projekt ist beispielsweise das Kowari Metastore Vorhaben, eine Open Source Datenbank-Entwicklung zur Speicherung von RDF und OWL. (http://kowari.org/)

# Retrieval auf RDF

Momentan existiert noch keine standardisierte Anfragesprache für RDF Dokumente. Auf Grund der existierenden XML Repräsentation von RDF liegt es jedoch nahe, eine Anfragesprache an XML für das Retrieval auf RDF zu verwenden. Mit XML-Query [XQuery] basierend auf XPath [XPATH] liegt eine standardisierte Anfragesprache an XML vor, die in den wichtigsten Programmiersprachen implementiert ist.

Es zeigt sich jedoch sehr schnell, dass XQuery nur in Spezialfällen für das Retrieval auf RDF geeignet ist. Da die Entwicklungen von RDF Anfragesprachen jedoch noch weit von einer Standardisierung und Implementierung entfernt sind, gibt es zu dem Umweg über XML häufig keine Alternative.

Aus diesem Grund wird zunächst das Retrieval auf XML Dokumenten mit Hilfe von XPath und die Anwendung dieser Technik auf RDF Dokumente vorgestellt. Im letzten Kapitel wird ein Überblick über den aktuellen Stand der Entwicklung von speziellen RDF Anfragesprachen gegeben. Insbesondere wird mit RQL [RQL] eine Anfragesprache vorgestellt, die sowohl RDF als auch RDF Schema unterstützt. Schließlich wird die aktuelle Arbeit an der RDF-Retrieval-Sprache SPARQL umrissen.

## Retrieval auf XML

Ein XML Dokument zeichnet sich im Wesentlichen durch seine baumartige Struktur aus:

```
<document class="H.3.3">
    <author>John Smith</author>
    <title>XML Retrieval</title>
    <chapter>
        <heading>Introduction</heading>
        This text explains all about XML and IR.
    </chapter>
    <chapter>
        <heading>Extensible Style Language</heading>
        <section>
            <heading>Examples</heading>
        </section>
        <section>
            <heading>Syntax</heading>
        </section>
    </chapter>
</document>
```

Die grundlegende Funktionsweise von XPath wird nun anhand einiger Beispiele demonstriert.

Die XPath Anweisung `//heading` liefert als Ergebnis die Werte alle `<heading>` Elemente zurück: `Introduction`, `Extensible Style Language` und `Syntax`. Die Anweisung `//chapter/heading` liefert alle `<heading>` Elemente, deren Elternelement ein `<chapter>` ist. Im obigen Beispiel würden also nur die Werte `Introduction` und `Extensible Style Language` als Ergebnis geliefert werden.

Durch das Anhängen eines Filters `[..]` an den Pfad lässt sich die Ergebnismenge weiter einschränken. Zum Beispiel liefert die Anfrage `//chapter[heading]` alle `<chapter>` zurück, die ein Kindelement `<heading>` haben. (Man vergleiche dies mit der Anweisung `//chapter/heading`, die als Ergebnis `<heading>` und keine `<chapter>` liefert!)

Weiterhin gibt es eine Reihe von Vergleichsoperatoren: Die Anfrage `//chapter[heading="XML"]` liefert alle `<chapter>` Elemente zurück, deren Heading-Kindelemente den String „XML“ enthält. Auch Attribute lassen sich über diesen Mechanismus abfragen. Die Anfrage `/document[@class="H.3.3" \$and\$ author="John Smith"]` liefert alle Dokumente zurück, deren Attribut `class` und deren Kindelement `<author>` mit den angegebenen Werten übereinstimmen.

Die hier vorgestellten Anfragen beschreiben nur einen kleinen Teil der Funktionalität von XPath. Eine vollständige Beschreibung liefert die Spezifikation [XPATH].

# XQuery

Die folgende Abhandlung über XQuery ist im Wesentlichen dem Cashmere-int Newletter 3 entnommen [CNL3XQ].

XQuery ist eine Abfragesprache, die die Struktur von XML intelligent nutzt, um Abfragen über Daten zu formulieren, die entweder physisch in XML gespeichert sind oder über eine Middleware als XML betrachtet werden können. Entwicklungsanforderung war die Spezifizierung einer Sprache, in der Abfragen prägnant und leicht verständlich sind. XQuery ist flexibel genug, um XML Daten in Dokumenten und Datenbanken abzufragen. XQuery wurde von der XML-Abfragesprache Quilt abgeleitet, die wiederum einige Features von verschiedenen Sprachen wie XPath, XQL, SQL etc. entliehen hat. XQuery wird vom W3C definiert und wird von allen bedeutenden Datenbankentwicklern wie IBM, Microsoft, Oracle, Software AG unterstützt. Am treffendsten kann XQuery folgendermaßen beschrieben werden: „XQuery ist für XML was SQL für relationale Datenbanken ist.“ Folglich ist XQuery eine Sprache zur Suche und Extraktion von Elementen und Attributen in XML Dokumenten.

