Beschreibung von Web-basierten Informationssystemen mittels RDF-Metadaten

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)

durch den Fachbereich Wirtschaftswissenschaften der
Universität Duisburg-Essen
Standort Essen

Vorgelegt von:
Reinhold Klapsing
Geburtsort: Borken (Westf.)
Essen 2003

Tag der mündlichen Prüfung:	18. Juli 2003
Erstgutachter:	Prof. Dr. Gustaf Neumann
Zweitgutachter:	Prof. Dr. Stefan Eicker

Danksagung

Danken möchte ich meinem Doktorvater Herrn Prof. Dr. Gustaf Neumann, der mich auf meinem wissenschaftlichen Weg stets mit Rat unterstützte; dies auch aus der Ferne, nachdem er einen Ruf an die Wirtschaftsuniversität Wien angenommen hat.

Ebenfalls danke ich Herrn Prof. Dr. Stefan Eicker für seine Ratschläge und für die Bereitschaft als Gutachter tätig zu werden.

Danken möchte ich auch denjenigen, die sich im Rahmen von Konferenzen, Workshops und Journal- und Buchbeiträgen konstruktiv mit meinen Arbeiten auseinandersetzten bzw. einen organisatorischen Rahmen für die wissenschaftliche Auseinandersetzung lieferten. Stellvertretend seien hier genannt: Jérôme Euzenat (Hrsg. des Buchs 'The Emerging Semantic Web', IOS Press), Hans Ulrich Buhl (Hrsg. des Tagungsbandes der fünften internationalen Wirtschaftsinformatikkonferenz, Physica-Verlag), Peter Mertens (Hrsg. des Buchs 'XML - Komponenten in der Praxis', Springer-Verlag), San Murugesan (Gast-Hrsg. des 'IEEE MultiMedia Journal'), Guus Schreiber (Bereichshrsg. des 'Journal of Electronic Transactions on Artificial Intelligence') und Tim Berners-Lee, der auf eine frühe Arbeit, die ich zusammen mit Dr. Wolfram Conen erstellt habe, aufmerksam machte und dadurch (zumindest für mich) fruchtbare Diskussionen auslöste.

Eine Quelle inspirierender Ideen war die Doktorandengruppe ALICE, die mir Gelegenheit bot in Vorträgen, Diskussionen und Gesprächen den jeweils aktuellen Stand meiner Dissertation darzustellen. Hierfür danke ich Herrn Dr. Wolfram Conen, Herrn Fredj Dridi und Herrn Dr. Eckhart Köppen sehr.

Besonders danken möchte ich Herrn Dr. Wolfram Conen, der durch seinen hervorragenden analytischen Geist und durch seine unermüdliche Diskussionsbereitschaft jederzeit ein kompetenter Ansprechpartner war. Die vielen Stunden, die wir gemeinsam an wissenschaftlichen Beiträgen gearbeitet haben, waren für mich sehr bereichernd.

Für die freundliche und produktive Arbeitsatmosphäre am Fachbereich Wirtschaftswissenschaften danke ich meinen Arbeitskollegen. Insbesondere für die organisatorische und administrative Unterstützung danke ich Herrn Prof. Dr. Günther Pernul, Herrn Prof. Dr. Rony G. Flatscher, Frau Waltraud Schulte-Eversum, Herrn Martin Rüschoff und Herrn Reinhard Thöne.

Bedanken möchte ich mich besonders bei meiner Familie, insbesondere der Familie Carius danke ich für die mentale Unterstützung.

Ganz besonders möchte ich mich bedanken bei meinen lieben Eltern - Maria und Johannes Klapsing - die mir immer jede Unterstützung angeboten haben und ohne die ich den eingeschlagenen Weg nicht hätte wählen können.

Nicht zuletzt danke ich besonders meiner Frau Sabine Carius, die während der Zeit der Anfertigung der Dissertation viele Entbehrungen auf sich genommen hat und mich in meinem Vorhaben immer bestärkt und ermutigt hat.

Reinhold Klapsing Essen, im Januar 2003

Inhaltsverzeichnis

1 Motivation

Das World Wide Web (WWW) [9] ist eine weit verbreitete Technologie, mit der viele Arten von Anwendungen realisiert werden, die Dienste auf vernetzten Rechnern bereitstellen. Die Art der Anwendungen variiert von wenig komplexen und kurzlebigen Systemen bis hin zu sehr komplexen, weltweit verteilten Anwendungen, für die ein langfristiger Einsatz geplant ist. Eine Anforderung an den ursprünglichen Entwurf [9] des WWW (im weiteren auch Web genannt) war der einfache Austausch von Dokumenten mittels heterogener Netzwerksysteme. Kann die Bereitstellung von einzelnen Dokumenten noch ad hoc erfolgen, ist eine komplexe Web-Anwendung ohne systematische Planung und Umsetzung nur schwer zu beherrschen. Viele Entwickler beginnen jedoch direkt mit der Implementierung einer Anwendung. Diese Vorgehensweise und die daraus resultierenden Probleme sind aus der Softwareentwicklung bekannt. Zur Vermeidung dieser Probleme liegt die Nutzung von etablierten Softwareentwicklungs- und -modellierungsmethoden (siehe z.B. [84]) auch für die Entwicklung von komplexen Web-Anwendungen nahe. Der Einsatz dieser Methoden ist jedoch nicht immer angemessen, da die speziellen Web-Implementierungstechnologien und die dokumentenzentrierte hypermediale Architektur des WWW, sowie die Koexistenz von strukturierten und unstrukturierten Daten nicht oder nur wenig berücksichtigt wird. Es ist daher notwendig, diese Methoden für die Entwicklung von Web-Anwendungen anzupassen bzw. neue Methoden zu entwerfen. Forschungsaktivitäten, die sich mit diesem Thema beschäftigen, können unter dem Oberbegriff Web-Engineering zusammengefaßt werden (vergl. [85, 86, 116, 126, 155, 157, 87]). Auch die vorliegende Arbeit beschäftigt sich mit Fragestellungen, die sich aus diesem Bereich ergeben. Speziell wird untersucht, ob sich das Resource Description Framework (RDF) [27, 101] als formales, semantisch interoperables Beschreibungsmittel für Web-basierte Informationssysteme eignet.

RDF [27, 101] ist die technologische Grundlage des sich in Entwicklung befindlichen Semantic Web [53, 182]. Dies erläutert Tim Berners-Lee im Text

"Semantic Web Road map" [10]. RDF wird durch das World Wide Web Consortium (W3C) [186] entwickelt. Das W3C ist ein internationaler Zusammenschluß von Forschungseinrichtungen, Firmen und Organisationen, die das Ziel verfolgen, die weitere Entwicklung von Web-Techniken zu steuern. Das W3C sieht Einsatzgebiete von RDF für eine Reihe von Anwendungen. Hierzu gehören z.B. (vgl. [101]) intelligente Agenten, digitale Bibliotheken, die Verbesserung von Ergebnissen aus Suchmaschinen, Bewertung von Inhalten einer Web-Anwendung durch Dritte (sogn. content rating), automatischer Austausch von Daten, die Benutzerpräferenzen beschreiben. Im Artikel "The Semantic Web" [19] stellen Tim Berners-Lee et al. ihre Visionen hinsichtlich des zukünftigen Semantic-Web dar. Als Beispiel werden Web-Agenten angeführt, die für ihren Benutzer einen Termin bei einem Arzt arrangieren und dabei Bedingungen (engl. constraints), wie z.B. kurzer Anfahrtsweg und guter Ruf des Arzt, berücksichtigen.

Das Semantic Web soll es ermöglichen, Daten derart auszutauschen, daß deren Bedeutung/Semantik (engl. semantic) automatisiert interpretiert werden kann. Web-Anwendungen des Semantic Web sollen mittels der ausgetauschten Daten "Schlußfolgerungen ziehen" (engl. inferencing) bzw. "Überlegungen" (engl. reasoning) anstellen können, um so z.B. als intelligente Agenten stellvertretend für einen Benutzer agieren zu können.

Das große Potential, das im Semantic-Web gesehen wird, zeigt sich z.B. durch die Forschungsförderungen der USA (vgl. DARPA/DAML-Projektes [51]) bzw. der Europäischen Union (vgl. OntoKnowledge/OIL-Projekt [135], OntoWeb [136]), die der Erforschung und dem Aufbau des Semantic-Web dienen.

1.1 Fragestellung und Ziel

Zum Aufbau des Semantic Web sind neue Technologien notwendig, die nicht nur syntaktische, sondern auch semantische Interoperabilität ermöglichen sollen. Syntaktische Interoperabilität kann durch den Einsatz von XML [25] erreicht werden. Die RDF-Spezifikationen [101, 27] wurden entworfen, um die Semantik von Daten auf interoperable Art basierend auf XML zu spezifizieren: "...one of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner".

Es gibt bisher jedoch noch keine Einigung darüber, inwieweit RDF diese Anforderung erfüllt. Eine wesentliche Fragestellung der vorliegenden Arbeit ist, ob RDF "semantische Interoperabilität" ermöglicht (und damit eine geeignete technologische Grundlage für das Semantic Web ist). Unter semantischer Interoperabilität wird im Rahmen dieser Arbeit folgendes verstanden: Semantische Interoperabilität ist gegeben, wenn zwei Interpreter ein RDF-Dokument austauschen und die dadurch repräsentierten Begriffe in ihrer Bedeutung äquivalent interpretieren, d.h. gleiche Begriffe haben gleiche Bedeutung.

In der RDF-Spezifikation [101] wird angeführt, daß RDF zur Beschreibung der Inhalte (engl. content) und der Inhaltsbeziehungen einer Web-Anwendung[1] eingesetzt werden kann. Weitere Fragestellung dieser Arbeit ist, ob sich RDF zur formalen Beschreibung der Inhalte, Strukturen und Zusammenhänge der Bestandteile von Web-Anwendungen bzw. allgemeiner gefaßt, für das Web-Engineering einsetzen läßt. Das in dieser Arbeit vorgestellte Beschreibungsmodell soll folgende Anforderungen erfüllen.

Formal überprüfbare Anforderungen

1. Die Inhalte einer Web-Anwendung sollen über die Erfüllung der Forderung nach Darstellbarkeit durch Web-Clienten (z.B. mittels HTML [144]) hinaus, auch von Maschinen in ihrer Bedeutung interpretiert werden können.

2. Die Inhalte der Web-Anwendung sollen in verschiedenen textuellen Repräsentationen (z.B. HTML und WML [189]) zur Verfügung gestellt werden können. Es sollen also nicht nur HTML-basierte Web-Anwendungen entwickelt werden können, sondern auch XML-basierte Anwendungen, die z.B. die Interaktion mit mobilen Web-Clienten ermöglichen.

3. Es soll ein Ansatz zur Erzeugung von Web-Anwendungen gewählt werden, dessen Anwendbarkeit nicht auf ein bestimmtes Gebiet von Web-Anwendungen festgelegt ist (anwendungsbereichsunabhängiges Beschreibungsmodell).

4. Die Modellierungsdaten sollen zur Web-Anwendung auch im Laufe der Evolution der Web-Anwendung konsistent bleiben.

5. Das Beschreibungsmodell einer Web-Anwendung soll durchgängig sein, d.h. aus den Modellierungsdaten und den (Meta-)Daten einer Web-Anwendung soll die Web-Anwendung automatisch generiert werden können.

6. Das Beschreibungsmodell soll die Modellierungsdaten und die Inhalte einer Web-Anwendung in einem einheitlichen Datenformat erfassen können.

[1] In der RDF-Spezifikation[101] wird der Begriff "Web Site" verwendet, der hier in den Begriff Web-Anwendung übersetzt wird.

7. Das Beschreibungsmodell soll in allen Phasen des Lebenszykluses einer Web-Anwendung eingesetzt werden können, in denen Modellierungsdaten und sonstige Metadaten der Web-Anwendung erfaßt bzw. verarbeitet werden.

8. Verwaltungsdaten (z.B. die Festlegung, wer für bestimmte Informationen der Web-Anwendung verantwortlich ist oder wann Informationen veralten) sollen durch das Beschreibungsmodell erfaßbar sein.

9. Das Beschreibungsmodell soll erweiterbar sein, d.h. es sollen neue Beschreibungsmittel hinzugefügt werden können, ohne daß die grundlegenden Datenstrukturen des Beschreibungsmodells geändert werden müssen.

10. Das Beschreibungsmodell soll die Verwendung verschiedener Abstraktionsebenen zulassen. Die Web-Anwendung soll einerseits sehr abstrakt beschrieben werden können (z.B. in Bezug auf die generellen strukturellen Aspekte der Web-Anwendung). Andererseits soll eine sehr detaillierte Beschreibung möglich sein, wenn es um die Beschreibung von feinstrukturiertem Inhalt geht.[2]

11. Das Beschreibungsmodell soll die Granularität der zu entwerfenden Informationsstruktur nicht vorschreiben.

12. Das Beschreibungsmodell soll die Trennung verschiedener Modellierungsaspekte[3] unterstützen:

    a) Die Definition der Inhalte und die Definition der Darstellung der Inhalte sollen getrennt modelliert werden können.

    b) Die Definition der Inhalte und die Definition der strukturellen Anordnung der Inhalte sollen getrennt modelliert werden können.

13. Wiederverwendung von (a) Modellierungskonzepten und (b) Web-Anwendungskonzepten soll unterstützt werden.

14. Die Modellierungsdaten und sonstige Metadaten einer Web-Anwendung sollen in einem standardisierten Format[4] verfaßt werden.

15. Es soll eine graphische Repräsentation des Beschreibungsmodells verfügbar sein.

[2]Garzotto bezeichnet dies als *authoring-in-the-large* und *authoring-in-the-small* (vgl. [68, 69]).

[3]Coda et. al. bezeichnen dies als *seperation of concerns* (vgl. [37]).

[4]Unter einem standardisierten Format werden hier Datenformate verstanden, die von Standardisierungsgremien, wie z.B. dem W3C oder der IETF veröffentlicht werden.

16. Es sollen frei verfügbare Werkzeuge,[5] deren Programm-Code öffentlich verfügbar ist, zur Verarbeitung des Beschreibungsmodells genutzt werden können.

Soziotechnische Anforderungen

1. Es soll eine Syntax gewählt werden, für die potentiell viele Werkzeuge entwickelt werden.[6]
2. Die Beschreibungsmittel sollen einfach, verständlich und intuitiv nutzbar sein.
3. Die graphische Repräsentation soll den Entwicklungsprozeß unterstützen.
4. Ein arbeitsteiliger Entwicklungsprozeß soll unterstützt werden.

Die Erfüllung formaler Anforderungen läßt sich durch Einsatz formaler Methoden bzw. durch Aufzeigen des Vorhandenseins bestimmter Eigenschaften zeigen. Die Erfüllung soziotechnischer Anforderungen kann ohne weitere Untersuchung nicht objektiv betrachtet werden, da die Beurteilung des Vorhandenseins einer Eigenschaft von der subjektiven Bewertung eines Beobachters abhängt.

1.2 Zusammenfassung der Ergebnisse der Arbeit

Dieser Abschnitt stellt die Essenz der im Rahmen dieser Arbeit gewonnenen Ergebnisse vor. Es werden Begriffe benutzt, die einer weiteren Definition bzw. Erklärung bedürfen. Für die genauen Definitionen und Erklärungen sei auf die folgenden Kapitel der Arbeit verwiesen.

In der vorliegenden Arbeit wird festgestellt, daß die durch die RDF-Spezifikationen beschriebenen Technologien ohne Anpassungen bzw. Erweiterungen nicht ausreichen, um semantische Interoperabilität zwischen Web-Anwendungen zu ermöglichen. Das hat eine Reihe von Gründen, die in den folgenden Abschnitten erläutert werden. Ein wesentlicher Grund ist, daß das in den RDF-Spezifikationen vorgestellte Datenmodell nicht vollständig formal beschrieben ist und deshalb verschiedene Interpretationen zuläßt.

[5] Unter frei verfügbaren Werzeugen werden Softwareprogramme verstanden, die z.B. unter der GNU Public License erhältlich sind.

[6] Es kann davon ausgegangen werden, daß für ein standardisiertes Format eine Reihe von Werkzeugen entwickelt werden. Insofern hängt diese Anforderung mit der Anforderung der Verwendung eines standardisierten Formates zusammen.

1.2.1 Formalisierung der RDF-Semantik mittels Prädikatenlogik

In der vorliegenden Arbeit wird für die in den RDF-Spezifikationen beschriebenen Kernkonzepte eine formale Definition in Prädikatenlogik erster Ordnung (engl. first order logic - FOL) vorgestellt. Die formale Repräsentation der Semantik der RDF-Konzepte ist eine notwendige Voraussetzung, um semantische Interoperabilität zwischen Web-Anwendungen zu ermöglichen. Die im Rahmen dieser Arbeit erstellte Interpretation[7] der Semantik der RDF-Konzepte (siehe auch [39]) war eine der ersten verfügbaren Formalisierungen von RDF in Prädikatenlogik erster Ordnung[8] und wurde von der RDF-Forschungsgemeinschaft diskutiert bzw. als Ausgangspunkt für weitere Arbeiten verwendet (vgl. z.B. [13, 94, 128, 148]). Durch die Formulierung der Fakten und Regeln in Prädikatenlogik ersten Ordnung kann diese Formalisierung auch als Referenzmodell aufgefaßt werden. Es ist möglich, die angegebenen Fakten und Regeln in verschiedene Logik-Sprachen zu überführen. In dieser Arbeit werden die Regeln und Fakten beispielsweise in Prolog- und Datalog-Notation angegeben. Die angegebenen Regeln und Fakten ermöglichen es, RDF-Triple-Mengen als Faktenbasen zu benutzen, welche auf die Einhaltung der in den RDF-Spezifikationen angegebenen Zwänge (engl. constraints) überprüft und damit validiert werden können. Die Faktenbasen können mittels Inferenz-Maschinen befragt werden. Wie oben angeführt, können die Regeln und Fakten in verschiedene Logik-Sprachen überführt werden, womit verschiedene Inferenz-Maschinen und damit auch verschiedene Abfragesprachen zur Verfügung stehen. Es besteht die Möglichkeit, die für einen bestimmten Anwendungszweck geeignete Logik-Sprache und Inferenz-Maschine auszuwählen. Die Integration in schon vorhandene Systeme wird so unterstützt.

Aktuell ergeben sich Änderungen an der Semantik von RDF durch die Definition einer RDF-Modelltheorie (RDF-MT)[76, 77]. Die RDF-Modell-Theorie steht derzeit noch zur Diskussion und eine zeitnahe Standardisierung ist unwahrscheinlich. Dennoch wird auf die sich daraus ergebenden Konsequenzen bzgl. der Semantik der RDF-Konzepte in dieser Arbeit eingegangen. Hierzu wird die Semantik, die durch die RDF-MT definiert wird, ebenfalls in Prädi-

[7] Eine Formalisierung von RDF kann nur eine Interpretation der intendierten Semantik sein, da RDF nur informal beschrieben ist. Wie aus vielen Diskussionen in den RDF-Mailinglisten entnommen werden kann, gibt es sehr wohl verschiedene Ansichten bezüglich der Interpretation der Semantik der RDF-Konzepte.

[8] Pierre-Antoine Champin hat mit seinem RDF Tutorial [32] im März 2000 ebenfalls eine Formalisierung in FOL angegeben. Die in der vorliegenden Arbeit vorgestellte Formalisierung ist jedoch umfassender, trennt die Repräsentationsebene von der Wissensebene (vgl. Reimer[149]) und vermeidet negative Fakten.

katenlogik erster Ordnung repräsentiert und die FOL-Interpretation der RDF-Spezifikationen wird der FOL-Interpretation der RDF-MT gegenübergestellt und diskutiert.

1.2.2 Implementierung der RDF-Semantik mittels Prolog

Die RDF-Schema-Spezifikation definiert Beschreibungsmittel (das RDF-Meta-Schema), die genutzt werden können, um RDF-Vokabulare – die RDF-Schemata genannt werden – zu erstellen. Das RDF-Meta-Schema ist also ein Meta-Vokabular zur Beschreibung von RDF-Schemata. Ein RDF-Schema definiert Konzepte, mit denen Aussagen über Sachverhalte in einem bestimmten Anwendungsbereich gemacht werden können. Ein RDF-Schema kann als einfache Ontologie aufgefaßt werden.[9] Das RDF-Meta-Schema selbst ist in RDF-Notation ausgedrückt.[10] Das bedeutet, daß die Anwendung des RDF-Meta-Schemas aufgrund der deutungsbedürftigen RDF-Syntax und Semantik zu deutungsbedürftigen RDF-Schemata und RDF-Meta-Schemata führt.

Durch die Verwendung der in dieser Arbeit angegebenen Formalisierung der RDF-Konzepte sind diese in ihrer Bedeutung eindeutig festgelegt. Um unabhängig von einem speziellen Wissensrepräsentations- bzw. Inferenz-System zu sein, wird zur Formalisierung der RDF-Semantik mathematische Logik (Prädikatenlogik erster Ordnung) als 'kleinster gemeinsamer Nenner' eingesetzt. Hiervon ausgehend können Implementierungen für verschiedene Inferenz-Systeme (z.B. Prolog, Datalog) entwickelt werden. In Kapitel 5 dieser Arbeit wird eine Implementierung der RDF-Semantik in Prolog vorgestellt. Ein Werkzeug – der RDF-Schema-Explorer – basiert auf dieser Implementierung. Der RDF-Schema-Explorer kann zur Abfrage und Validierung von RDF-Dokumenten eingesetzt werden.

Es gibt eine Reihe von Ansätzen [72, 112, 134, 145], die vorschlagen, Abfragen als Bestandteil von RDF-Dokumenten auszutauschen bzw. Abfragen in RDF-Syntax auszudrücken. Eine Standardisierung einer derartigen Abfragesprache ist jedoch noch nicht abgeschlossen. Bis zum Zeitpunkt einer entsprechenden Standardisierung wird daher aus pragmatischen Gründen in Kapitel 5 dieser Arbeit vorgeschlagen, Prolog-Code in RDF-Dokumente zu integrieren. Die XML/RDF-Syntax erlaubt die Verwendung von CDATA-Sektionen[11] (engl.

[9] Staab, Erdmann und Maedche schreiben hierzu in [163]: “To phrase the role of RDFS in knowledge engineering terminology, it allows to define a simple ontology that particular RDF documents may be checked against to determine consistency.”

[10] Das RDF-Meta-Schema ist deshalb auch ein RDF-Schema.

[11] CDATA-Sektionen werden in der XML-Spezifikation definiert. Siehe Abschnitt 2.7 in [25].

CDATA section), die Text[12] beinhalten, der nicht als Markup interpretiert wird. Das kann genutzt werden, um in CDATA-Sektionen Prolog-Syntax auszutauschen. Die durch ein RDF-Dokument erfaßten Aussagen werden zusammen mit dem, in den CDATA-Sektionen definierten, Prolog-Code durch den Prolog-Interpreter verarbeitet.

1.2.3 Vorschlag eines RDF-basierten Web-Engineering-Ansatzes

Grundlegende Idee des Ansatzes, ist die Anwendung von RDF auf das Web-Engineering. Der im Rahmen dieser Arbeit entwickelte Ansatz wurde in Auszügen bereits unter dem Namen eXtensible Web Modeling Framework (XWMF) [95, 96] vorgestellt.

Kern des XWMF ist ein Vokabular, das Beschreibungsmittel definiert, die zur Modellierung von Web-Anwendungen genutzt werden können. Das Vokabular wird in RDF-Syntax repräsentiert. Durch die Repräsentation der Modellierungsdaten und der Inhalte einer Web-Anwendung in RDF-Syntax können diese interoperabel ausgetauscht und von Maschinen in ihrer Bedeutung interpretiert werden. Das textuelle Darstellungsformat der Inhalte einer Web-Anwendung wird durch XWMF nicht vorgeschrieben. Beliebige Textformate (die dem *ISO-8859-1 Character Encoding Standard entsprechen*) bzw. beliebiges Markup wie z.B. HTML, XML, XHTML oder WML kann genutzt werden. Das Beschreibungsmodell ermöglicht die Erfassung der Modellierungsdaten und der Inhalte einer Web-Anwendung in einem einheitlichen Datenformat (als Datenformat dient die RDF-Syntax). Aus diesen Daten läßt sich die Web-Anwendung automatisch generieren. Durch die Möglichkeit der vollständigen Erfassung der Modellierungsdaten und der Inhalte, sowie der automatischen Generierung der Web-Anwendung, kann diese im Laufe ihrer Evolution konsistent zu den Beschreibungsdaten gehalten werden. Das Beschreibungsmodell ist erweiterbar. Neue Beschreibungsmittel können hinzugefügt werden, ohne daß die grundlegenden Datenstrukturen des Beschreibungsmodells geändert werden müssen. Der Web-Anwendungsentwickler hat so die Möglichkeit, für alle Phasen des Lebenszykluses einer Web-Anwendung die benötigten Beschreibungsmittel zu erstellen und diese anzuwenden (sofern sie nicht schon durch vorhandene Vokabulare bereit gestellt werden). Der Web-Anwendungsentwickler kann ebenso RDF-Vokabulare zur Erfassung der Verwaltungsdaten einer Web-Anwendung erstellen (z.B. organisatorische Verantwortlichkeit für Inhalte oder Verfallsdaten

[12] Unter Text werden hier Zeichensequenzen verstanden, wie sie durch die XML-Spezifikation erlaubt werden.

für Inhalte). Das Beschreibungsmodell stellt Beschreibungsmittel zur Definition von Konzepten, Meta-Konzepten, Meta-Meta-Konzepten usf. zur Verfügung. Das ermöglicht die Beschreibung einer Web-Anwendung auf verschiedenen Abstraktionsebenen. Infolgedessen können sowohl Top-Down- als auch Bottom-Up-Entwicklungsansätze angewendet werden. Die Beschreibungsmittel zwingen den Web-Entwickler nicht zur Definition einer bestimmten Granularität der Informationsstruktur. Der Web-Entwickler kann die Größe der Informationsstrukturen frei wählen. Das unterstützt evolutionäre Entwicklungsansätze, bei denen anfangs oft eine sehr grobkörnige Informationsstruktur gewählt wird. Zu einem späteren Zeitpunkt kann die Informationsstruktur verfeinert werden. Die Trennung verschiedener Modellierungsaspekte wird durch das Beschreibungsmodell unterstützt. Inhalte lassen sich getrennt von deren Darstellung und von deren strukturelle Anordnung beschreiben. Die Wiederverwendung von Modellierungskonzepten und Web-Anwendungskomponenten (verschiedener Abstraktionsebenen) wird unterstützt. Graphische Repräsentationsmöglichkeiten sind durch die Verwendung von RDF-Graphen bzw. durch die in Kapitel 6 vorgestellte graphische Notation gegeben. Alle Beschreibungsmodelle einer Web-Anwendung werden in RDF-Syntax abgespeichert, so daß sie generell von RDF-Werkzeugen bzw. Anwendungen des Semantic Web verarbeitet werden können.

1.2.4 Werkzeuge

Weitere Ergebnisse dieser Arbeit sind Werkzeuge, mit denen die in dieser Arbeit vorgeschlagenen theoretischen Ansätze auf deren Anwendbarkeit überprüft werden können. Außerdem wurden Werkzeuge entwickelt, die das Arbeiten mit RDF-Modellen vereinfachen sollen.

Auf die Notwendigkeit der Formalisierung des RDF-Datenmodells wurde bereits eingegangen. Als Formalismus wird in dieser Arbeit die Prädikatenlogik erster Ordnung zur Erfassung der Semantik des RDF-Datenmodells gewählt.

Für die gefundene Regelmenge wird eine in Zusammenarbeit mit Wolfram Conen erstellte Implementierung in Datalog angegeben (siehe Anhang 9.7). Die Datalog-Implementierung kann in Kombination mit dem vom W3C bereitgestellten RDF-Parser SiRPAC (Simple RDF Parser & Compiler) [159] und dem SiLRI (Simple Logic-based RDF Interpreter)[158] verwendet werden um RDF-Modelle zu validieren und zu befragen. Im Rahmen dieser Arbeit wurde dies mit der Version 1.14 des SiRPAC und Version 1.1.1 des SiLRI getestet.[13]

[13] Mit der Datalog-Implementierung ließ sich arbeiten, jedoch ist der SiLRI zumindest in der getesteten Version nicht frei von Fehlern.

Eine in Zusammenarbeit mit Wolfram Conen erstellte Implementierung der gefundenen Regelmenge[14] in Prolog[15] wird ebenfalls bereitgestellt (siehe Anhang 9.6). Diese kann in Kombination mit dem SWI-Prolog-RDF-Parser[16] genutzt werden, um RDF-Modelle zu validieren bzw. zu befragen.

Darauf aufbauend wurde in Zusammenarbeit mit Wolfram Conen eine Web-basierte Version implementiert (RDF-Schema-Explorer [38]), die ein Web-Interface zur Eingabe und Abfrage von RDF-Dokumenten bereitstellt. Neben den logischen Prädikaten, die die RDF-Semantik erfassen, bietet der RDF-Schema-Explorer zusätzliche logische Prädikate zur Abfrage eines RDF-Modells. Der RDF-Schema-Explorer steht online unter [38] zur Verfügung. Dem Benutzer werden durch das Web-Interface des RDF-Schema-Explorers Beispiele für RDF-Dokumente zur Verfügung gestellt, die Test- bzw. Demonstrationszwecken dienen.

Zur automatischen Generierung von Web-Applikationen aus RDF-Syntax kann der WebObjectComposer (siehe Abschnitt 6.4) genutzt werden. Der WebObject-Composer liest die Metadaten und Daten, die eine Web-Anwendung beschreiben, ein und erzeugt daraus die entsprechende Web-Anwendung. Der in Prolog geschriebene WebObjectComposer nutzt den SWI-Prolog-RDF-Parser zum Einlesen der RDF-Syntax. Mittels der vorgeschlagenen Prolog-RDF-Regeln (siehe Anhang 9.6) und des XWMF-Schemas (siehe Anhang 9.8) wird die Beschreibung einer Web-Anwendung analysiert und die entsprechende Implementierung der Web-Anwendung generiert. Die zur Verfügung gestellten logischen Prolog-Prädikate können genutzt werden, um Abfragen über den gesamten Beschreibungsraum der Web-Applikationen zu stellen. Die Resultate der Abfragen sind semantisch interpretierbar. Der XWMF-HTTP-Server integriert den WebObjectComposer und den SWI-Prolog-HTTP-Server (in einer leicht modifizierten Version). Die Web-basierte Demonstrationsanwendung [193] zeigt ein Beispiel einer Web-Anwendungsbeschreibung, aus der durch den WebObject-Composer die entsprechenden Web-Seiten generiert werden, die mittels Standard-Web-Browser abgerufen werden können. Zudem kann die Web-Anwendungsbeschreibung von standardkonformen (RDF-unterstützenden) Semantic-Web-Werkzeugen verarbeitet werden.

[14] Im strengen Sinne kann bei einer Implementierung in Prolog nicht von einer Menge von logischen Regeln gesprochen werden, da in Prolog-Syntax die Regeln als Folge angeordnet und interpretiert werden.

[15] Im Rahmen dieser Arbeit wird das SWI-Prolog [175] genutzt. Das SWI-Prolog ist open source und steht unter der GPL-Lizenz zur Verfügung.

[16] Der SWI-Prolog-RDF-Parser ist Bestandteil des SGML-Packages von SWI-Prolog ab der Version 4.0.0.

Eine frühe Version des WebObjectComposer und weitere Werkzeuge (siehe XWMF-Home [192]) liegen in der Scriptsprache XOTcl [133] vor. Zum Einlesen von RDF-Dokumenten wird eine modifizierte Version des von Gustaf Neumann und Uwe Zdun entwickelten RDF-Parsers xoRDF (Version 0.9 beta) [132] genutzt. Das in Zusammenarbeit mit Alexander Block entwickelte RDF-Handle [21] integriert den xoRDF-Parser und stellt Methoden zur Verarbeitung bzw. zur Abfrage von RDF-Modellen zur Verfügung. Zur Formulierung von Abfragen steht eine Abfragesprache zur Verfügung, die in ihrer Notation an SQL angelehnt wurde, jedoch auf die Abfrage von RDF-Modellen ausgerichtet ist. Die XOTcl-Version des WebObjectComposer nutzt das RDF-Handle, um eine in RDF-Syntax vorliegende Beschreibung einer Web-Anwendung zu verarbeiten und daraus die entsprechende Implementierung der Web-Anwendung zu generieren. Ein Entwickler kann den Status einzelner Elemente einer Web-Anwendung mit dem WebObjectBrowser[17] abfragen. Vor dem Erzeugungsprozeß der Web-Implementierung liegen die Elemente einer Web-Anwendung als XOTcl-Klassen bzw. -Objekte vor. Die Klassen- bzw. Instanz-Hierarchie kann ein Entwickler mittels des WebObjectBrowsers darstellen. Der gemeinsam mit Block entwickelte RDF-Editor GraMTor dient der graphischen Erstellung von RDF-Modellen. Die graphische Repräsentation eines RDF-Modells kann in verschiedenen Formaten abgespeichert werden. Als Speicherformate stehen, neben anderen, die durch die RDF-Spezifikationen definierte RDF-Syntax und die Triple-Notation zur Verfügung. Die graphischen Elemente des WebObject-Browsers und des GraMTor wurden mittels $W_a f^e$[131] entwickelt.

Der in Zusammenarbeit mit Ralf Goertzen entwickelte VisiRDF [180] ist ebenfalls ein RDF-Editor, der die graphische Erstellung von RDF-Modellen ermöglicht. Implementiert ist der VisiRDF in SWI-Prolog [175] und dem XPCE-Toolkit [191]. Zum Einlesen von RDF-Dokumenten wird der SWI-Prolog-RDF-Parser genutzt.

XRDF [46, 45] ist ein im Rahmen dieser Arbeit entwickeltes alternatives Datenmodell zur RDF und ein dazugehöriger Syntax-Vorschlag (siehe folgenden Abschnitt). Eine von Wolfram Conen entwickelte Web-basierte Version des RDF2XRDF-converter [146] ermöglicht die Transformation von RDF-Dokumenten in XRDF-Dokumente.

[17] Der WebObjectBrowser ist eine angepaßte Version des XOTcl-Werkzeugs Inspector, welches durch Fredj Dridi entwickelt wurde.

1.2.5 XRDF – ein alternativer Vorschlag zu RDF

Die in der RDF-Spezifikation angegebene Syntax ist umstritten (siehe die Diskussionen in den RDF-Mailinglisten [107, 108, 106, 109, 110, 111]). Ein wesentlicher Kritikpunkt ist, daß die "Integration" von RDF in XML- bzw. XML-Schema-Anwendungen nur unter bestimmten Bedingungen möglich ist. Dies resultiert daraus, daß die RDF-Syntax zwar eine XML-Syntax ist, die aber nicht validierbar ist, da *keine* XML-DTD (und kein XML-Schema) für die RDF-Syntax existieren kann. Weiterer Kritikpunkt ist, daß nicht eindeutig definiert ist, wie ein RDF-Dokument in einen RDF-Graph oder in eine Menge von Statements (Triple) abzubilden/zu transformieren ist. Dies kann zu unterschiedlichen semantischen Interpretationen eines RDF-Dokuments führen. Außerdem gilt die RDF-Syntax als zu komplex, was das Abfassen von RDF-Dokumenten und die Entwicklung von RDF-Werkzeugen erschwert. Das W3C hat aufgrund der in den RDF-Mailinglisten geäußerten Kritik beschlossen (vgl. [174]), daß eine neue einfachere XML-Syntax für RDF entwickelt werden soll.