## Grundlagen

Der Hauptbaustein von XQuery ist die Expression, ein String von Unicode Zeichen. Es ist mit verschiedenen Arten von Expressions ausgestattet, die aus Schlüsselwörtern, Symbolen und Operanden bestehen. Wie auch XML ist XQuery eine Sprache, in der Groß-Klein-Schreibung relevant ist. XQuery 1.0 und XPath 2.0 haben dasselbe Datenmodell und unterstützen dieselben Funktionen und Operatoren. XQuery ist mit verschiedenen W3C Standards wie z. B. XML, Namespaces, XSLT, XPath und XML Schema kompatibel. XQuery 1.0 ist ein stabiles W3C Candidate-Recommendation.

## Query Formulierung

### Funktionen

XQuery benutzt Funktionen, um Daten aus XML Dokumenten zu extrahieren. Es stellt eine große Auswahl an eingebauten Funktionen zur Verfügung, die von Stringverarbeitung, Daten- und Zeitkonvertierung, Knotenverarbeitung, booleschen Operatoren bis hin zu numerischen Werten reichen.

### Path Expressions

XQuery benutzt Path Expressions um ein XML Dokument zu erschließen. Die Path Expressions werden gemeinsam mit den Funktionen auf das XML Dokument angewandt, um Daten abzurufen.

### Prädikat

XQuery benutzt Prädikate um die aus XML-Dokumenten extrahierten Daten einzuschränken. Prädikate werden mit Path Expressions kombiniert.

### FLWOR

In XQuery können Elemente entweder mit Path Expressions kombiniert mit Prädikaten oder mit FLOWR Expressions selektiert und gefiltert werden. FLOWR ist ein Akronym für:

- **F**or[in] - bindet eine Variable an jedes Element das mit dem [in]-Ausdruck abgerufen wird
- **L**et - ermöglicht die Verwendung von Variablen (optional)
- **W**here - präzisiert ein Kriterium (optional)
- **O**rder by - präzisiert die Sortierreihenfolge des Ergebnisses (optional)
- **R**eturn - präzisiert, was im Ergebnis zurückgeliefert werden soll

Der Vorteil bei der Benutzung von FLOWR Expressions ist, dass es eine SQL-ähnliche Syntax besitzt und mehr Funktionalität und Flexibilität hinzufügt.

```
for $x in doc("book.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Die Übersetzung der oben angeführten FLOWR Expression in einfaches Deutsch wäre: *Für die Variable x im Dokument "Book.xml" sollen alle Titel Elemente abgerufen werden, wobei der Preis des Buches größer als 30 ist und die Ergebnisse sollen dem Titel nach geordnet sein.*

## Grundlegende Syntaxregeln

- XQuery ist groß-/klein-sensitiv, was bedeutet dass kleingeschriebene Variablen nicht dieselben sind wie die großgeschriebenen.
- XQuery Elemente, Variablen müssen gültige XML Namen sein.
- Ein XQuery String Wert kann in Einzel- oder Doppelanführungsstrichen stehen.
- Eine XQuery Variable wird definiert mit $ gefolgt von einem Namen, z. B. $bookstore.
- XQuery Kommentare werden beschränkt durch (: und :), z. B. (:XQuery Comment:).

### Bedingte Ausdrücke (Expressions) in XQuery

XQuery ermöglicht auch den Gebrauch des bedingten Ausdruckes if-then-else. Der if-then-else Satz kann nach dem Return-Ausdruck der FLOWR Expression hinzugefügt werden.

### Vergleiche in XQuery

In XQuery gibt es zwei Möglichkeiten, Werte zu vergleichen:

- Allgemeiner Vergleich: =, !=, <, <=, >, >=. Wenn diese Operatoren für einen Vergleich herangezogen werden, wird die Bedingung als wahr ausgegeben falls beliebige Attribute die Bedingung erfüllten.
- Wertevergleich: eq, ne, lt, le, gt, ge. Wenn diese Operatoren für einen Vergleich herangezogen werden, wird die Bedingung als wahr ausgegeben, genau dann wenn ein Attribut die Bedingung erfüllt.

### Typenbeschreibung

- XQuery ist eine streng typisierte Programmiersprache.
- Wie Java, C# und andere Programmiersprachen kennt es eine Mischung aus statischen und dynamischen Typisierungen.
- Typen in XQuery unterscheiden sich von Klassen in Objekt Orientierten Programmiersprachen.
- Typen entsprechen dem XQuery Datenmodell.

### Schlussfolgerung

- XQuery ist ein leistungsfähiges und komfortables Tool zur Analyse und Erzeugung von XML.
- XQuery ist protokollunabhängig und kann daher auf jedem System mit vorhersehbaren Ergebnissen angewandt werden.
- Es gibt keine Standardimplementationen, aber es befindet sich eine Liste mit den bekannten Implementationen auf der XQuery-Site.
- XQuery ist kompatibel mit anderen W3C-Standards, nämlich Namespaces, XML Schema, XPath.
- XQuery lässt sich nur in sofern für RDF-Retrieval einsetzen, als XPath hier nutzbar ist.

## Anwendung von XPath auf RDF

In diesem Abschnitt wird nun gezeigt, wie sich Anfragesprachen an XML für das Retrieval auf RDF einsetzen lassen und welche Probleme bei dieser Vorgehensweise auftreten.

#### Beispiel 1

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://purl.org/dc/elements/1.1/" >
  <rdf:Description>
    <dc:creator>Stefan Kokkelink</dc:creator>
  </rdf:Description>
...
</rdf:RDF>
```