Der in dieser Arbeit vorgestellte XRDF-Ansatz beinhaltet eine zur RDF-Syntax alternative XML-Syntax. Um die XRDF-Syntax einfach zu halten, wird bewußt eine sehr einfache XML-DTD definiert. Das dazugehörige Datenmodell wird in formaler Notation angegeben, um eine eindeutige Interpretierbarkeit sicherzustellen. Faktenbasen, die in Datenstrukturen des hier entwickelten Datenmodells abgebildet sind, können in Datenstrukturen des RDF-Datenmodells abgebildet werden (und vice versa).[18] Dabei kommt es zu keinem Informationsverlust, wie im Abschnitt 7.3 bewiesen wird.

Für die vorgeschlagene XML-Syntax ist eindeutig definiert, wie syntaktische Elemente in Datenstrukturen des Datenmodells zu überführen sind. Zusätzlich wird eine an der graphischen RDF-Notation angelehnte Notation entwickelt, die die Komposition und Dekomposition von Graphen untestützt. Diese Eigenschaft ist besonders bei der werkzeugunterstützten Entwicklung von Faktenbasen nützlich. Die Abbildung der Datenstrukturen in der graphischen Notation ist ebenfalls eineindeutig definiert.

Der XRDF-Ansatz wurde bereits aufgegriffen. Dies resultierte in der eingeladenen Veröffentlichung des Ansatzes (vgl. [46]) im Buch *The emerging Semantic-Web* [50]. Außerdem wurden dem Ansatz zugrundeliegende Ideen von Drew McDermott et al. in weiteren weiteren Arbeiten (vgl. z.B. [118]) verwendet.

[18] Dies bedeutet, daß Metadaten, die in der XRDF-Syntax notiert werden, in die RDF-Syntax abgebildet und damit von schon vorhandenen RDF-Werkzeugen verarbeitet werden können.

1.3 Wissenschaftstheoretische Einordnung zur Wirtschaftsinformatik

Zur wissenschaftstheoretischen Einordnung der Ergebnisse dieser Arbeit soll im folgenden untersucht werden, welches die Forschungsgegenstände und die Erkenntnisziele der Wirtschaftsinformatik sind.

Im Jahr 1994 wurde eine Untersuchung (vgl. [99, 98]) durchgeführt, deren Ziel es war, die zentralen Forschungsmethoden und Theoriekerne der Wirtschaftsinformatik der nächsten zehn Jahren zu identifizieren. Dazu wurde die Delphi-Methode[19] und die AHP-Methode (Analytic Hierarchy Process)[20] miteinander kombiniert. Die Autoren der Studie ziehen folgendes Fazit:

> Als wesentliches Untersuchungsergebnis ist - bei aller Zurückhaltung wegen der angesprochenen methodischen Probleme - festzuhalten, daß im Gegensatz zu einem stabilen Konsens bezüglich der zukünftigen Forschungsgegenstände bezüglich der Forschungsmethoden und der Theoriegrundlagen keine einheitliche "Vision" über die Positionierung der Wirtschaftsinformatik in der Wissenschaftswelt der nächsten 10 Jahre entwickelt wird. Vielmehr ist ein fragmentiertes und bisweilen auch lückenhaftes Bild der zukünftigen Forschungsmethoden und Theoriekerne der Wirtschaftsinformatik festzustellen. [21]

Weiter stellt Teubner in [176] eine Sammlung von Aussagen zur Wirtschaftsinformatik als Wissenschaft zusammen. Die Aussagen sind Veröffentlichungen entnommen, die von Vertretern der Wirtschaftsinformatik verfaßt wurden. Zusammenfassend ist daraus zu entnehmen, daß bisher kein Konsens über die Definition des Forschungsgegenstandes und der Forschungsmethode der Wirtschaftsinformatik existiert.

In [102] schreibt Lehner 1999:

[19] Die Delphi-Methode ist eine strukturierte schriftliche Befragung von Mitgliedern eines Panels.

[20] Der AHP-Ansatz ist eine Methode zur Entscheidungsunterstützung, mit der Abhängigkeiten von miteinander verbundenen Größen einer Entscheidung hierarchisch analysiert werden.

[21] Anmerkung der Autoren der Untersuchung [99, 98] zur inhaltlichen Interpretation der Ergebnisse: Grundsätzlich kann und muß man die Frage stellen, inwieweit 30 ausgewählte Personen die Meinung einer Community widerspiegeln. Allerdings handelt es sich bei den beteiligten Experten um Vertreter derjenigen Gruppen von Personen, die die Zukunft "gestalten", d.h. mit hoher Wahrscheinlichkeit die von ihnen erwarteten Änderungen in der Community durchsetzen werden.

> Zusammenfassend läßt sich festhalten, daß die Wirtschaftsinformatik, wie viele andere junge Wissenschaften, unter einem Mangel an theoretisch fundierten Grundlagen leidet. Die vorhandenen Grundlagen sind vielfach noch unzureichend offengelegt und damit auch wissenschaftlich schwer diskutierbar.

In seinen Anmerkungen zum Status der Wirtschaftsinformatik macht Frank in [65] die Aussage:

> Die Wirtschaftsinformatik hat sich im Laufe der Zeit zu einer Veranstaltung mit erstaunlich vielen Facetten entwickelt. Einschlägige Arbeiten reichen von soziologisch ausgerichteten Untersuchungen über betriebswirtschaftliche und mikroökonomische Ansätze bis hin zu Forschungsvorhaben, die weit in das Gebiet der Informatik hineinreichen.

Es ist also festzustellen, das noch nicht klar abgegrenzt ist, was Forschungsgegenstand, Forschungsmethodik oder Theorie der Wirtschaftsinformatik ist. Es gibt nicht eine(n) einzelne(n) Forschungsgegenstand, Forschungsmethode und Theorie der Wirtschaftsinformatik, sondern jeweils mehrere.

Um eine wissenschaftstheoretische Einordnung von Forschungsergebnissen (inklusive der hier vorliegenden Arbeit) als zur Wirtschaftsinformatik gehörig vornehmen zu können, werden im folgenden einige Definitionsvorschläge zum Forschungsgegenstand der Wirtschaftsinformatik diskutiert.

Die von den Teilnehmern der oben erwähnten Befragung (vgl. [99, 98]) vorgeschlagenen zentralen Forschungsgegenstände werden in der Studie in einer Rangfolge angeordnet. Auf den höchsten Rängen werden folgende Forschungsgegenstände genannt:

1. Wissenschaft mit starkem Bezug zur Organisationslehre, die versucht, den Aufbau und die Abläufe sozio-technischer Systeme zu beschreiben und zu optimieren.

2. Funktionale Betriebswirtschaftslehre, die vornehmlich die Funktion der betrieblichen Datenverarbeitung/Informationsverarbeitung im Unternehmen erforscht (z.B. Informationsverarbeitung als Querschnittsfunktion)

3. Informationswissenschaft, die versucht, die Ökonomie des Leistungsfaktors Information und seine zielgerichtete Bereitstellung zu erforschen.

4. Innovationswissenschaft, die sich mit der Definition von Anforderungen an neue Informations- und Kommunikationstechniken beschäftigt und die daraus resultierenden Produkte und Verfahren in die Unternehmen einführt.

Ziel der vorliegenden Arbeit ist die wissenschaftliche Auseinandersetzung mit RDF, insbesondere zu untersuchen, inwieweit sich RDF zum Austausch semantisch interpretierbarer Information eignet sowie Erweiterungsvorschläge und Verbesserungsvorschläge zur Erreichung diese Ziels zu liefern. Weiter wird ein Ansatz vorgestellt, der zeigt wie (erweitertes) RDF zur Beschreibung/Modellierung/Konstruktion Web-basierter Informationssysteme eingesetzt werden kann.

RDF kann als Grundlage des Semantic Web gesehen werden. Eine Anforderung an das Semantic Web ist die verbesserte (zielgerichtete) Bereitstellung von Information. Die Anwendung von wissensbasierten Ansätzen auf das Web soll zu neuen und innovativen Lösungen führen. Das Semantic Web und (allgemeiner) Web-basierte Informationssysteme können als sozio-technische Systeme verstanden werden, die der Anbindung externer Kommunikationspartner und der innerbetrieblichen Datenverarbeitung/Informationsverarbeitung dienen. Insofern kann argumentiert werde, daß ein Bezug zu den unter (1)-(4) aufgeführten Forschungsgegenständen[22] besteht und die vorliegende Arbeit der Wirtschaftsinformatik zugeordnet werden kann.

Weiter definieren Lehner/Hildebrand/Maier folgendes (siehe S. 6 in [103])

> Die Wirtschaftsinformatik ist eine Realwissenschaft, da Phänomene der Wirklichkeit ("IKS in Wirtschaft und Verwaltung") untersucht werden. Die Wirtschaftsinformatik ist ebenso eine Formalwissenschaft, da die Beschreibung, Erklärung, Prognose und Gestaltung der IKS der Entwicklung und Anwendung formaler Beschreibungsverfahren und Theorien bedürfen. Die Wirtschaftsinformatik ist weiterhin eine Ingenieurwissenschaft, da insbesondere die Gestaltung von IKS eine Konstruktionssystematik verlangt.
>
> ...
>
> Ziel wissenschaftlicher Untersuchungen in der Wirtschaftsinformatik ist die Gewinnung von Theorien, Methoden, Werkzeugen und intersubjektiv nachprüfbaren Erkenntnissen über/zu IKS ...

[22] Soweit bei semantischer Uneindeutigkeit der verwendeten Begriffe überhaupt entscheidbar.

Gewonnene Ergebnisse der vorliegenden Arbeit sind Theorien[23] (z.B. die FOL-Interpretation von RDF), Methoden[24] (z.B. die Anwendungsvorschrift zum Einsatz von RDF zur Beschreibung von Web-basierten Informationssystemen), Werkzeuge (z.B. der WebObjectComposer) und intersubjektiv nachprüfbare Erkenntnisse über/zu Informations- und Kommunikationssystemen (z.B. da FOL verwendet wird, ein allgemein anerkannter Formalismus, der unabhängig von einer subjektiven Interpretation der verwendeten Symbole ist). Insofern werden die genannten Ziele wissenschaftlicher Untersuchungen in der Wirtschaftsinformatik durch die vorliegende Arbeit erreicht.

1.4 Forschungsmethodik und Aufbau der Arbeit

In diesem Abschnitt soll auf die in dieser Arbeit verwendete Forschungsmethodik eingegangen werden. Dazu wird zunächst untersucht, was geeignete Forschungsmethoden der Wirtschaftsinformatik sind.

Bezüglich der in der Wirtschaftsinformatik zu verwendenden Forschungsmethoden ergibt die Delphi-Untersuchung [99, 98] von König et al.:

> Die Abschlußauswertung ergibt folgendes Bild der Einschätzung der Teilnehmer, mit welchem relativen Gewicht jede Methodenklasse von der Wirtschaftsinformatik angewendet werden soll.[25]

Methodenklassen	%-Anteile[26]
- konstruktive Methoden, die überwiegend deduktionsgetrieben sind und primär zur Veränderung von Sachverhalten dienen	60%
- empirische Methoden, die überwiegend induktionsgetrieben sind und primär dem Zweck der Überprüfung von Theorien zur Erklärung gegebener Sachverhalte dienen	40%

[23]Eine Theorie kann verstanden werden als "System von Aussagen"

[24]Eine Methode kann verstanden werden als Verfahrensvorschrift zur Konstruktion von Informationssystemen.

[25]Die beiden zur Auswahl stehenden Methodenklassen wurden von den Moderatoren der Untersuchung vorgegeben.

[26]Anmerkungen der Autoren der Untersuchung: Sechs der 30 Mitglieder des Panels sind der Auffassung, daß die Vorteilhaftigkeit bestimmter Methoden nur vor dem Hintergrund konkreter Fragestellungen behandelt werden kann. Drei andere Teilnehmer stellen fest, daß man bei Anwendung jeder Methode deduzieren und induzieren muß. Ein Teilnehmer merkt

Im Tagungsband “Wirtschaftsinformatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven” [6] wird die Wirtschaftsinformatik ebenfalls als empirische als auch konstruktivistische Forschung dargestellt (siehe S. 159-376).

Zum Positivismus (Empirismus) halten Lehner/Hildebrand/Maier (an anderer Stelle) fest (siehe S. 25 in [103]):

> Die heute am weitesten verbreitete wissenschaftstheoretische-methodische Richtung ist der **kritische Rationalismus** (eng verbunden mit dem Namen Karl Popper[27]). Die im kritischen Rationalismus vorgeschlagene Forschungsweise besteht aus einer wechselseitigen Verknüpfung und Folgebeziehung von Vermutung und Zurückweisung, Spekulation und Prüfung (vgl. Grochla 1978). Die Hauptforderung dieser Richtung besteht darin, daß wissenschaftliche Aussagen informativ, d.h. empirisch gehaltvoll sein müssen. Der kritische Rationalismus ist am Wissenschaftsmodell der Naturwissenschaften entwickelt worden. Er ist unter Vernachlässigung einiger der relativ hohen Anforderungen auch als methodologische Grundlage für die Forschung in verschiedenen Bereichen der Betriebswirtschaftslehre (z.B. Organisationslehre) akzeptiert worden (vgl. Grochla 1978, Kieser/Kubicek 1978). Unter den gleichen Voraussetzungen kann der kritische Rationalismus auch als methodologische Grundlage für die Forschungsaktivitäten der Wirtschaftsinformatik dienen.

Weiter definieren Lehner/Hildebrand/Maier folgendes (siehe S. 7 in [103])

> Wirtschaftsinformatiker wenden Methoden und Werkzeuge aus den Real,- Formal- und Ingenieurwissenschaften an und entwickeln diese weiter. Der soziotechnische Erkenntnisgegenstand der Wirtschaftsinformatik verlangt, daß bei der Auswahl und der Kombination der anzuwendenden Methoden/Werkzeuge nicht nur Fragen der technischen Effizienz, sondern auch die ökonomische und soziale Einsetz-

an, daß empirische Methoden zur ’Überprüfung von Theorien’ präsupponieren, daß die Wirtschaftsinformatik über Theorien im strengen wissenschaftstheoretischen Verständnis des Theoriebegriffs verfügt. Da dies allenfalls in rudimentären Ansätzen zutrifft, dürfte nach Einschätzung dieses Teilnehmers ein mehr als bescheidenes Gewicht für empirische Methoden zur Theorieüberprüfung eher dem Wunschdenken der Zunft der Wirtschaftsinformatiker als der epistemischen Qualität ihres Wissensfundus entsprechen.

[27]Werke von Poper sind z.B.: *Logik der Forschung* [139] und *Die beiden Grundprobleme der Erkenntnistheorie* [138]

> barkeit (einschließlich der Akzeptanzsicherung der verschiedenen gesellschaftlichen Gruppen[28]) beachtet wird.

Holl faßt die Wirtschaftsinformatik ebenfalls als empirische Wissenschaft auf. In [81] schreibt er:

> Die Wirtschaftsinformatik läßt sich (zumindest in wesentlichen Teilen) selbst als empirische Wissenschaft auffassen, da sich als tertium comparationis mit den Naturwissenschaften gerade die essentiellen empirischen Verfahren herausstellen: Beobachtung, Modellbildung und Modellformalisierung. (Formale Modelle sind die wesentlichen Formen empirisch-wissenschaftlicher Erkenntnisse.) Wegen dieser *Vergleichbarkeit* ist es sinnvoll, erkenntnistheoretische Überlegungen der Naturwissenschaften auch in der Wirtschaftsinformatik vorzunehmen.

Zum Konstruktivismus halten Lehner/Hildebrand/Maier fest (siehe S. 25 in [103]):

> Der **Konstruktivismus** nimmt eine Trennung zwischen Ontologie und Erkenntnistheorie vor (vgl. Floyd 1989, Schmidt 1987). Folgt man der konstruktivistischen Sichtweise, dann kommt Erkenntnis und damit auch jede Modellbildung durch "Konstruktion" zustande. Es geht demnach also nicht um die Abbildung, die einer vorgegebenen Realität zu entsprechen hat, sondern um eine Konstruktion der Erkenntnisse in einer Form, die für unsere Zwecke brauchbar ist. Jede Erkenntnis ist damit an einen Beobachter gebunden, der mit anderen Beobachtern über gemeinsam zugängliche Bereiche Konsens herstellen kann. Eng damit zusammenhängend ist das Konzept der Perspektiven (vgl. Floyd 1989, vgl. auch Frank 1994).

Zusammenfassend kann also festgestellt werden, daß zur Erlangung von Erkenntnisgewinn in der Wirtschaftsinformatik sowohl Empirismus als auch Konstruktivismus angewendet werden kann.

[28] Akzeptanzsicherung in der wissenschaftlichen Gesellschaft ist eng mit dem Begriff Paradigma verbunden. Kuhn (vgl. [100]) beschreibt Paradigma als "das, was den Mitgliedern einer wissenschaftlichen Gemeinschaft gemeinsam ist, und umgekehrt besteht eine wissenschaftliche Gemeinschaft aus Menschen, die ein Paradigma teilen".