Die Anweisung `//dc:creator` liefert alle `dc:creator` zurück, unabhängig davon, an welcher Stelle diese Property in dem RDF Dokument vorkommt. Durch die Angabe eines vollständigen Pfades `/rdf:RDF/rdf:Description/dc:creator` lässt sich die Resultatmenge einschränken. Auch komplexere Anfragen lassen sich auf diese Weise an RDF Dokumente stellen:

#### Beispiel 2

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/" >
  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_A">
    <dc:creator>Hartmut Polzer</dc:creator>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_B">
    <dc:publisher>Springer-Verlag</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

Die Anfrage `//rdf:Description[dc:*]/@about` liefert alle URIs zurück, die durch Dublin Core Metadaten beschrieben wurden.

### Beispiel 3

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/" >
  <rdf:Description rdf:ID="Document">
    <dc:publisher>Springer-Verlag</dc:publisher>
    <dc:creator>
      <rdf:Description>
        <vCard:FN>Hartmut Polzer</vCard:FN>
        <vCard:EMAIL>hartmut@gmx.de</vCard:EMAIL>
        .....
      </rdf:Description>
    </dc:creator>
  </rdf:Description>
</rdf:RDF>
```

Die Anfrage

`//dc:creator/rdf:Description/vCard:FN[../../../dc:publisher="Springer-Verlag"]` liefert alle Autorennamen, die eine Arbeit im Springer-Verlag publiziert haben.

Das hier beschriebene Retrieval auf RDF läßt sich nur dann mit Hilfe von XML Anfragesprachen sinnvoll durchführen, wenn man die XML Struktur des zugrunde liegenden RDFs genau kennt. Das kann z.B. dann der Fall sein, wenn man die XML Präsentation selbst erstellt hat.

Im Allgemeinen eignet sich diese Form des Retrievals auf RDF nicht, wie die folgenden Überlegungen zeigen:

- Für einen RDF Graphen kann es viele Repräsentationen in XML geben.
- Das Datenmodell von RDF ist ein gerichteter Graph, das Datenmodell von XML ist im Wesentlichen ein Baum.

### Beispiel 4

Die beiden folgenden RDF Dokumente sind äquivalent (d.h. der Parser erzeugt dasselbe Tripelset):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/" >
  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_A">
    <dc:creator>Hartmut Polzer</dc:creator>
    <dc:publisher>Springer-Verlag</dc:publisher>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/" >
  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_A">
    <dc:creator>Hartmut Polzer</dc:creator>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.iwi-iuk.org/Document_A">
    <dc:publisher>Springer-Verlag</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

Es sollen nun alle Autoren gefunden werden, die im Springer-Verlag publiziert haben. In der ersten XML Version würde die Anfrage `//dc:creator[../dc:publisher="Springer-Verlag"]` das gewünschte Resultat bringen. In der zweiten XML Version gibt es jedoch auf diese Anfrage keinen Treffer, obwohl die zugrunde liegenden RDF Graphen identisch sind!