Merkmal wissenschaftlicher Methodik ist, daß eine Aussage erst dann als als wissenschaftlich gesichertes Wissen gilt, wenn sie begründbar (als wahr erachtet werden kann) und intersubjektiv nachprüfbar ist.

Aufbau der Arbeit

Die Aufgabe der Begründung wissenschaftlicher Aussagen ist nicht zu lösen von der Frage nach der Bedeutung der verwendeten sprachlichen Mittel. In der vorliegenden Arbeit werden deshalb in Kapitel 2 und 3 die für die vorliegende Arbeit wesentlichen Begriffe eingeführt. Im Kapitel 2 werden die Grundlagen Web-basierter Informationssysteme erläutert und im Kapitel 3 erfolgt die Darstellung des Resource Description Framework (RDF).

Zur Beantwortung der Fragestellung, ob RDF zum Austausch semantisch interpretierbarer Information genutzt werden kann, erfolgt in Kapitel 3 eine Analyse und eine Diskussion von RDF. Die gewonnenen Erkenntnisse dienen als Grundlage einer Formalisierung von RDF in Prädikatenlogik erster Ordnung.

In Kapitel 4 wird die entwickelte Formalisierung angegeben. Die RDF-Spezifikationen selbst definieren per "Konstruktion" ein Datenmodell und eine Syntax. Die Definitionen sind allerdings weitgehend informal abgefaßt, weshalb die Forderung nach intersubjetiver Erfahrbarkeit nicht gegeben ist. Deshalb kann die in dieser Arbeit vorgestellte Formalisierung nur *eine* Interpretation von RDF sein. Diese Interpretation ist allerdings, da in einem allgemein anerkannten Formalismus abgefaßt, intersubjektiv erfahrbar. Eine Akzeptanzsicherung wurde angestrebt, indem die Formalisierung der RDF-Forschungsgemeinschaft zugänglich gemacht wurde [40, 39]. Die dadurch ausgelöste Kritik [13] bzw. die entstandene Diskussion [94] werden in der vorliegenden Arbeit berücksichtigt. Implementierungen der Formalisierung in Datalog und Prolog ermöglichen die Überprüfung der praktischen Einsetzbarkeit.

Kapitel 5 erläutert die Implementierung der RDF-Semantik in Prolog. Außerdem wird ein Ansatz zur Integration von Prolog-Code in RDF-Dokumente vorgeschlagen. Die Ergebnisse wurden der RDF-Forschungsgemeinschaft zugänglich gemacht und zur Diskussion gestellt [41, 43]. Eine Web-basierte Implementierung [38] ermöglicht den Test der hier vorgeschlagenen Technik.

Die Anwendung von RDF zur Beschreibung von Web-basierten Informationssystemen wird in Kapitel 6 vorgeschlagen. Ein RDF-Schema definiert die Konzepte mit denen Web-basierte Informationssystem beschrieben werden können. Eine entsprechende RDF-Beschreibung einer Web-Anwendung kann automatisch in eine Web-basierte Implementierung überführt werden. Zur Überprüfung des Ansatzes können die im Rahmen dieser Arbeit entwickelten Werkzeuge (speziell der WebObjectComposer) herangezogen werden, ebenso wie das gezeigte Beispiel einer Web-Anwendungsbeschreibung. Die Anforderungen, die an diesen Ansatz gestellt werden, sind in Abschnitt 1.1 aufgeführt. Die Überprüfung der Erfüllung der Anforderungen wird in Abschnitt 6.6 vorgenommen. Zur Akzeptanzsicherung wurde der Ansatz der Forschungsgemeinschaft zur Überprüfung vorgelegt [95, 96, 192].

Ein alternativer Syntax- und Modellvorschlag zu RDF wird in Kapitel 7 vorgestellt. In der RDF-Forschungsgemeinschaft gemachte Erfahrungen (die z.B. in den verschiedenen RDF-Mailinglisten und entsprechenden Forschungsarbeiten dokumentiert sind) wurden aufgegriffen, um eine Syntax und ein Modell zu entwickeln. Viele der Probleme, die die RDF-Syntax bzw. das RDF-Datenmodell bereiten, werden vermieden. Syntax und Modell werden in formaler Notation und somit intersubjektiv erfahrbar notiert. Der Vorschlag wurde der RDF-Forschungsgemeinschaft zugänglich gemacht und diskutiert [46, 45]. Das hier vorgeschlagene Modell mit der dazugehörigen Syntax läßt sich in RDF-Syntax überführen. Eine entsprechende Implementierung (RDF2XRDFConverter [146]) ermöglicht die praktische Überprüfung.

Die entwickelten Formalisierungen, Beispielanwendungen und Werkzeuge sollen zur intersubjektiven Überprüfbarkeit und Wiederholbarkeit der gewonnenen Erkenntnisse dienen, welche zudem falsifizierbar[29] sind. Die Methodik der Arbeit ist konstruktivistisch geprägt, jedoch werden die Ergebnisse der Arbeit auch in der Realwelt erprobt.

[29] Zum Begriff *falsifizierbar* (vgl. [151]): Unter Falsifikation versteht man den allgemeinen Nachweis der Falschheit einer Aussage. Also die Widerlegung eines empirischen Allsatzes durch ein Gegenbeispiel. So wird z.B. die Behauptung "Alle Raben sind schwarz" durch einen nichtschwarzen Raben falsifiziert. Popper meinte, daß in den Wissenschaften nicht nur ein isoliertes Gegenbeispiel genügt, das man durch Zusatzhypothesen vielleicht erklären könnte, sondern daß eine konkurrierende These zur Verfügung stehen muß, die mit dem Gegenbeispiel verträglich ist. Falsifikation ist also die Bewährung einer Gegenhypothese. Von ihr ist die Falsifizierbarkeit, d.h. die Widerlegbarkeit, zu unterscheiden. Nach Popper sind nur falsifizierbare, nicht aber falsifizierte Sätze in den Wissenschaften zuzulassen.

2 Web-basierte Informationssysteme

In den folgenden Abschnitten werden technologische Grundlagen Web-basierter Informationssystem erläutert. Es wird jedoch keine umfassende Erläuterung gegeben. Vielmehr geht es um die Einordnung der grundlegenden Web-Techniken und die Einführung von Begriffen, die im weiteren Verlauf der Arbeit verwendet werden.

In dem Artikel *As We May Think* [30] beschreibt Vannevar Bush das Memex-System. Es besteht aus Informationsknoten, die über Referenzen miteinander verbunden sind. Das ist das grundlegende Modell von Hypertext-Systemen.[1] Als technologische Basis nutzte Bush (anfangs) Mikrofilm, um Inhalte abzuspeichern. Das grundlegende Konzept ist jedoch auf elektronische Speichersysteme übertragbar. Es wurden eine Reihe verschiedener Hypertext-Systeme entwickelt. Ein Versuch, die generellen Abstraktionen, die in Hypertext-Systemen verwendet werden, zu erfassen, ist das *Dexter Hypertext Reference Model* [73]. Das WWW kann ebenfalls als Hypertext-System aufgefaßt werden. Tim Berners-Lee stellt im Artikel *The World-Wide Web* [9] die Grundlagen des WWW vor. Er beschreibt ein Rechnernetz, in dem Daten gespeichert werden, die über Hyperlinks[2] weitere Daten referenzieren.

2.1 Grundlegende Web-Spezifikationen im Überblick

Zur Realisierung des WWW wurden Systeme entwickelt, die Standards von Standardisierungremien, wie z.B. der IETF [91] und der ISO [88] implementieren. Außerdem wurde das World Wide Web Consortium (W3C) gegründet,

[1] Die Begriffe Hypermedia und Hypertext werden oft synonym benutzt. Eine Abgrenzung kann jedoch vorgenommen werden. Hypermedia-Systeme verarbeiten zusätzlich zu Hypertext weitere Datenformate, wie z.B. Bilder, Audiosequenzen, Videosequenzen. In dieser Arbeit werden die Begriffe Hypertext und Hypermedia synonym verwendet, sofern auf die Unterscheidung nicht explizit hingewiesen wird.

[2] Der Begriff Hyperlink bzw. Link wird in Abschnitt 2.4 erläutert.

das Standards speziell für die Entwicklung von Web-basierten Informationssystemen erstellt. Abbildung 2.1 zeigt Spezifikationen im Überblick, die für die Entwicklung von Web-basierten Informationssystemen genutzt werden. In den folgenden Abschnitten wird auf die Merkmale und das Zusammenspiel der verschiedenen Techniken eingegangen.

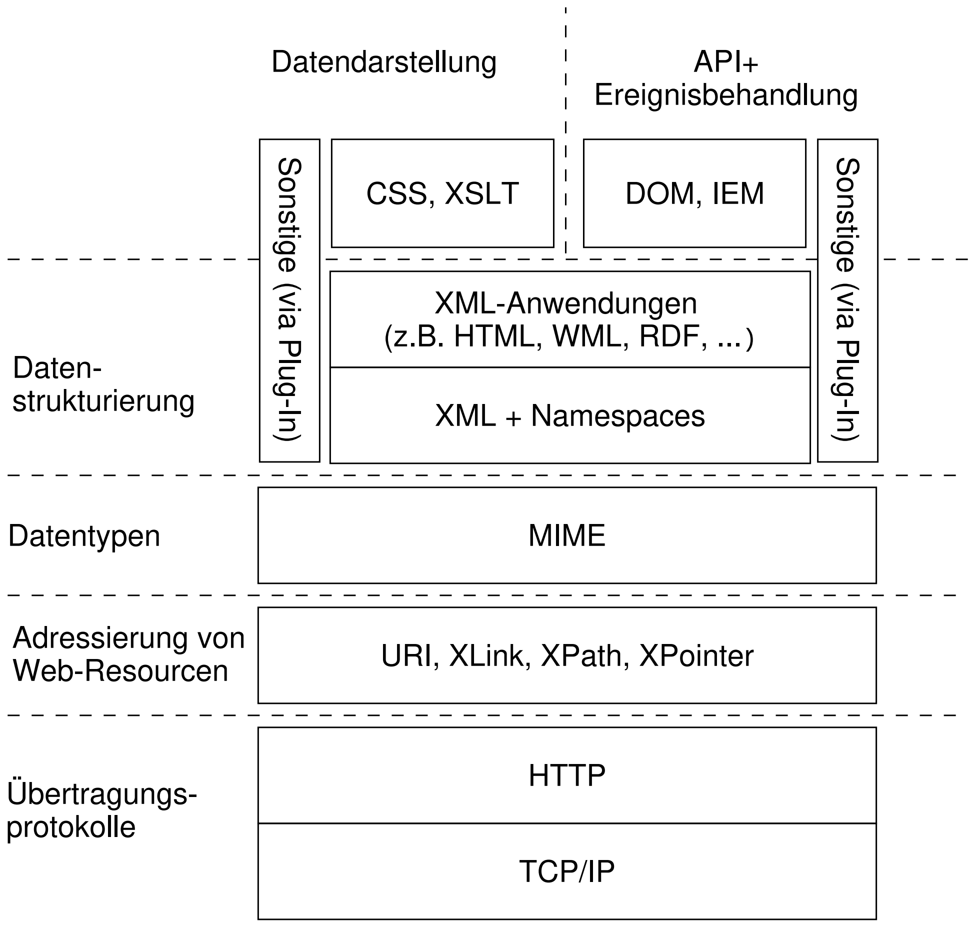

Abbildung 2.1: Grundlegende Web-Spezifikationen

2.2 Architektur

Das **H**yper**T**ext **T**ransfer *Protocol* (HTTP) [64] definiert den Aufbau von Nachrichten (engl. message) und Methoden, mit denen Informationen zwischen einem Web-Client[3] und einem Web-Server ausgetauscht werden können. Ein

[3]In der HTTP-Spezifikation [64] wird ein Web-Client auch User Agent (UA) genannt. Ein User-Agent ist ein Rechnerprozeß, der eine Anfrage an einen Web-Server richtet. Beispiele für einen UA sind Browser, Spider, Robot, Proxy.

Web-Client sendet eine Anfrage-Nachricht (engl. request message) an einen Web-Server, der eine Antwort-Nachricht (engl. response message) zurücksendet und wieder in den Ursprungszustand zurückgeht (zustandsloses Client-Server-Modell).

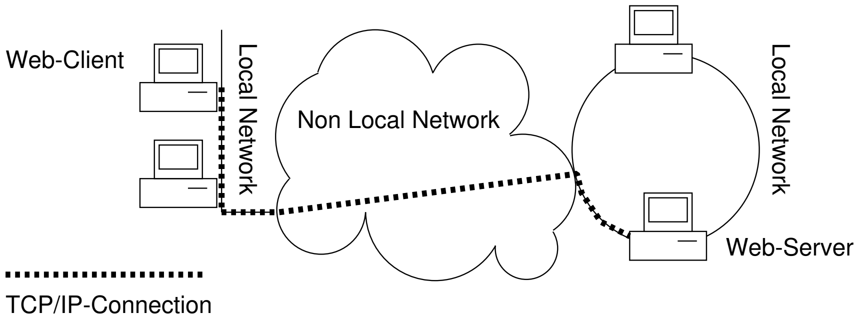

Abbildung 2.2: Einsatz von TCP/IP zum Datentransport mittels heterogenen Rechnernetzen

Zum Aufbau einer bidirektionalen, virtuellen Verbindung zwischen Client und Server werden das Transmission Control Protocol [141] und das Internet Protocol [140] genutzt. TCP/IP ermöglicht den Zusammenschluß von Rechnernetzwerken, die verschiedene physikalische Übertragungsmedien nutzen. Abbildung 2.2 zeigt, stark vereinfacht, den Zusammenschluß verschiedener Rechnernetze mittels TCP/IP. HTTP nutzt die Funktionalität von TCP/IP, um Anfrage- und Antwort-Nachrichten zu übertragen. Die Beziehungen zwischen den verschiedenen Protokollen kann in einem Schichtenmodell dargestellt werden. Abbildung 2.3 zeigt das Web-Protokoll-Modell.

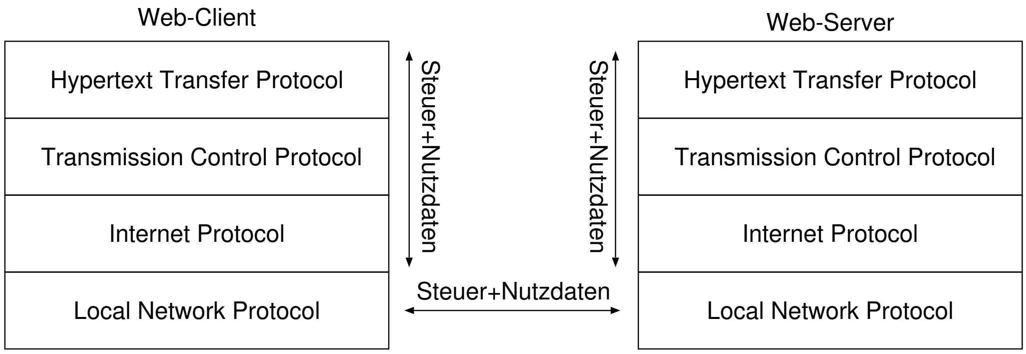

Abbildung 2.3: Web-Protokoll-Modell

Abbildung 2.4 zeigt den Aufbau einer Request-Message anhand eines Beispiels. HTTP definiert verschiedene Request-Methoden. Im Beispiel wird mit der `GET`-Methode eine Web-Resource (`/index.html`) von einem Web-Server angefordert, zu dem vorher eine TCP/IP-Verbindung aufgebaut wurde. Ein Header enthält

Header-Felder mit dazugehörigen Header-Werten. Header enthalten Metadaten, mit denen Web-Clienten und Web-Server ihr Kommunikationsverhalten steuern bzw. Web-Resourcen beschreiben.

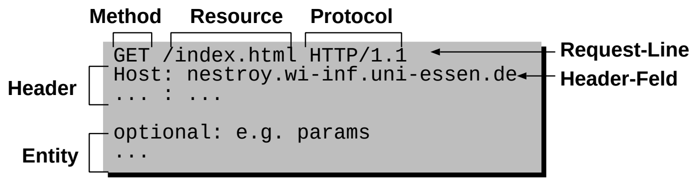

Abbildung 2.4: HTTP-Request-Message

Abbildung 2.5 zeigt exemplarisch den Aufbau einer Response-Message. Der Web-Server informiert den Web-Client mittels Status-Codes über den Erfolg der Verarbeitung einer Request-Message.[4] Im Beispiel wird ein Web-Client durch den Status-Code `200` informiert, daß die Request-Message vom Web-Server erfolgreich verarbeitet werden konnte. Der Header enthält das Header-Feld `Content-Type`. Als Wert dieses Feldes kann ein MIME-Type[5] angegeben werden, der das Datenformat der Entity angibt. Die Entity enthält die Nutzdaten einer Message. Im Beispiel wird der MIME-Type `text/html` verwendet, was bedeutet, daß die Nutzdaten im HTML-Format vorliegen.

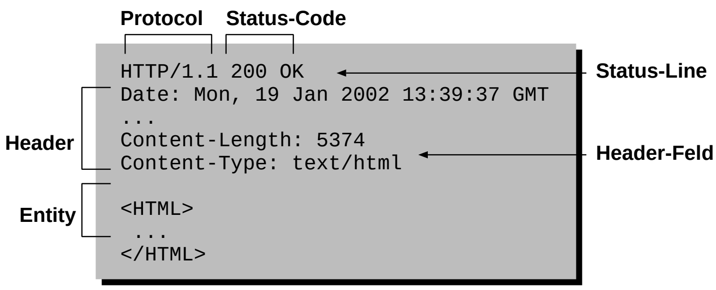

Abbildung 2.5: HTTP-Response-Message

Daten, die durch einen Uniform Resource Locator (URL)[6] [17, 18, 63] referenziert werden können, werden als Web-Resourcen bezeichnet.[7] Abbildung 2.6 zeigt einen URL, der Daten mit dem Resource-Namen `/index.html` referenziert, die vom Web-Server `www.server.de` angefordert werden. Der Web-Server

[4] Diese Aussage ist stark vereinfacht. Darüber hinaus kann ein Web-Server über Status-Codes das weitere Kommunikationsverhalten eines Web-Client steuern. Für eine vollständige Erläuterung der Status-Codes siehe [64].

[5] MIME: Die Multipurpose Internet Mail Extensions werden in den RFCs 2045-2049 definiert.

[6] Für weitere Informationen zur Adressierung von Web-Resourcen siehe auch Abschnitt 2.4.

[7] Diese Definition folgt der Definition einer Web-Resource in den RDF-Spezifikationen.

liest die entsprechende Web-Resource aus dem Resource-Speicher[8] und sendet sie zum Web-Client.

Abbildung 2.6: Beispiel eines URLs

Referenziert ein URL ein CGI-Script, übergibt der angesprochene Web-Server über das *Common Gateway Interface* [36] Daten an ein CGI-Script, das diese verarbeitet und eine Web-Resource eines bestimmten MIME-Types generiert. Die Web-Resource wird über den Web-Server an den Web-Client ausgeliefert, der diese verarbeitet (z.B. am Bildschirm darstellt). Kann der Web-Client das Datenformat, das durch den MIME-Type angegeben wird, nicht verarbeiten, wird die Web-Resource, sofern vorhanden, einem Plug-In [57] zur weiteren Verarbeitung übergeben. Abbildung 2.8 illustriert den Abruf einer Web-Resource, die durch ein CGI-Script generiert wird. Abbildung 2.7 zeigt ein Beispiel eines URLs, welcher ein CGI-Script referenziert, an das die Attribut-Wert-Paare aus dem Query-String übergeben werden.

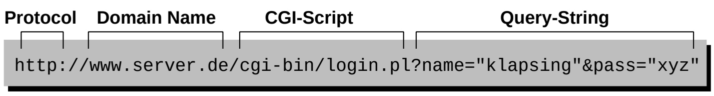

Abbildung 2.7: URL der ein CGI-Script referenziert

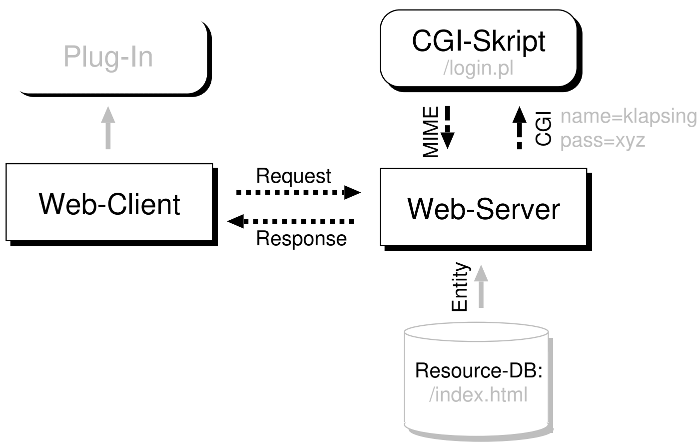

Abbildung 2.8: Web-Server kooperiert mit CGI-Script

CGI-Scripte können zur Integration von weiteren Informationssystemen ge-

[8]Eine Resource-Speicher kann z.B. ein Datei-System sein. Eine Web-Resource wird in diesem Fall in einer Datei abgespeichert.

nutzt werden. Beispielsweise können Unternehmensdatenbanken in eine Web-Anwendung integriert werden. Es gibt eine Reihe von Technologien zur Entwicklung von (interaktiven) Web-Anwendungen, welche die CGI-Schnittstelle benutzten oder in Web-Servern integriert sind (siehe z.B.: [5, 123, 1, 4, 2, 3, 173, 127]).

Eine Web-Resource kann Programm-Code (z.B. [60, 172]) enthalten, der in Abhängigkeit eines Ereignisses (engl. event) durch den Web-Client (oder durch ein im Web-Client integriertes Plug-In [57]) ausgeführt wird. Das *Intrinsic Event Model*, welches Bestandteil von HTML [144] ist, definiert, welche Ereignisse ausgelöst werden können. Beispiele für Ereignisse sind: Mausereignisse, HTML-Form-Ereignisse (z.B. bei fehlerhafter Benutzer-Eingabe), Laden und Entladen eines HTML-Dokumentes durch den Client. Das *Document Object Modell* (DOM) erweitert das Intrinsic Event Model. Die DOM-Spezifikationen [187] definieren eine Anwendungsschnittstelle (engl. Application Programming Interface (API)), die unabhängig von einer Programmiersprache ist. Die Erstellung, Modifikation und Abfrage von XML- und HTML-Dokumenten ist über diese Schnittstelle möglich.

2.3 Datenstrukturierung

Das Format der Nutzdaten einer Request- oder Response-Message wird durch den MIME-Type definiert. Es gibt eine Reihe vordefinierter Datentypen, die durch die MIME-Standards (RFC2045-2049) festgelegt werden. Durch einen Erweiterungsmechanismus ist jedoch auch die Verwendung benutzerdefinierter Datentypen möglich, so daß prinzipiell beliebige Datenformate genutzt werden können. In Web-basierten Informationssystemen haben sich Datenformate zur Datenstrukturierung durchgesetzt, die Anwendungen der Standard Generalized Markup Language (SGML) [90] bzw. der Extensible Markup Language [25] sind.

Standard Generalized Markup Language

Standard Generalized Markup Language (SGML) [90] ist ein Standard der *International Organization for Standardization* (ISO) zur Repräsentation von elektronischen Texten mittels geräte- und systemunabhängiger Methoden. Der Standard definiert eine Metasprache zur formalen Beschreibung einer Auszeichnungssprache (engl. markup language). Ursprünglich wurden Textauszeichnungen (engl. markup) innerhalb von Texten genutzt, um Setzern Anweisungen zu geben, in welcher Form ein bestimmter Text gedruckt werden soll. Eine Auszeichnungssprache spezifiziert, welche Auszeichnungen innerhalb von Texten erlaubt sind, welche angegeben werden müssen und wie Auszeichnungen von Text unterschieden werden.

Deskriptive Auszeichnung: SGML ist eine deskriptive Auszeichnungssprache. Das bedeutet, daß Textauszeichnungen, die der Strukturierung bzw. Kategorisierung von Teilen eines Dokumentes dienen, von prozeduralen Anweisungen, welche definieren, wie ein Dokument zu verarbeiten ist, getrennt. Beispielsweise kann eine Textauszeichnung wie `<H1>...</H1>` eine Überschrift innerhalb eines Textes markieren, aber es wird dadurch nicht festgelegt, wie eine Überschrift zu verarbeiten ist. Die Trennung deskriptiver Textauszeichnungen von prozeduralen Anweisungen ermöglicht die Verarbeitung des gleichen Dokuments durch unterschiedliche Software und damit potentiell zu verschiedenen Zwecken. Ein Dokument, das eine Reihe von Überschriftenmakierungen enthält, kann z.B. (a) in eine dem Dokument entsprechende Postscript-Datei transformiert werden, wobei das Inhaltsverzeichnis automatisch generiert wird und die Überschriften in einem sich vom übrigen Text abhebenden Font dargestellt werden, (b) in eine dem Dokument entsprechende Web-Resource transformiert werden, wobei das automatisch generierte Inhaltsverzeichnis aus Hyperlinks besteht, welche den Anfang einzelner Abschnitte referenzieren[9].

Interoperabilität von Zeichenkodierungen: Eine grundlegendes Entwurfsziel von SGML war es, zu ermöglichen, daß Dokumente interoperabel durch verschiedene Hard- und Software verarbeitet werden können, ohne daß Information verloren geht (Systemunabhängigkeit). Die deskriptive Auszeichnung von Texten, für die eine Grammatik angegeben wird, reicht nicht, um diese Anforderung zu erfüllen. Eine weitere Voraussetzung ist, daß unterschiedliche Zeichenkodierungen (engl. character sets) ineinander überführt werden können. Dazu unterstützt SGML einen Mechanismus zur Zeichenersetzung (engl. string substitution), der systemunabhängig ist. Die Zeichenketten (engl. strings), die für diesen Zweck definiert werden können, nennen sich Entities. Entities werden bei der Verarbeitung eines Dokumentes in korrespondierende Zeichenketten expandiert bzw. transformiert.

Document Type Definition SGML definiert eine Grammatik, die zur Erstellung von *Document Type Definitions* (DTD) dient. Eine DTD definiert formal, welche Bestandteile ein Dokument eines bestimmten Typs haben muß/darf und wie diese zu strukturieren sind. Ein Dokument, für das ein bestimmter Typ (DTD) angegeben wird, kann auf Übereinstimmung mit der DTD geprüft (va-

[9] Für ein Beispiel einer praktischen Anwendung von SGML siehe das DocBook-Projekt [188].

lidiert) werden. Zum Einlesen und zur Validierung von Dokumenten dienen Syntaxanalysierer (engl. parser) und Gültigkeitsprüfer (engl. validator). Verschiedene Dokumente des gleichen Typs können (bei erfolgreicher Validierung) auf einheitliche Weise verarbeitet werden. Abbildung 2.9 zeigt ein Beispiel einer DTD und Abbildung 2.10 ein Dokument, das dieser DTD entspricht. Ein Dokument, das einer DTD entspricht, wird auch Instanz (dieser DTD) genannt.

```
<!ELEMENT html O O (head, body)>
<!ATTLIST html lang CDATA #IMPLIED>
<!ELEMENT HEAD O O (%head.content;) +(%head.misc;)>
<!ELEMENT BODY O O (%block;|SCRIPT)+ +(INS|DEL)>
```

Abbildung 2.9: Beispiel einer DTD

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
 <head>
  ...
 </head>
 <body>
  ...
 </body>
</html>
```

Abbildung 2.10: Beispiel eines Dokuments vom Typ html

SGML bezeichnet eine strukturierte, textuelle Einheit als Element. Struktur bedeutet in diesem Zusammenhang, daß Elemente wiederum Elemente enthalten können. Der Autor einer DTD definiert durch ein Inhaltsmodell (engl. content model) welche Strukturen erlaubt sind. Zur Definition von Strukturen dienen Symbole für obligatorische Bestandteile, Sequenzen (,) und Alternativen (|). Für diese Bestandteile können Kardinalitäten (`*,+,?`) angeben werden . Im Beispiel 2.9 wird das Element `html` definiert, das die Elemente `head` und `body` in der angegebenen Reihenfolge enthalten muß. Der Name eines Elementes wird durch eine Markierung (engl. tag) angegeben. Beispielsweise sind `<html>` und `</html>` Start- (engl. begin tag) und Ende-Tag (end tag) eines Elements vom Typ `html`.[10] SGML stellt weitere Notation zur Definition von z.B. Attributen und Entities zur Verfügung.

[10]Elemente verschiedenen Typs haben verschiedene Namen. Oft werden sprechende Element-Namen vergeben, so daß der Eindruck entstehen könnte, dies würde schon die Semantik des Elements festgelegt. Die Definition von Semantik, über strukturelle Beschreibungen hinaus, ist durch SGML nicht möglich. Der Name eines Elements wird durch die Tags zu einem generischen Identifikator (engl. generic identifier). Erst durch die Verarbeitung (Interpretation) der Elemente durch eine Anwendung erhalten diese ihre Bedeutung. Verschiedene Anwendungen können jedoch das gleiche Dokument unterschiedlich interpretieren.

HyperText Markup Language

Die HyperText Markup Language (HTML - aktuelle Version 4.01 [144]) ist derzeit die am weitesten verbreitete Auszeichnungssprache im Web. HTML ist eine SGML-Anwendung, d.h. es existiert eine DTD, die eine Grammatik für HTML definiert. Neben Auszeichnungselementen zur Beschreibung der Strukturierung, der Referenzierung von Web-Resourcen, der Interaktivität und der Einbettung von Web-Resourcen existieren Auszeichnungselemente zur Definition des Layout. Die Vermengung von Elementen mit anderer Zielsetzung als der Strukturierung von Dokumenten, speziell die Vermengung von Strukturierung- und Layout-Elementen, ist durch die evolutionäre Entwicklung von HTML bedingt. Durch die Einführung der Cascading Style Sheets (CSS) [23, 104] wird eine strikte Trennung von Strukturierung und Layout ermöglicht. Durch die Einführung von XML und XML-Namespaces (siehe folgende Abschnitte) können Elemente, die verschiedenen Zielsetzungen dienen, separiert werden. XHTML [82, 83] basiert auf einer Reihe von XML-DTDs, die es ermöglichen, HTML-Dokumente XML-konform zu verfassen. XHTML ist durch weitere XML-DTDs erweiterbar. Die Verwendung von XHTML ist vorteilhaft, da neben HTML-Werkzeugen auch XML-fähige Werkzeuge entsprechende XHTML-Dokumente verarbeiten können. Das gilt auch für die Wireless Markup Language (WML- aktuelle Version 2.0 [190]), die speziell für Dokumente entwickelt wurde, welche auf mobile Endgeräte übertragen werden. Zukünftig soll XHTML so weiterentwickelt werden, daß auch XHTML-Dokumente auf mobilen Endgeräten angezeigt werden können.

Extensible Markup Language

Die Extensible Markup Language (XML) [25] ist eine modifizierte Version von SGML, um diese Technologie in Web-basierten Informationssystemen anwenden zu können. Ziel bei der Entwicklung von XML war es, Kompatibilität zu SGML und HTML zu gewährleisten und die Entwicklung von Implementationen zu erleichtern. Eine Gegenüberstellung von SGML und XML wird in [33] gegeben.

XML-Schema [20, 61, 178] ist sowohl eine Weiterentwicklung, als auch eine Anwendung von XML. Wesentliches Merkmal ist, daß eine XML-Schema-DTD selbst in XML-Syntax formuliert wird. Eine XML-DTD definiert die erlaubte Struktur einer XML-Schema-DTD (weshalb XML-Schema eine Anwendung von XML ist). Eine XML-Schema-DTD definiert die erlaubte Struktur eines anwendungsspezifischen Dokumenten-Typs. Außerdem können durch eine XML-Schema-DTD Constraints für den Inhalt von Elementen definiert werden, so daß der Inhalt einem bestimmten Datentyp entsprechen muß.

Das Layout von XML-Dokumenten kann durch die Extensible Stylesheet Language (XSL) [184] gesteuert werden, die aus folgenden Teilen besteht: (a) XSL Transformations (XSLT) [34], (b) XML Path Language (XPath) [35] und (c) XSL Formatting Objects (XSL-FO) [75]. XSLT dient der Transformation von XML-Dokumenten in eine Zielsyntax, die wieder XML sein kann, jedoch nicht sein muß. Der Autor eines Stylesheet definiert die Zielsyntax. XPath ist eine (Abfrage-)Sprache, die von XSLT genutzt wird, um einzelne Teile eines XML-Dokumentes zu adressieren bzw. darauf zuzugreifen. XSL Formatting Objects (XSL-FO) definieren, in welchem Layout XML-Dokumente angezeigt werden sollen. In Stylsheets werden XSLT- und XSL-FO-Elemente verschachtelt. Eine eindeutige Identifikation von Transformations-Syntax und Layout-Syntax ist durch XML-Namespaces gewährleistet.

XML-Namespaces

Ein XML-Namespace [24] ist ein Namensraum für Namen von XML-Elementen und XML-Attributen, der durch eine URI-Referenz [17] identifiziert wird. In einem XML-Dokument verwendete XML-Elemente und XML-Attribute können Namensräumen zugeordnet werden. Ein XML-Dokument kann Markup enthalten, das verschiedenen Namensräumen zugeordnet wird. URI-Referenzen, die Namensräume identifizieren, sind identisch, wenn die Namensraum-Namen zeichenweise äquivalent sind. Die XML-Namespace-Spezifikation definiert, daß auf die durch den Namensraum-Name referenzierte Web-Resource nicht (z.B. zu Validierungszwecken) zugegriffen werden muß.[11] Abbildung 2.11 zeigt ein Beispiel einer Namensraum-Deklaration.

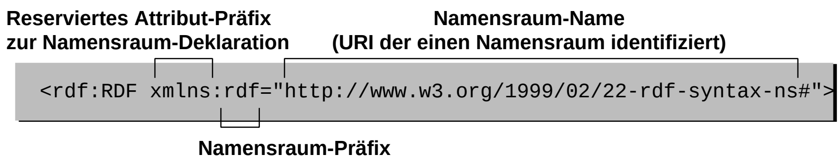

Abbildung 2.11: Beispiel einer Namensraum-Deklaration

Abbildung 2.12 zeigt die Verwendung von XML-Namensräumen. Im Beispiel wird im Element `html` ein Namensraum mit Präfix `astro` deklariert. Das Präfix `astro`, das Trennzeichen `:` und der lokale Name (engl. LocalPart)[12] `position`

[11] Anwendungsabhängig kann jedoch die Verarbeitung referenzierter Schemata bzw. DTDs notwendig werden.

[12] Der LocalPart ist der Name eines XML-Elements.

bilden zusammengenommen einen qualifizierten Namen (engl. qualified name). Ein qualifizierter Name ist bei Ersetzung des Präfix durch den Namensraum-Name eindeutig[13]. Weiter wird im Beispiel ein Default-Namespace deklariert. Dieser gilt für alle Elemente und Attribute, die nicht qualifizierte Namen sind.

Default-Namespace

```
<html
  xmlns="http://www.w3.org/TR/2000/REC-xhtml1-20000126#"
  xmlns:astro="http://www.astronautics.gal/schema#">

 <body>
  The position of the <astro:planet>earth</astro:planet>
  is the <astro:position>sun system</astro:position>
 </body>
</html>
```

Qualifizierter Name

Abbildung 2.12: Benutzung von XML-Namensräumen

2.4 Adressierung von Web-Resourcen

Der RFC2396 [17][14] definiert den Uniform Resource Identifier (URI). Ein URI ist eine Zeichenkette, mit der eine abstrakte oder physikalische Resource identifiziert bzw. referenziert wird. Eine Resource ist folgendermaßen definiert:

RFC2396-Definition: A resource[15] can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.

Der RFC2396 definiert eine Grammatik, deren Strukturvorgaben alle URIs entsprechen müssen. Die Grammatik definiert den generischen Aufbau von URIs (s. Abbildung 2.13).