## RQL: RDF Query Language

Im letzten Abschnitt wurde deutlich, dass sich Anfragesprachen für XML nur sehr beschränkt für das Retrieval auf RDF eignen. Dieser Absatz basiert auf einer Entwicklung aus den Jahren 1996 bis 2001, die hierauf aufbauende Entwicklung an SPARQL wird im darauffolgenden Absatz dargestellt. Voraus laufende Entwicklungen waren u.a.:

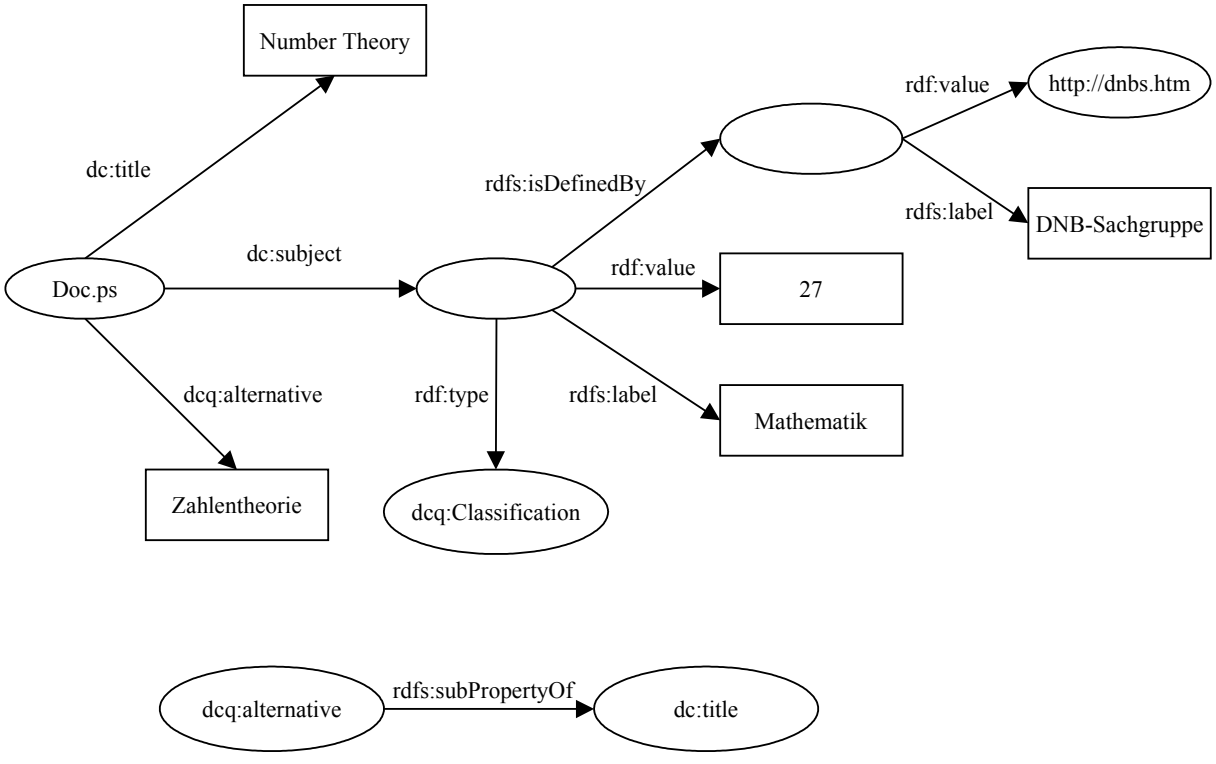
- Metalog - The RDF querying system (1999-05-02) [Metalog]
- RDF Query Specification (1998-12-03) [RDFQS]
- A Query and Inference Service for RDF ( 1998-11-18 ) [QISRDF]
- Querying Semistructured (Meta)data and Schemas on the Web: The case of RDF&RDFS (2000-02-25) [RQL]

Eine echte Unterstützung von RDF Schema bietet nur die in [RQL] spezifizierte Anfragesprache RQL. Sie wird hier vorgestellt, auch weil hier die Grundlage für alle weiteren Entwicklungen gelegt wurden und diese damit verständlicher werden.

Die folgende Liste fasst die wesentlichen Eigenschaften von RQL zusammen:

- RQL basiert auf dem Graphenmodell von RDF.
- Die Syntax von RQL ähnelt SQL.
- RQL unterstützt RDF Schema, insbesondere die durch (`subClassOf, subPropertyOf`) definierten Relationen von Klassen und Properties.

Die unterschiedlichen Anfragemöglichkeiten von RQL werden anhand des folgenden RDF Graphen demonstriert.



## Einfache Anfragen

Das Vorhandensein bestimmter Properties in einem RDF Graphen kann durch die Angabe ihrer URI überprüft werden. Zum Beispiel liefert die Anfrage `dcq:alternative` (die URI wird wie üblich mit Hilfe des jeweiligen Namespaces abgekürzt dargestellt) als Ergebnis das folgende Paar:

| Source | Target |
|---|---|
| Doc.ps | Zahlentheorie |

Die Anfrage `rdf:value` ergibt:

| Source | Target |
|---|---|
| genID_2 | http://dnbs.htm |
| genID_1 | 27 |

RQL berücksichtigt die Relation `subPropertyOf` der zugrunde liegenden RDF Schemas. Die Anfrage `dc:title` ergibt:

| Source | Target |
|---|---|
| Doc.ps | Number Theory |
| Doc.ps | Zahlentheorie |

Es werden also automatisch auch die Werte alle Subproperties von `dc:title` angezeigt! Man kann diese Funktionalität durch den `^` Operator verhindern. So würde die Anfrage `^dc:title` nur den einen Wert „NumberTheory“ liefern.

Analog lassen sich Klassen abfragen. Auch hier wird immer bezüglich der Relation `subClassOf` expandiert.

### Filter

Um die Graphstruktur von RDF auszunutzen, bietet RQL einen select-from-where Filter analog zu SQL an. Zum Beispiel ist die Anfrage `dc:title` äquivalent zur Anfrage **`select`** `X,Y` **`from`** `{X}dc:title{Y}`. Diese Anfrage setzt sich aus zwei Teilen zusammen: Zunächst werden die Objekte angegeben, die als Resultat geliefert werden sollen (hier Source X und Target Y), dann wird ein Teilgraph angegeben, in denen die Objekte auftreten müssen (hier `{X}dc:title{Y}`). Möchte man nur die Ressourcen erhalten, von denen eine Property `dc:title` startet, so müsste man folgende Anfrage stellen: **`select`** `X` **`from`** `{X}dc:title{Y}`.

Es können noch weitere Bedingungen formuliert werden. Im folgenden Beispiel werden im Ergebnis alle Titel (`dc:title`) von Ressourcen angezeigt, deren `dc:subject` auf eine Ressource zeigt, deren `rdf:value` „Mathematik“ ist:

```
select {Y} from {X}dc:title{Y},{X}dc:subject{Z}.rdf:value{W}
where W="Mathematik"
```

Bezüglich des obigen RDF Graphen würde diese Anfrage als Ergebnis „Zahlentheorie“ und „Number Theory“ liefern. (Zur Erinnerung: Es wird bezüglich der Relation `supPropertyOf` expandiert.)

Auf diese Weise lassen sich sehr komplexe Anfragen realisieren. Für eine Übersicht über die gesamte Funtionalität von RQL sei auf die Spezifikation verwiesen.

# SPARQL – Eine Retrieval-Sprache für RDF

Die folgende Abhandlung über SPARQL ist im Wesentlichen dem Cashmere-int Newletter 3 entnommen [CNL3SQ].

In den letzten Jahren wurden viele Bemühungen der Ressourcen-Erschließung im Semantic Web unternommen. Viele RDF-generierende Tools, Parser und Datenbanken sind bereits erstellt worden. Viele Tools werden derzeit entwickelt oder weiterentwickelt. RDF-Datenbanken-Implementierungen sind im Web zugänglich, sowohl als kommerzielle als auch als Open-Source Software. Einige basieren auf relationalen Datenbanken; andere nutzen die Möglichkeit RDF-Daten kodiert in RDF/XML in einer XML-Datenbank zu speichern.

Leider ist bis jetzt kein Query-Sprachen-Standard aus diesen Entwicklungs-Aktivitäten entstanden. Implementierungen – basierend auf relationalen Datenbanken – definieren häufig ihre eigene Query-Sprache. Einige dieser Sprachen erinnern in ihrer Syntax an RDF selbst. Einerseits kann dieser Ansatz als ein natürlicher Weg betrachtet werden, Informationen, die in RDF kodiert sind, anzusprechen. Andererseits wurde RDF konstruiert, um für Maschinen lesbar zu erschließen, folglich sind RDF-Ausdrücke für Menschen schwierig zu lesen. Softwareentwicklungen, die auf XML-Datenbanken basieren, bieten die Möglichkeit, RDF/XML mit XQuery, einer XML Query-Sprache und/oder XPath-Ausdrücken, abzufragen; alles sind W3C-Standards zur Suche in XML-Dokumenten. Das Problem dieses Ansatzes ist, dass, im Vergleich mit XML, RDF eine flexiblere Struktur hat. Das zu Grunde liegende Datenmodel von RDF ist ein Graph während das von XML ein Baum ist. Das bedeutet, dass die XML-Darstellung eines RDF-Dokuments nicht eindeutig ist. Demnach kann die Information, die man durch Abfragen bestimmter XML-Darstellungen von RDF-Dokumenten erhält, unvollständig sein.

Um diese Probleme anzugehen, hat die W3C RDF Data Access Working Group (DAWG) [DAWG] Use Cases entworfen, die Anwendungs-Szenarien einer RDF-Anfrage-Sprache zusammenstellen. Ausgehend davon wurde eine Liste von Anforderungen an eine solche Sprache erstellt, darunter auch die potentielle Lesbarkeit durch den Nutzer [RDAUCR]. Am 19. April 2005 hat das W3C eine neue Version des SPARQL Working Draft veröffentlicht [SQLR]. SPARQL steht für "Protocol And RDF Query Language" und bietet ein menschenlesbares Interface zur RDF-Datengewinnung. SPARQL ist ausgelegt, die Anforderungen und Designzielsetzungen, die durch die DAWG beschrieben wurden, zu erfüllen. Auf den ersten Blick sieht die Syntax von SPARQL wie eine Mischung aus SQL und Prolog aus. Das Interfacedesign von SPARQL ist ziemlich intuitiv.

Die Entwicklung von SPARQL zielt auf die Erstellung eines Standards im Bereich der Ressourcen-Erschließung im Semantic Web ab. Einige Open-Source Projekte wie z. B. Jena (von HP Labs Semantic Web Research entwickeltes Java-Framework für RDF) [HPLSWR] haben bereits ihr Interesse signalisiert, SPARQL in ihren RDF Datenbanken zu implementieren. Die Entwickler von Sesam [OpenRDF], einer Open-Source RDF-Datenbank, erklären in ihrem Forum, SPARQL zu unterstützen, sobald die Entwicklung an der Sprache abgeschlossen ist [OpenRDF-SPARQL].

Diese Fakten lassen vermuten, dass SPARQL in seiner endgültigen Version gute Chancen hat, in der RDF-Community angenommen zu werden. Die Arbeit an SPARQL ist noch nicht beendet und es gibt noch einige offene Fragen.

# Werkzeuge zur Nutzung von RDF

Bei der Präsentation von Werkzeugen zum Umgang mit RDF geht es zunächst darum, exemplarisch die Probleme, die existierende Dienste zum Retrieval von mit Metadaten angereicherten Internetdokumenten aufwerfen, zu diskutieren. Eines der hier auftretenden Probleme ist die Benutzung unterschiedlicher Metadaten-Formate. Wir konzentrieren uns hier auf den Bereich von Personendaten und stellen den Internetstandard vCard vor.

Abschließend wird ein Werkzeug vorgestellt, das die Erstellung von Metadaten in RDF unterstützt und die Personendaten im vCard Profil speichert.