```
<scheme>://<authority><path>?<query>
```

Abbildung 2.13: Generischer Aufbau von URIs

[13] Diese Aussage gilt nur eingeschränkt, wie in den folgenden Abschnitten noch gezeigt wird.
[14] Der RFC2396 ist ein Standard der IETF.
[15] Unter einer Web-Resource wird in dieser Arbeit eine durch einen URI referenzierte Resource verstanden, auf die mittels Internet-Protokollen zugegriffen werden kann.

- Der `scheme`-Teil eines URIs ist ein vordefinierter Bezeichner, der definiert, gemäß welcher weiteren Spezifikation ein URI zu interpretieren ist. Beispiele für vordefinierte Bezeichner mit dazugehörigen Spezifikationen sind z.B. URN [124], HTTP [64], FTP[142].

- Der `authority`-Teil ist ein Name, der durch eine Namensvergabestelle (z.B. der Internet Corporation for Assigned Names and Numbers [177]) verwaltet wird. Abhängig vom scheme-Teil wird diese Name interpretiert. Wird z.B. http als scheme-Bezeichner genutzt, referenziert der `authority`-Teil einen Informationsraum eines Web-Anbieters.

- Der `path`-Teil dient zur Referenzierung von Resourcen in einem Informationsraum. Es handelt sich um eine Zeichenkette, die durch das Zeichen / unterteilt werden kann, um eine Hierarchie von Resourcen referenzieren zu können.

- Der `query`-Teil ist eine Zeichenkette, die durch die referenzierte Resource interpretiert wird. Resourcen können verarbeitende Prozesse sein, die z.B. zur dynamischen Abfrage von Informationen genutzt werden.

Ein URI kann ein Uniform Resource Locator (URL) [17, 18, 63] oder ein Uniform Resource Name (URN) [124] sein (siehe auch [181]). Der RFC 2396 definiert URL und URN wie folgt:

RFC2396-Definition: The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.[16]

Zur Referenzierung einzelner Teile einer Web-Resource kann ein Fragment-Bezeichner (engl. fragment identifier - siehe [17]) genutzt werden. Ein Fragment-Bezeichner wird durch ein Doppelkreuz (engl. crosshatch) "`#`" vom URI getrennt. Die Bedeutung eines Fragment-Bezeichners ist abhängig vom MIME-Type [66] der referenzierten Resource. In Abhängigkeit des MIME-Types einer Web-Resource wird festgelegt, welcher Teil der Web-Resource durch den Fragment-Bezeichner referenziert wird.

[16] Da die Begriffe URI, URL und URN unklar definiert sind, wird der Zusammenhang der Begriffe vom W3C und der IETF in [179] genauer erläutert.

Die Verwendung von Links in XML-Dokumenten und die Referenzierung von Teilen eines XML-Dokumentes spezifiziert das W3C durch die XML Linking Language (XLink) [55], die XML Path Language (XPath) [35] und die XML Pointer Language (XPointer) [56].

Die XLink-Spezifikation definiert XML-Syntax, mit der Links in XML-Dokumente eingefügt und beschrieben werden können. Unidirektionale Hyperlinks, wie sie in HTML benutzt werden können, sind möglich. Darüberhinaus ist die Definition von Links mit erweiterter Funktionalität möglich. Die erweiterte Funktionalität umfaßt beispielweise die Definition von Links, die Beziehungen zwischen mehr als zwei Resourcen beschreiben, die Beschreibung von Links durch Metadaten und die Trennung von Links und verlinkten Resourcen. Die XLink-Requirements-Spezifikation [54] legt fest, welche Anforderungen XLink erfüllen soll. Im Design von XLink sind Erfahrungen berücksichtigt, die mit Hypermedia-Technologien wie z.B. HyTime [89], TEI [162] und Dexter [73] gemacht wurden.

Die XPath- und XPointer-Spezifikationen definieren Syntax und Semantik, um Bestandteile von XML-Dokumenten zu referenzieren, d.h. es handelt sich um Fragment-Bezeichner für XML-Dokumente. XPath ermöglicht zudem die Überprüfung einer Übereinstimmung von XML-Syntax mit einem Muster. Diese Funktionalität wird im Zusammenhang mit XSLT genutzt.

Stickler schlägt in [165][17] URIs vor, die vor allem für die Nutzung in RDF-Applikationen entworfen wurden. Die Spezifikation definiert den Uniform Resource Primitive (URP), dessen vorrangige Eigenschaft es ist, nicht der Dereferenzierung (also dem Zugriff auf Web-Resourcen) zu dienen. Die zu übertragenden Nutzdaten sind bereits Bestandteil eines URPs. Hauptzweck eines URPs ist die Annotation bzw. Klassifikation von Web-Resourcen. Ein URP kann in einen Uniform Resource Term (URT) und Uniform Resource Value (URV) unterschieden werden. Ein URT ist ein URI, der zur Repräsentation von Vokabularen bzw. Taxonomien dient. Ein URI mit dem Scheme voc [170] ist ein Beispiel für einen URT. Ein URV ist ein URI, der einer lexikalisch definierten Datenstruktur entsprechen muß, mit der Daten in Form eines URIs übertragen werden. Beispiele für URV-Schemes sind data [115], auth [166], qname [167], tdl [168], uri [169], und xmlns [171].

[17] Diese Drafts wurden bei der IETF zur Überprüfung eingereicht.

3 Resource Description Framework

Dieses Kapitel stellt das Resource Description Framework (RDF) vor, welches durch das XWMF, ein im Rahmen dieser Arbeit entwickelter (siehe Kapitel 6) Ansatz, zur Modellierung und Beschreibung von Web-basierten Informationssystemen genutzt wird.

RDF wird durch die Spezifikationen *Resource Description Framework Model and Syntax Specification*[1] [101] (RDF-M+S) und *Resource Description Framework (RDF) Schema Specification*[2] [27] (RDFS) definiert. RDF kann für den Web-basierten und interoperablen Austausch von Metadaten, die Resourcen[3] beschreiben, genutzt werden. Ziel von RDF ist es zu erreichen, daß die Semantik, die durch Metadaten repräsentiert wird, von Automaten interpretiert werden kann.

Die RDF-Spezifikationen[4] definieren ein Datenmodell und eine Syntax, die XML und XML-Namensräume als Grundlage zur Repräsentation von Semantik nutzt. Im Gegensatz zur RDF-Syntax dient XML ausschließlich der Strukturierung von Daten (wenn nicht zusätzlich XML-Namensräume verwendet werden). Die Be-

[1] Die RDF-M+S-Spezifikation hat den Status einer W3C-Recommendation. Eine Recommendation ist die höchste Standardisierungsstufe, die das W3C vergibt. Der Standardisierungsprozeß (W3C-Recommendation-Track) sieht folgende Standardisierungsstufen vor: Working Draft, Last Call Working Draft, Candidate Recommendation, Proposed Recommendation, Recommendation.

[2] Die RDFS-Spezifikation hat den Status einer W3C-Candidate-Recommendation.

[3] Zur Schreibweise und Verwendung der Begriffe Resource und Property: Im Rahmen dieser Arbeit wird nicht die deutsche Schreibweise Re*ss*ource sondern Re*s*ource verwendet, um deutlich zu machen das durch RDF beschreibbare Resourcen gemeint sind. Weiter wird statt des Begriffs Eigenschaft der englische Begriff Property verwendet, um auch diesbezüglich auf die RDF-Spezifikationen abzuheben.

[4] Das sind die beiden oben genannten und weitere, ergänzende Dokumente (Vocabulary Description Language [28], Model Theory [76, 77], Test Cases [71], Syntax Specification (Revised) [7], RDF: Concepts and Abstract Data Model [97], RDF Primer [113], Requirements for a Web Ontology Language [79]), die sich zur Zeit in der Entwicklung befinden.

deutung der Daten für einen maschinellen *generischen Interpreter*[5] wird durch XML jedoch nicht festgelegt. Dazu folgendes Beispiel:

```
<Buch href="http://www.galaxy.gal/HitchhikersGuide">
  <Titel>The Hitchhiker's Guide to the Galaxy</Titel>
  <Autor>Douglas Adams</Autor>
</Buch>
```

Ein Mensch kann aus dem XML-Fragment herauslesen, daß "Douglas Adams" der Autor des Buchs mit dem Titel "The Hitchhiker's Guide to the Galaxy" ist.

Ein generischer Interpreter kann folgendes XML-Fragment von dem obigen *semantisch* (in seiner Bedeutung) nicht unterscheiden (ein einfacher generischer Interpreter kann allenfalls feststellen, daß verschiedene Token für die strukturgebenden Elemente verwendet worden sind[6]).

```
<xyz abc="http://www.galaxy.gal/HitchhikersGuide">
  <ghi>The Hitchhiker's Guide to the Galaxy</ghi>
  <jkl>Douglas Adams</jkl>
</xyz>
```

Ziel von RDF ist es, Daten derart durch RDF-Syntax zu repräsentieren, daß diese von generischen Interpretern auch semantisch verarbeitet werden können.

3.1 Begriffseinführungen und Beispiel

Ein RDF-Beschreibungsmodell kann als Graph dargestellt werden. Ein RDF-Graph besteht aus Knoten und gerichteten markierten Kanten. Ein Knoten repräsentiert entweder eine Resource oder ein Literal (ein String oder ein XML-Datentyp). Jede Entität, die eindeutig durch einen URI identifiziert werden kann, ist eine Resource[7] (wie z.B. eine HTML-Seite bzw. ein Fragment davon, ein Bild, eine Audio- oder Video-Datei). Die Markierung (engl. label) einer gerichteten Kante repräsentiert eine Eigenschaft/ein Prädikat (engl. predicate) einer Resource. Abbildung 3.1 zeigt ein Statement (bestehend

[5] Unter einem generischen Interpreter sei hier ein System verstanden, das über die zu interpretierenden Daten hinaus über kein spezielles Domänenwissen verfügt.

[6] Fortgeschrittenere Systeme, z.B. Systeme zur Verarbeitung natürlicher Sprache, könnten die Token bestimmten Konzepten zuordnen und so auf die Semantik schließen. Derartige Systeme gehen über die Zielsetzung von RDF und von dieser Arbeit hinaus und werden deshalb hier nicht betrachtet.

[7] Eine Resource kann auch eine Entität sein, auf die nicht via Web zugegriffen werden kann: z.B. ein gedrucktes Buch oder eine Person.

aus `subject`, `predicate`, `object`), welches aussagt, daß `Douglas Adams` der Autor (`creator`) der Resource `http://www.galaxy.gal/HitchhikersGuide` ist. Vokabular, das zur Beschreibungen von Web-Resourcen genutzt werden kann, wird durch RDF-Schemata definiert. Im Beispiel referenziert der URL `http://www.schema.org/lit#` das RDF-Schema, in dem die Bedeutung des Properties `creator` definiert ist. Ein Property kann genutzt werden, um Aussagen (engl. statement) über Eigenschaften von Web-Resourcen oder über Beziehungen (engl. relations) zwischen Web-Resourcen auszudrücken. Ein Property wird Prädikat (engl. predicate) genannt, wenn es in einem Statement verwendet wird.

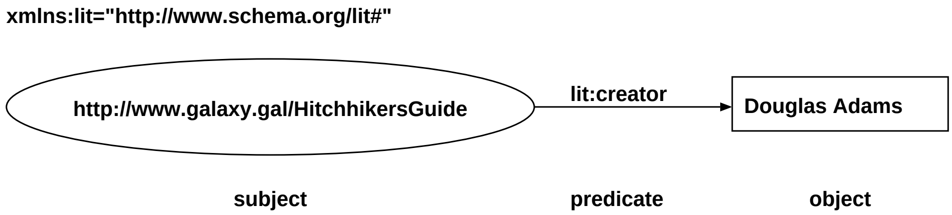

Abbildung 3.1: Beispiel eines RDF-Statements

Die RDF-M+S-Spezifikation definiert eine XML-Syntax, in der Beschreibungsmodelle notiert werden können. Das durch den in Abbildung 3.1 gezeigten Graph dargestellte Beschreibungsmodell kann durch die in in Abbildung 3.2 gezeigte Syntax repräsentiert werden. Zeile 1 deklariert das Dokument als XML-Dokument. In Zeile 3 wird das Namensraum-Präfix `rdf` deklariert, welches das RDF-Schema referenziert, in dem das RDF-M+S-Vokabular definiert ist. Damit ist für Zeile 2 festgelegt, daß das XML-Element `RDF` aus diesem Namensraum gemeint ist. Zeile 4 deklariert das Namensraum-Präfix `lit`, womit ein weiteres Vokabular referenziert wird. RDF ermöglicht es, Properties aus *verschiedenen* Vokabularen innerhalb *eines* Beschreibungsmodells zu verwenden. Das Element `rdf:Description` in Zeile 6 legt mit dem Attribut `rdf:about` fest, über welche Web-Resource (über welches subject) eine Aussage gemacht werden soll (also über das Buch HitchhikersGuide, das durch URL `http:// www.galaxy.gal /HitchhikersGuide` referenziert wird). Das XML-Element `lit:creator` in Zeile 7 mit dem Inhalt (engl. content) `"Douglas Adams"` repräsentiert das Prädikat und das Objekt der Aussage.

Im Beispiel referenzieren die Namensraum-Präfixe `rdf` und `lit` RDF-Schemata, die Vokabular zur Beschreibung von Web-Resourcen zur Verfügung stellen. Zur Erstellung eines RDF-Schemas dient ein durch die RDFS-Spezifikation [27] definiertes Meta-Vokabular (im folgenden RDFS-Vokabular genannt). Ein RDF-Schema legt z.B. fest, welchen Typen von Web-Resourcen ein RDF-Property

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#"
  xmlns:lit="http://www.schema.org/ex#">

  <rdf:Description rdf:about="http://www.galaxy.gal/HitchhikersGuide">
      <lit:creator>Douglas Adams</lit:creator>
  </rdf:Description>

</rdf:RDF>
```

Abbildung 3.2: RDF-Statement linearisiert in XML-Syntax

zugeordnet werden darf und welche Werte für dieses Property erlaubt sind.[8] Damit kann, neben der Zuweisung von Attributen, definiert werden, welche Beziehungen zwischen bestimmten Typen von Web-Resourcen erlaubt sind.[9]

Ein RDF-Schema kann selbst mittels eines RDF-Beschreibungsmodells ausgedrückt werden. Abbildung 3.3 zeigt ein RDF-Schema, welches für das Property `creator` festlegt, daß dieses Web-Resourcen von Typ `Book` zugeordnet werden und Werte vom Typ `Literal` annehmen darf. Eine graphische Repräsentation einer Instanz dieses Schemas wurde bereits in Abbildung 3.1 gezeigt.[10]

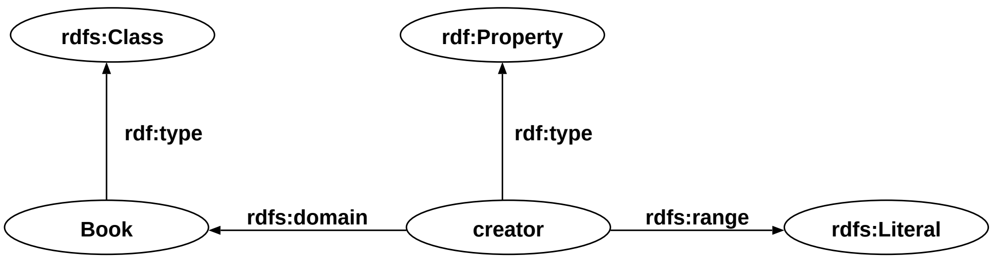

Abbildung 3.3: Graphische Repräsentation eines RDF-Schemas

Abbildung 3.4 zeigt die Zusammenhänge im Überblick. Auf der untersten Ebene wird die RDF-Beschreibung gezeigt, die Vokabular aus den Namensräumen

[8] Diese Aussage basiert auf einer bestimmten Interpretation der Semantik von RDF und RDFS. Es gibt jedoch auch andere Interpretationen. Auf alternative Interpretationen wird im Kapitel 4 näher eingegangen.

[9] Die Autoren der RDFS-Spezifikation vergleichen das Typ-System von RDF mit Typ-Systemen objekt-orientierter Programmiersprachen und finden Ähnlichkeiten (explizit wird die Programmiersprache Java genannt). Das Typ-System unterscheidet sich jedoch durch einige Besonderheiten von der traditionellen Sichtweise bei der eine Klasse definiert, welche Attribute eine Instanz der Klasse haben darf. Ein RDF-Schema definiert dagegen, welchen Klassen ein Property zugeordnet werden darf. Die Klasse kennt nicht ihre Attribute, sondern die Attribute kennen ihre Klasse. In der RDF Schema Spezifikation wird das als property-zentrierter Ansatz bezeichnet.