## TheO – Metadaten für Online-Dissertationen

Der Suchdienst TheO (Theses Online) ermöglicht die Recherche in elektronisch veröffentlichten Dissertationen in der Bundesrepublik Deutschland. Die Deutsche Bibliothek, hat XMeta-Diss als Austauschformat für Dissertations-Metadaten entwickelt, das auf XML basiert. Dennoch werden von den meisten Hochschulen, intern die Metadaten noch in HTML vorgehalten. TheO recherchiert auf dieser HTML-Datenbasis.

Da die Kodierung der Metadaten mit Hilfe des HTML Meta-Tags erfolgt, lassen sich nur sehr flache Datenstrukturen repräsentieren. Insbesondere stellt die unzureichende Darstellung von Personendaten für das Retrieval ein Problem dar. Zum Beispiel liefert die Anfrage *Autor="Husmann"* das folgende Resultat (gekürzt):

| | |
|---|---|
| *Titel:* | Ein Triangulierungsverfahren zur Approximation mit Dahmen-Micchelli-... |
| *Autor:* | Husmann, Markus |
| *Gutachter:* | Gonska, Heinz H.; Prof.Dr. Zhou, Xinlong; Priv.Doz.Dr. |

Es ist erst nach einiger Überlegung zu erkennen, wer von den Gutachtern Professor und wer Privatdozent ist. Zusätzliche Angaben wie Email Adresse oder Homepage lassen sich mit der HTML Kodierung nicht realisieren.

Dieses Beispiel macht deutlich, dass für das ein qualitativ hochwertiges Retrieval ein möglichst anerkannter Standard für die strukturierte Beschreibung von Personendaten benötigt wird.

## Speicherung von Personendaten

Weltweit gibt es viele verschiedene Standards zur Speicherung von Personendaten. Ein Beispiel dafür liefern die verschiedenen Dialekte von MARC. Dies führt zu Problemen beim Zusammenführen verschiedener Datenpools oder bei der Speicherung von Personendaten, die aus dem Internet gewonnen werden.

Um dieses Problem zu lösen, gibt es einerseits die Möglichkeit, die Entwicklung von Crosskonkordanzen zwischen den einzelnen Formaten zu nutzen. Leider steigt die Anzahl der benötigten Konkordanzen quadratisch in Abhängigkeit von der Anzahl der Formate. Jedes neue Format verlangt damit einen sehr großen Arbeitsaufwand durch den Bedarf an neuen Crosskonkordanzen.

Eine weitere Möglichkeit ist die Entwicklung eines Superformates, in das jedes Format übersetzt werden kann. Versuche zur Entwicklung eines solchen Superformates sind bisher allerdings immer gescheitert.

### Position des W3C

Das World Wide Web Consortium (W3C) hat dieses Problem erkannt und durch die Request for comments (RFC) 2425 und 2426 den vielen Formaten für Personendaten ein weiteres hinzugefügt.

Dieses neue Format hat allerdings gegenüber vielen anderen Formaten einen sehr großen Vorteil: Es ist relativ unabhängig von kulturellen und sprachspezifischen Besonderheiten, die es auf der ganzen Welt gibt. Diese Besonderheiten, wie Buchstaben, die nicht aus dem 26er Alphabet stammen, oder Namen, die nicht in Vor- und Nachnamen aufgeteilt sind, führen in klassischen Formaten zu Transliterationen, die für Menschen aus anderen Sprachräumen (z.B. S-Laute aus Osteuropa) nicht verständlich sind.

Ziel von RFC2425 ist die Standardisierung von Adresslisten. Aufbauend darauf definiert RFC2426 mit vCard3.0 ein Format für elektronische Visitenkarten von Personen. RFC2426 ist nach Definition nicht für die Speicherung von Organisationen anwendbar.

## vCard (RFC 2426)

Hier eine Liste von Attributen, die mit vCard3.0 kodiert werden können. Einige dieser Attribute müssen in einer vCard vorkommen, andere sind optional.

- Name, der die Person eindeutig identifiziert (Prefix, Suffix, Titel)
- formatierte Version des Namens zur Präsentation
- strukturierte und präsentierbare Form der Adressen (Dienst- und Privatadresse)
- Telefon- und Faxnummern
- Information über Firmen- oder Organisationszugehörigkeit
- Geburtsdatum
- Bilder: Photo oder Logo der Organisation
- Audioinformation z.B. zur Präsentation der Aussprache des Namens
- Längen- und Breitengrad, die in Relation zu der Person stehen
- Emailadresse
- URL einer Homepage
- public key Information
- Zeit der letzten Änderung / Version
- Künstlername/Spitzname

Eine vollständige Liste der Attribute findet sich im RFC 2426 selbst. Der RFC baut auf anderen RFCs auf und garantiert dadurch gute Verträglichkeit zu anderen Internetkonventionen.

Die Kodierung von vCard3.0 erfolgt in ASCII Dateien, dies garantiert Portabilität und einfachen Transport per Email oder anderen Internetprotokollen.

Hier ein kurzes Beispiel einer vCard:

## Beispiel