[10] Unter der Annahme, daß bereits an anderer Stelle die Resource http://www.galaxy.gal/HitchhikersGuide als Instanz vom Type Book deklariert wurde.

`rdf` und `lit` verwendet. Das Schema `lit` definiert Vokabular zu Beschreibung von Literatur. Zur Erstellung des Schemas `lit` wird RDF-M+S- und RDFS-Vokabular benutzt.

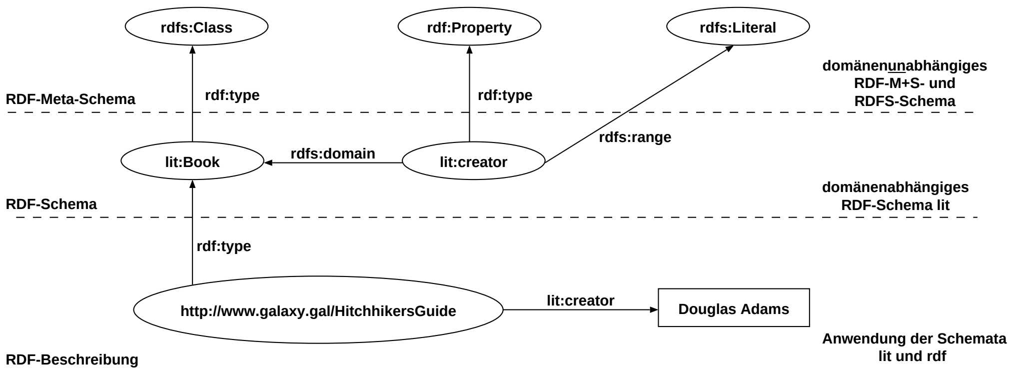

Abbildung 3.4: Zusammenhang von RDF-Beschreibungen und RDF-Schemata

Für weitere einführende Darstellungen siehe [113].

3.2 RDF-Datenmodell

Die RDF-M+S-Spezifikation definiert das RDF-Datenmodell, welches allen RDF-Beschreibungsmodellen zugrunde liegt.[11] Mit mengentheoretischen Termini wird ein abstraktes Datenmodell spezifiziert, das zur Erstellung von RDF-Ausdrücken genutzt werden kann. Im folgenden Abschnitt wird eine formale Darstellung der Mengenzusammenhänge gezeigt.

Mengentheoretische Betrachtungen

Die im folgenden gezeigten mengentheoretischen Definitionen[12] sind eine Interpretation der elf Definitionen des RDF-Datenmodells – die dem Abschnitt 5

[11]Die RDF-M+S-Spezifikation unterscheidet terminologisch nicht zwischen der Definition des Datenmodells (in dieser Arbeit RDF-Datenmodell genannt) und der Anwendung des RDF-Datenmodells (in dieser Arbeit RDF-Beschreibungsmodell bzw. kurz RDF-Beschreibung genannt).

[12]Sergey Melnik zeigt in [119] eine Interpretation des RDF-Datenmodells als algebraische Spezifikation. Die in diesem Abschnitt gezeigte Formalisierung ist weniger umfassend, da bewußt nur die Aussagen erfaßt werden, die in den elf Definitionen des RDF-Datenmodells gemacht werden. Eine umfassendere Formalisierung (in Prädikatenlogik erster Ordnung), die zusätzlich die durch die RDF-Schema-Spezifikation definierte Semantik einbezieht, wird in Kapitel 4 angegeben.

(Formal Model for RDF) der RDF-Spezifikation [101] entnommen werden können (siehe auch Anhang 9.1). Die Numerierung der im folgenden gezeigten Definitionen entspricht der Numerierung der elf Definitionen des RDF-Datenmodells, so daß eine eindeutige Zuordnung möglich ist. Jede Definition wird durch einen Kommentar ergänzt, der die intendierte Semantik weiter verdeutlicht bzw. diskutiert. Zur besseren Übersicht zeigt Abbildung 3.5 die mengentheoretischen Zusammenhänge graphisch.

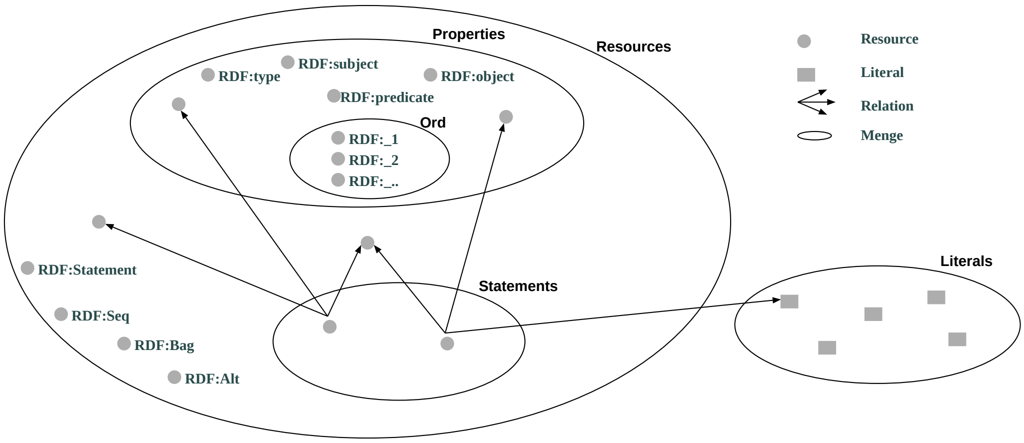

Abbildung 3.5: RDF-Konzepte

1. $Resources = \{x | x$ ist URI gemäß RFC2396$\}$

   Kommentar: Durch diese Definition wird festgelegt, daß die Menge *Resources* existiert. Ein Element der Menge *Resources* wird *Resource* genannt. Die Definition einer Resource ist sehr vage (vgl. [101]): *...All things being described by RDF expressions are called resources. ...* Weiter heißt es: *...Resources are always named by URIs plus optional anchor ids ... Anything can have a URI; the extensibility of URIs allows the introduction of identifiers for any entity imaginable.* Das bedeutet, eine Entität die durch einen RDF-Ausdruck beschrieben wird, ist eine Resource. Einschränkend könnte hier nur die Forderung wirken, daß jede Resource einen Namen haben muß, der einem URI entspricht. Diese Einschränkung wird jedoch relativiert, durch die Aussage, daß eine Erweiterungsmöglichkeit für URIs besteht, so daß URIs für alle Dinge definiert werden können. Die Forderung das jede Resource einen Namen haben muß ist allerding kritisch, da es sinnvoll sein kann, anonyme Resourcen in Beschreibungsmodellen zuzulassen. Die RDF-M+S-Spezifikation selbst verwendet beispielsweise anonyme Resourcen zur Beschreibung von strukturierten Werten. Diesbezüglich ist die RDF-M+S-Spezifikation widersprüchlich.

2. $Literals = \{x | x \text{ ist String gemäß der XML-Spezifikation}\}$

   Kommentar: Durch diese Definition wird festgelegt, daß die Menge *Literals* existiert. Ein Element der Menge *Literals* wird *Literal* genannt. Ein Literal ist eine Zeichenkette.

3. $Properties \subset Resources$

   Kommentar: Jedes Property ist auch eine Resource.

4. $Statements \subseteq Properties \times Resources \times (Resources \cup Literals)$

   Kommentar: Ein Statement ist ein geordnetes 3-tupel (engl. triple). An der ersten Stelle steht ein Property-Name, an der zweiten Stelle ein Resourcen-Name und an der dritten Stelle entweder ein Resourcen-Name oder ein Literal. Die Menge aller möglichen Statements ergibt sich aus dem kartesischen Produkt der Mengen Properties, Resources und Literals. Ein Beschreibungsmodell ist Untermenge der Menge aller möglichen Statements. Die RDF-Spezifikationen definieren nicht eindeutig, ob ein Modell auf die durch ein RDF-Dokument repräsentierten Statements beschränkt ist, oder ob vom Dokument aus referenzierte RDF-Dokumente (bzw. dadurch repräsentierte Statements) auch Bestandteil eines Modells sind. An zweiter Stelle, die das Subject eines Statements angibt, dürfen nur Resourcen-Namen stehen. Diese Einschränkung führt dazu, daß über Literale keine Aussagen gemacht werden können.[13]

5. RDF:type $\in$ *Properties*

   Kommentar: RDF:type ist Element der Menge Properties. RDF:type kann genutzt werden, um eine Resource explizit einer Klasse (engl. class) zuzuordnen (siehe auch die Definition für TypRel weiter unten). Eine Klasse ist eine Resource vom RDF:type RDFS:Class und kann als Menge aufgefaßt werden. Das Token RDF: ist Repräsentant für den URI[14], der das RDF-M+S-Schema referenziert. Die RDF-M+S-Spezifikation ist bezüglich des Namensraums rdf: unklar bzw. widersprüchlich. Zum einen deshalb, weil ausgesagt wird, daß RDF-Syntax eine Anwendung von XML ist. Dann müßte jedoch statt der EBNF-Notation eine DTD zur Syntax-Definition angegeben werden. Zum anderen wird das Namensraum-Präfix rdf: durch die EBNF-Notation unveränderlich festgelegt. Dies wiederspricht der XML-Namenspace-Spezifikation, die definiert, daß der Autor

[13] Über ein Literal kann allenfalls eine Aussage im Zusammenhang mit einer Resource gemacht werden, indem ein entsprechendes Statement reifiziert wird, und so über das Statement eine Aussage gemacht wird.

[14] http://www.w3.org/1999/02/22-rdf-syntax-ns#

eines XML-Dokumentes ein Namensraum-Präfix festlegen kann.[15] Zur eindeutigen Referenzierung eines Namensraums ist lediglich der entsprechende URI unveränderlich. Der URI des RDF-M+S-Schemas kommt jedoch in der EBNF-Notation überhaupt nicht vor. Außerdem unterschiedet XML Groß- und Kleinschreibung, weshalb RDF: und rdf: generell zu unterscheiden sind. Sollen Modell und Syntax diesbezüglich übereinstimmen, sind entsprechende Anpassungen notwendig.

6. $TypeRel = \{(\text{RDF:type}, sub, obj) \in Statements | sub \in Resources \wedge obj \in Resources\}$

   Kommentar: Eine Type-Relation kann genutzt werden, um eine Resource explizit einer Klasse zuzuordnen. Eine Type-Relation ist Element der Menge Statements. Das Subjekt und Objekt einer Type-Relation muß eine Resource sein. Unter der Annahme, daß die Mengen Resources und Literals disjunkt sind, bedeutet dies, daß Literale nicht in Type-Relationen vorkommen dürfen. Ob die Mengen Resources und Literals disjunkt sind, ist durch die RDF-M+S-Spezifikation nicht formal festgelegt. Die folgende der RDF-M+S-Spezifikation entnommene Aussage läßt diesen Schluß jedoch zu (wenn man das Wort *or* hier als exklusives oder versteht): *The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive datatype defined by XML.* Auch die weiter oben angeführte Definition 4 läßt diesen Schluß zu. Eine Zeichenkette kann im RDF-Datenmodell nicht explizit als Literal deklariert werden. Dies kann zu Zweideutigkeiten führen. Beispielsweise läßt sich eine Zeichenkette, die einem URI syntaktisch gleicht, nicht von einem URI unterscheiden, der eine Resource repräsentieren soll.

7. $\text{RDF:Statement} \in Resources$
   $\text{RDF:Statement} \notin Properties$

   Kommentar: RDF:Statement ist Element der Menge Resources, aber nicht Element der Menge Properties. Das Element RDF:Statement wird zur Reifikation eines Statements genutzt. Eine Reifikation wird (unter anderem) dazu genutzt, um Aussagen über Aussagen machen zu können. Zur Reifikation wird eine weitere Resource – im weiteren Reifikant genannt –

[15] An anderer Stelle weist die RDF-M+S-Spezifikation darauf hin, daß rdf: nur als Platzhalter für ein variables Namensraum-Präfix gesehen werden soll. In der Praxis hat die EBNF-Notation jedoch bereits dazu geführt, daß das Namensraum-Präfix rdf: von bestimmten RDF-Werkzeugen als unveränderliches Namensraum-Präfix behandelt wird.

in das Beschreibungsmodell eingefügt, die ein Statement repräsentiert. Der Reifikant ist als Typ RDF:Statement zu deklarieren.

8. RDF:predicate $\in$ *Properties*
RDF:subject $\in$ *Properties*
RDF:object $\in$ *Properties*

   Kommentar: RDF:predicate, RDF:subject und RDF:object sind vordefinierte Elemente der Menge Properties. Diese Elemente werden, wie auch RDF:Statement, zur Reifizierung eines Statements genutzt. RDF:predicate dient zur Kennzeichnung des Prädikats eines Statements; analog RDF:subject zur Kennzeichnung des Subjekts und RDF:object zur Kennzeichnung des Objekts.

9. $$\begin{aligned} ReifiedStatement = \{r \in Resources| \\ &(\text{RDF:predicate}, r, pred) \in Statements \wedge \\ &(\text{RDF:subject}, r, subj) \in Statements \wedge \\ &(\text{RDF:object}, r, obj) \in Statements \wedge \\ &(\text{RDF:type}, r, \text{Statement}) \in Statements\} \end{aligned}$$

   Kommentar: Zur Reifizierung eines Statements dient ein Reifikant r, der in vier Statements an Subjekt-Position vorkommt. Drei Statements dienen der Kennzeichnung der einzelnen Teile eines reifizierten Statements und ein Statement deklariert den Reifikant als Statement. Es geht aus der RDF-M+S-Spezifikation nicht klar hervor, ob die Menge Statements (d.h. die Menge aller Triple einer Beschreibung) und die Menge (bzw. das Konzept) Statement (d.h. die Menge der Resourcen/Reifikanten, die ein Statement repräsentieren) unterschieden wird. Wird keine Unterscheidung intendiert, ist die RDF-M+S-Spezifikation bezüglich der Zuordnung des Reifikanten als Element der Menge Statements nicht formal korrekt. Die Zuordnung eines Reifikanten als Element der Menge Statements ist nicht möglich, da ein Reifikant kein geordnetes 3-tupel ist, wie es Definition 4 vorschreibt. Außerdem ist die RDF-M+S-Spezifikation bezüglich der Implikation von Reifikationen für ein Beschreibungsmodell unklar. Die RDF-M+S-Spezifikation definiert nicht, ob nur eine Reifikation eines Statements existieren darf oder auch mehrere. Weiter ist nicht definiert, wie ein Kennzeichner oder Name eines Reifikants syntaktisch auszusehen hat. Abschnitt 4.1. der RDF-M+S-Spezifikation zeigt ein Beispiel einer Reifikation, in dem der Reifikant eine anonyme Resource ist. Anonyme Resourcen sind im RDF-Datenmodell jedoch nicht vorgesehen (siehe auch Kommentar der Definition 1).

10. RDF:Seq $\in (Resources \setminus Properties)$
RDF:Bag $\in (Resources \setminus Properties)$
RDF:Alt $\in (Resources \setminus Properties)$

Kommentar: RDF:Seq, RDF:Bag und RDF:Alt sind Elemente der Menge Resources, aber nicht der Menge Properties. Diese Elemente können genutzt werden, um Repräsentationen für Sammlungen (engl. collection) von Resourcen oder Literalen zu erstellen. Eine Resource oder ein Literal darf in einer Sammlung mehrfach vorkommen. Eine Resource vom Typ RDF:Bag dient der Repräsentation einer ungeordneten Sammlung, eine Resource vom Typ RDF:Seq der Repräsentation einer geordneten Sammlung (engl. sequence) und eine Resource vom Typ RDF:Alt der Repräsentation einer Sammlung von zueinander alternativen Resourcen oder Literalen. Einer Sammlung r wird ein Objekt o durch ein Prädikat RDF:_x (x ist natürliche Zahl) aus der Menge Ord (siehe Definition 11) zugeordnet. Die RDF-M+S-Spezifikation ist in ihren Definitionen bezüglich der mengentheoretischen Zusammenhänge zur Repräsentation von Containern nicht vollständig. Außerhalb der Definitionen 10 und 11 finden sich in der RDF-M+S-Spezifikation jedoch weitere Hinweise. Zur Verdeutlichung der Zusammenhänge sei hier ergänzend Definition 10a angeführt:

$$
\begin{aligned}
collection = \{c \in Resources| \\
&\quad container \in \{\mathrm{RDF{:}Seq}, \mathrm{RDF{:}Bag}, \mathrm{RDF{:}Alt}\} \wedge \\
&\quad o \in Resources \cup Literals \wedge \\
&\quad i \in Ord \wedge \\
&\quad (\mathrm{RDF{:}type}, c, container) \in Statements \wedge \\
&\quad (i, c, o) \in Statements
\end{aligned}
$$

11. $Ord \subset Properties$
$Ord = \{\mathrm{RDF{:}\_i} \in Properties | i \in N\}$

Kommentar: Es existiert eine Menge Ord, die Untermenge der Menge Properties ist. Ein Element der Menge Ord – im weiteren Ordinal-Property genannt – hat den syntaktischen Aufbau RDF:_x, wobei x eine natürliche Zahl ist. Eine Zahl x kann als Ordinalzahl aufgefaßt werden, mit der eine Ordnung beispielsweise für eine Repräsentation vom Typ RDF:Seq aufgebaut werden kann. Die RDF-M+S-Spezifikation ist unklar bzgl. der Benutzung von Ordinal-Properties. Es wird zwar (außerhalb der normativen Definitionen) festgelegt, daß mit RDF:_1 das erste Mitglied (Resource oder Literal) einer Sammlung zugeordnet werden muß und das alle weiteren Mitglieder aufsteigend durch RDF:_2 .. RDF:_n zugeordnet werden.