```
BEGIN:vCard
VERSION:3.0
FN:Judith Plümer
N: Plümer;Judith;;Dr.
SORT-STRING: Pluemer Judith
ORG: Universität Osnabrück
REV: 2000-06-01
TEL;TYPE=VOICE,MSG,WORK:+4-541-969-2526
EMAIL;TYPE=INTERNET,PREF:judith@work
EMAIL;TYPE=INTERNET:judith@home
END: vCard
```

Die Zeilen BEGIN, VERSION und END rahmen die vCard ein. Unter FN steht der Name, wie er z.B. auf einer Webseite präsentiert wird. Unter N steht eine detaillierte Liste der Namenskomponenten in der Reihenfolge Nachname, Vorname, zusätzlicher Name, Prefix, Suffix. Unter SORT-STRING kann festgelegt werden, wie der Name in einer alphabetischen Liste einsortiert werden soll. ORG kennzeichnet die Organisation, für die die Person arbeitet und REV das Datum, an dem die vCard zuletzt geändert wurde.

Hinter TEL verbirgt sich eine Telefonnummer (VOICE), die zum Arbeitsplatz gehört (WORK) und über die Möglichkeit zum hinterlassen einer Nachricht verfügt (MSG). Als letztes werden zwei Internet-Emailadressen angegeben, wobei die Person bevorzugt die Email an der ersten Adresse empfängt.

Weitere wichtige Elemente einer vCard sind binäre Daten, die nach festen Kodierungsmechanismen (RFC2047) in eine ASCII Form überführt werden. Dadurch wird der Transport von Bilder, Audio- und Videodaten möglich.

## RDF vCard

Renato Iannella hat eine Kodierung von vCard in RDF vorgeschlagen. Sie stellt keine separate Definition von vCard dar, sondern ermöglicht den Transport von vCard im RDF Datenmodell. Da wir uns schon eingehend mit RDF beschäftigt haben, ist es nicht mehr nötig, die RDF Kodierung des obigen (nun etwas verkürzten) Beispiels zu kommentieren:

```
<? xml version="1.0" ?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:vCard="http://imc.org/vCard/3.0#">
<Description>
   <vCard:FN>Judith Plümer</vCard:FN>
   <vCard:N rdf:parseType="Resource">
      <vCard:Family>Plümer</vCard:Family>
      <vCard:Given>Judith</vCard:Given>
   </vCard:N>
   <vCard:SORT-STRING>Pluemer Judith</vCard:SORT-STRING>
   <vCard:TEL rdf:parseType="Resource">
      <rdf:value>+4-541-969-2526</rdf:value>
      <rdf:type resource="http://imc.org/vCard/3.0#work" />
      <rdf:type resource="http://imc.org/vCard/3.0#voice" />
      <rdf:type resource="http://imc.org/vCard/3.0#msg" />
   </vCard:TEL>
   <vCard:EMAIL>
      <rdf:Alt>
         <rdf:li>
            <value>judith@work</value>
            <type resource=".../vCard/3.0#internet" />
            <type resource=".../vCard/3.0#pref" />
         </rdf:li>
         <rdf:li>
            <value>judith@home</value>
            <type resource=".../vCard/3.0#internet" />
         </rdf:li>
      </rdf:Alt>
   </vCard>
</Description>
</RDF>
```

# Referenzen

[CNL3SQ] E. Demidova, T. Severiens: **Cashmere-int Newletter 3 - SPARQL** (2005) *http://www.iwi-iuk.org/cashmere/htdocs/html/newsletter/data/sparql.shtml*

[CNL3XQ] S. Kucheria, T. Severiens: **Cashmere-int Newsletter 3: XPath & XQuery** (2005) *http://www.iwi-iuk.org/cashmere/htdocs/html/newsletter/data/xpath_and_xquery.shtml*

[DAWG] W3C: **RDF Data Access Working Group** (2004 - 2006) *http://www.w3.org/2001/sw/DataAccess/*

[DCMES] DCMI - Dublin Core Metadata Initiative: **DCMI Metadata Terms** (2005) *http://www.dublincore.org/documents/dcmi-terms/*

[DCRDF] DCMI: **Expressing Dublin Core metadata using the Resource Description Framework (RDF) [Working Draft]** (2006) *http://dublincore.org/documents/2006/05/29/dc-rdf/*

[HPLSWR] HP Labs: **Semantic Web Research** (2006) *http://www.hpl.hp.com/semweb/*

[HyREX] N. Fuhr: **HyREX - Hyper-media Retrieval Engine for XML** (2000 - 2006) *http://www.is.informatik.uni-duisburg.de/projects/hyrex/*

[Metalog] W3C: **Metalog - towards the Semantic Web** (1999) *http://www.w3.org/RDF/Metalog/*

[OpenRDF] openRDF.org: **...home of Sesame** (2006) *http://www.openrdf.org/*

[OpenRDF-SPARQL] openRDF.org: **Forum thread: Will Sesame support SPARQL?** (2005 - 2006) *http://www.openrdf.org/forum/mvnforum/viewthread?thread=502*

[PN] EPS: **PhysNet - Physics Departments and Documents Worldwide** (1995 - 2006) *http://www.physnet.net*

[PNCCQ] T. Severiens, C. Thiemann: **RDF Database for PhysNet and similar Portals** (2006) *erscheint in Cataloging & Classification Quarterly (CCQ)*

[QISRDF] S. Decker, D. Brickley, J. Saarela, J. Angele: **A Query and Inference Service for RDF** (1998) *http://www.ilrt.bris.ac.uk/discovery/rdf-dev/purls/papers/QL98-queryservice/*

[RDAUCR] W3C: **RDF Data Access Use Cases and Requirements** (2005) *http://www.w3.org/TR/rdf-dawg-uc/*

[RDF-MSS] W3C: **Recommendation, RDF Model and Syntax** (1999) *http://www.w3.org/1999/02/22-rdf-syntax-ns#*

[RDF-XSS] W3C: **Recommendation, RDF/XML Syntax Specification (Revised)** (2004) *http://www.w3.org/TR/rdf-syntax-grammar/*

[RDFIG] W3C: **RDF Interest Group** (1999 - 2004) *http://www.w3.org/RDF/Interest/*

[RDFQS] W3C: **RDF Query Specification** (1998) *http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html*

[RDFS] W3C: **Recommmendation, RDF Vocabulary Description Language 1.0: RDF Schema** (2004) *http://www.w3.org/TR/rdf-schema/*

[RDFSS] W3C: **Resource Description Framework (RDF) Schema Specification 1.0** (2000) *http://www.w3.org/TR/2000/CR-rdf-schema-20000327/*

[RFC3986] Network Working Group, T. Berners-Lee: **Uniform Resource Identifier (URI): Generic Syntax** (2005) *http://www.ietf.org/rfc/rfc3986.txt*

[RQL] G. Karvounarakis, V. Christophides, D. Plexousakis: **Querying Semistructured (Meta)Data and Schemas on the Web: The case of RDF & RDFS** (2000) *www.ics.forth.gr/isl/publications/paperlink/querying-semistructured-metadata-and.pdf*

[SM] Sergey Melnik: **Storing RDF in a relational database** (2000 - 2001) *http://www-db.stanford.edu/~melnik/rdf/db.html*

[SQLR] W3C: **SPARQL Query Language for RDF (CR)** (2006) *http://www.w3.org/TR/rdf-sparql-query/*

[SWIG] W3C: **Semantic Web Interest Group** (2004 - 2006) *http://www.w3.org/2001/sw/interest/*

[XHTML] W3C: **XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) - W3C Recommendation 26 January 2000, revised 1 August 2002** (2000 - 2002) *http://www.w3.org/TR/xhtml1/*

[XML] W3C: **Extensible Markup Language (XML)** (1996 - 2003) *http://www.w3.org/XML/*

[XPATH] W3C: **Recommendation, XML Path Language (XPath)** (1999) *http://www.w3.org/TR/xpath*

[XQuery] W3C: **Candidate Recommendation, XQuery 1.0: An XML Query Language** (2005) *http://www.w3.org/TR/xquery/*