Es ist jedoch nicht definiert, wie zu verfahren ist, wenn Mitglieder aus einer Sammlung entfernt oder einer Sammlung hinzugefügt werden.

Abbildung 3.6 zeigt die mengentheoretischen Zusammenhänge am Beispiel aus dem vorangegangenen Abschnitt. `Statement_2` ist Element der Menge `Statements`, welches Repräsentant der dreistelligen Relation (`http://../HitchhikersGuide`, `lit:creator`, `Douglas Adams`) ist.

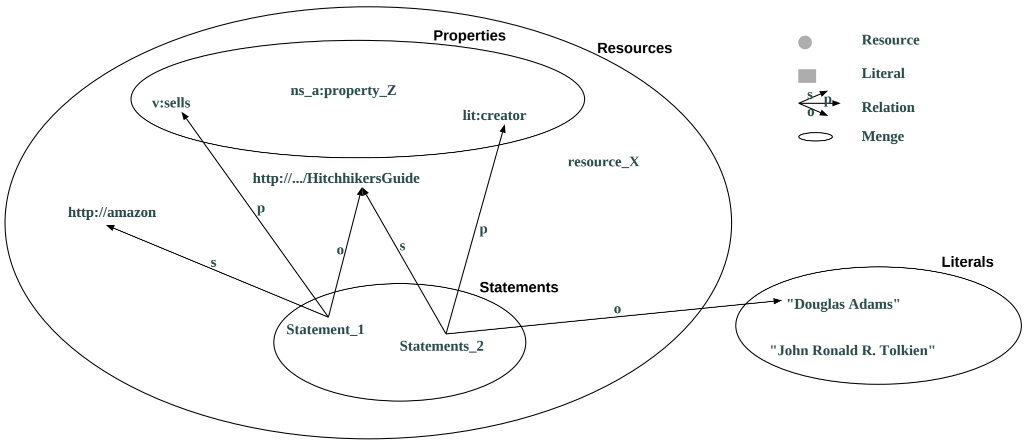

Abbildung 3.6: Mengendarstellung des Beispiels

3.3 RDF-Repräsentationen

Zur Repräsentation der RDF-Ausdrücke stehen drei Notationen zur Verfügung: (1) der RDF-Graph, (2) die Triple-Notation und (3) die RDF-Syntax (welche eine Anwendung von XML ist).[16]

3.3.1 Graphische Notation

Der RDF-Graph besteht, wie schon erläutert, aus Knoten und gerichteten markierten Kanten. Ein Knoten repräsentiert entweder eine Resource oder ein Literal. Eine Resource wird als Oval, das einen Resourcen-Namen (URI) beinhaltet,

[16] In der RDF-M+S-Spezifikation wird angeführt, daß die drei Repräsentationen dieselbe Semantik darstellen. *"... These representations have equivalent meaning. ..."* Dies ist jedoch nicht der Fall. Beispielsweise hat das xml:lang-Attribut der RDF-Syntax keine Entsprechung im RDF-Graph. Weiter definieren die RDF-Spezifikationen keine Transformations-Grammatiken. Das bedeutet: die Transformation von einer Repräsentation (z.B. RDF-Syntax) in eine andere Repräsentation (z.B. RDF-Graph) ist nicht genau beschrieben und daher immer nur eine Interpretation des Autors eines RDF-Transformations-Werkzeugs.

dargestellt und ein Literal als Viereck, das einen String beinhaltet. Ein Prädikat (engl. predicate) wird als gerichtete markierte Kante dargestellt. Die Kante wird mit einem entsprechenden Property-Namen beschriftet.[17] Ein Statement wird durch zwei Knoten, die durch eine gerichtete Kante verbunden sind, dargestellt. Der Knoten, von dem die Kante ausgeht, wird als Subjekt (engl. subject) einer Aussage bezeichnet und der Knoten, auf den die Kante zeigt, als Objekt (engl. object). Vierecke sind Blätter des Graphen, d.h. von Vierecken gehen keine Kanten aus. Vierecke repräsentieren daher immer Objekte. Ein Oval kann sowohl Subjekt als auch Objekt sein. Abbildung 3.7 zeigt den abstrakten Aufbau eines RDF-Graphen (der aus einem Statement besteht):

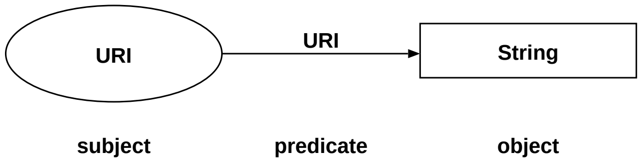

Abbildung 3.7: Einfacher RDF-Graph

Zur Verdeutlichung der Charakteristika von RDF-Graphen wird im folgenden eine mengentheoretische Formalisierung gezeigt.

Ein RDF-Graph kann als gerichteter markierter (Hyper-)Graph aufgefaßt werden. Sei $G = (OvalNodes, SquareNodes, Edges, \mathrm{URIs}, Strings, EdgeLabels)$ ein gerichteter markierter (Hyper-)Graph.[18]

1. $\mathrm{URIs} = \{x | x \text{ ist URI gemäß RFC2396}\}$
   Kommentar: Es existierte eine Menge von URIs. Das Alphabet, aus dem URIs hervorgehen, entspricht dem Alphabet aus RFC2396.

[17] Die RDF-Spezifikationen machen keine klare Aussage darüber, wie der Prädikat-Name im RDF-Graph darzustellen ist: (2) als URI oder (1) als Konkatenation von Namensraum-Präfix:Property-Name (wobei diese Konkatenation in einen URI transformiert werden kann). In den Beispiel-Graphen der Spezifikationen wird Variante 2 genutzt.

[18] In der vorliegenden mengentheoretischen Formalisierung wird das Chrakteristikum, daß ein RDF-Graph azyklisch ist (es gibt keinen Pfad von einem Knoten, der zu sich selbst führt) nicht einbezogen. Die RDF-Core-Working-Group (eine Arbeitsgemeinschaft des W3Cs, welche die Weiterentwicklung von RDF steuert) hat die Entscheidung getroffen, Zyklen in RDF-Graphen zukünftig zuzulassen, um so eine Möglichkeit zu haben, Gleichheit zu modellieren. Auch die Behandlung von anonymen Knoten wurde nicht gesondert hervorgehoben. In der vorliegenden Formalisierung kann ein anonymer Knoten durch einen URI repräsentiert werden. Diese Widersprüchlichkeit resultiert aus der Widersprüchlichkeit der RDF-Spezifikationen bezüglich dieses Merkmals.

2. $Strings\{x|x$ ist String gemäß der XML-Spezifikation$\}$
   Kommentar: Es existiert eine Menge von Strings (das Alphabet, aus dem die Strings hervorgehen, entspricht dem XML-Alphabet für Strings).

3. $OvalNodes \subset$ URIs
   Kommentar: Es existiert eine Menge von URIs in ovalen Knoten.

4. $SquareNodes \subset Strings$
   Kommentar: Es existiert eine Menge von Strings in viereckige Knoten.

5. $OvalNodes \cap SquareNode = \emptyset$
   Kommentar: Die Mengen OvalNodes und SquarNodes sind disjunkt, d.h. es gibt keine Resourcen, die Literale sind, und vice versa. Das bedeutet aber nicht, daß es keinen String gibt, der syntaktisch einem URI entspricht.

6. $EdgeLabels \subset$ URIs
   Kommentar: Es existiert eine Menge von URIs (bzw. XML-Namensräume, die zumindest in URIs transformiert werden können), die Markierung von gerichteten Kanten sind.

7. $Edges \subset EdgeLabels \times OvalNodes \times (OvalNodes \cup SquareNodes)$
   Kommentar: Menge aller gerichteten markierten Kanten. Es handelt sich um eine dreistellige Relation. Der Eintrag an erster Stelle wird Prädikat genannt, der an zweiter Stelle Subjekt und der an dritter Stelle Objekt. OvalNodes dürfen als Subjekt und als Objekt vorkommen. SquareNodes dürfen nur als Objekt vorkommen.

Eine Verbindung der hier gezeigten mengentheoretischen Definitionen für RDF-Graphen zu den mengentheoretischen Definitionen für das RDF-Datenmodell kann durch folgende Definition hergestellt werden:

- $Edges = Statements$

Die in diesem Abschnitt gezeigten mengentheoretischen Betrachtungen beziehen sich ausschließlich auf die grundlegenden Bestandteile von RDF-Graphen. Die Semantik spezieller Knoten- oder Kanten-Markierungen wurden nicht untersucht. Die Interpretation bestimmter Knoten- und Kanten-Markierungen resultiert in weiteren Zwängen für RDF-Graphen. Beispielweise lassen sich mit dem Prädikat RDF:subClassOf Klassenhierarchien modellieren. Hierbei gilt die Einschränkung, daß eine Klasse nicht ihre eigene Oberklasse sein darf. Graphgrammatisch betrachtet bedeutet dies, daß es keinen Pfad von einem Knoten über Kanten mit der Kanten-Markierung RDFS:subClassOf zu sich selbst geben darf. Derartige Betrachtungen sind jedoch nicht Gegenstand dieses Abschnitts, sondern werden in Kapitel 4 behandelt. Weiter wurde keine Graphtransformationsgrammatik angegeben, d.h. es ist nicht definiert, wie bei Modifikation (Einfügen und Löschen von Knoten oder Kanten) eines RDF-Graphen verfahren werden muß. Diesbezüglich machen die RDF-Spezifikationen keinerlei Aussagen.

3.3.2 Triple-Notation

Abschnitt 5 der RDF-M+S-Spezifikation zeigt eine Triple-Notation in der RDF-Modelle aufgeschrieben werden können. Die Triple-Notation wird ausschließlich anhand von Beispielen eingeführt. Eine Formalisierung hierfür gibt die RDF-M+S-Spezifikation nicht an. Abbildung 3.8 zeigt das Beispiel-Statement aus Abbildung 3.1 in Triple-Notation:

```
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],"Douglas Adams"}
```

Abbildung 3.8: Beispiel-Statement in Triple-Notation

Im folgenden wird eine Grammatik für die Triple-Notation in EBNF-Notation angegeben (für eine Erläuterung der EBNF-Notation siehe Abschnitt 6 der XML-Spezifikation). Diese Formalisierung ist eine Interpretation der Triple-Beispiele aus der RDF-M+S-Spezifikation.

```
triple    ::= '{' predicate',' subject',' object '}'
predicate ::=     URI | QName
subject   ::= '[' URI | QName ']'
object    ::= '[' URI | QName ']' | String
URI       ::= (URI gemäß RFC2396)
String    ::= '"'(String gemäß der XML-Spezifikation)'"'
QName     ::= (siehe Produktion 6 aus der XML-Namespace-Spezifikation)
```

Abbildung 3.9: EBNF-Grammatik der Triple-Notation

Aus den Triple-Beispielen in der RDF-M+S-Spezifikation wird nicht klar, ob für ein Prädikat der entsprechende URI oder der QName in einem Triple angegeben werden muß. In der vorliegenden Grammatik gilt diesbezüglich Wahlfreiheit. Zur Überprüfung der Übereinstimmung eines Prädikats, das durch eine QNamen repräsentiert wird, mit einem Prädikat, das durch einen URI repräsentiert wird, ist der QName in den durch ihn repräsentierten URI zu transformieren.

Dies gilt auch für Subjekte und Objekte, wobei ein URI oder ein QName in eckigen Klammern zu schreiben ist. An Objekt-Position dienen eckige Klammern bzw. doppelte Anführungsstriche zur Unterscheidung von Resourcen und Literalen. An Subjekt-Position ist diese Unterscheidung eigentlich nicht notwendig, da nur Resourcen an Subjekt-Position erlaubt sind. Es ist deshalb nicht klar, warum in der RDF-M+S-Spezifikation die Schreibweise bzgl. eines Subjektes nicht äquivalent zur Schreibweise eines Prädikates (also ohne eckige Klammern) gehalten wird.

Anhand der hier angegebenen EBNF-Notation wird deutlich, daß bei der Triple-Notation die Typisierung einer Entität nicht nur über die Stellung im Triple erfolgt, sondern zusätzlich durch weitere syntaktische Elemente (eckige Klammern und doppelte Anführungstriche).

Der Working-Draft *RDF Test Cases* [71] definiert im Abschnitt 3 ebenfalls eine Triple-Notation (sogn. N-Triples) in EBNF-Notation. Die N-Triple-Notation ist ein zeilenbasiertes textuelles (7-bit US-ASCII) Format, durch das ein RDF-Graph linearisiert werden kann. Die N-Triple-Notation ist eine eingeschränkte Version der N3-Notation [14, 15]. Sowohl die N-Triple-Notation als auch die N3-Notation enthalten jedoch grammatikalische Elemente, über die die RDF-M+S-Spezifikation keine Aussage macht und die über die darin gezeigte (einfache) Triple-Notation hinausgehen.

Hinweis: Die RDF-M+S-Spezifikation definiert: *a triple* $\{p, s, o\}$ *is an arc from s to o, labeled by p.* Faßt man $Triple$ als Menge von `triple`[19] auf so kann mit $Triple = Edges$ eine Verbindung zum RDF-Graphen hergestellt werden und über $Edges = Statement$ zum RDF-Datenmodell.

3.3.3 RDF-Syntax

Im Rahmen des einführenden Beispiels zu einer RDF-Beschreibung wurde bereits eine Linearisierung (engl. serialization) eines RDF-Graphen in RDF-Syntax gezeigt (siehe Abbildung 3.2). Abschnitt 6 der RDF-M+S-Spezifikation definiert die Grammatik der RDF-Syntax durch Angabe einer entsprechenden EBNF-Notation (siehe auch Anhang 9.3). Durch Anwendung der EBNF-Produktionen läßt sich ein RDF-Dokument erstellen, das in seiner syntaktischen Struktur einem XML-Dokument entspricht. RDF-Dokumente sind wohlgeformte (engl. well-formed) XML-Dokumente. Eine Validierung eines RDF-

[19] Unter Vernachlässigung der syntaktischen Elemente '[', ']', ',' und '"'.

Dokumentes kann jedoch nicht durch XML-Werkzeuge[20] vorgenommen werden, da keine XML-DTD für die RDF-Syntax definierbar ist. Dies ist aufgrund der, durch die EBNF definierten syntaktischen Struktur von RDF-Dokumenten für den generellen Fall nicht möglich, da diese von Vokabular abhängig ist, das Benutzer in RDF-Schemata definieren können (z.B. wegen der typedNodes - siehe Produktion 6.13 der EBNF-Grammatik zur RDF-Syntax). Diese Einschränkung durch die RDF-Syntax und deren Komplexität verwirrt viele Benutzer und ist unter Vernachlässigung bestimmter Anforderungen[21] an die RDF-Syntax auch nicht notwendig. Abschnitt 7 dieser Arbeit zeigt eine einfache XML-Syntax zum Austausch von RDF-Beschreibungen. Die RDF-Syntax wird von der RDF-Core-Working-Group derzeit überarbeitet. [7] zeigt die bisherigen Ergebnisse bezüglich der Überarbeitung der RDF-Syntax.[22] Einen entsprechenden Standard gibt es jedoch noch nicht. Es ist Zielvorgabe der RDF-Core-Working-Group, die durch [7] revidierte RDF-Syntax weitgehend kompatibel zur aktuell gültigen (standardisierten) RDF-Syntax zu halten. Im folgenden wird die aktuell gültige RDF-Syntax im Detail diskutiert und an Stellen, an denen es notwendig ist, auf zukünftige Änderungen hingewiesen.

Es gibt zwei Arten von RDF-Syntax: (1) die *Basic Serialization Syntax* (siehe Abschnitt 2.2.1 der RDF-M+S-Spezifikation) und (2) die *Basic Abbreviated Syntax* (siehe Abschnitt 2.2.1 der RDF-M+S-Spezifikation). Ziel beim Entwurf der Basic-Abbreviated-Syntax war es, eine, im Vergleich zur Basic-Serialization-Syntax kompaktere und für den Menschen lesbarere Syntax zur Repräsentation von RDF-Beschreibungen zu entwickeln. Die Basic-Abbreviated-Syntax ist jedoch nicht so ausdrucksstark wie die Basic-Serialization-Syntax (d.h. nicht alle RDF-Beschreibungen lassen sich in Basic-Abbreviated-Syntax ausdrücken). Basic-Serialization-Syntax und Basic-Abbreviated-Syntax könne gemischt verwendet werden, d.h. ein Teil einer Beschreibung kann in der kompakteren Syntax verfaßt werden, und der Rest in der ausdrucksstärkeren Basic-Serialization-Syntax.

Die RDF-Syntax wird im folgenden entlang der in Abschnitt 6 der RDF-M+S-Spezifikation gezeigten EBNF-Definitionen diskutiert.

[20] Eine Validierung eines RDF-Dokumentes gegen die Constraints die ein RDF-Schema definiert ist jedoch durch entsprechende RDF-Werkzeuge möglich.

[21] Die RDF-M+S-Spezifikationen legen nicht klar und eindeutig die Anforderungen fest, die die RDF-Syntax erfüllt bzw. erfüllen können soll. Anforderungen werden an verschiedenen Stellen der Spezifikationsdokumente erwähnt. Einige Anforderungen sind vage bzw. implizit angegeben.

[22] Aber auch für diese RDF-Syntax wird es, nach dem derzeitigen Stand der Diskussionen in den RDF-Mailinglilsten und der vorliegenden Arbeitspapiere (engl. working drafts), keine XML-DTD geben.

```
[6.1] RDF           ::= ['<rdf:RDF>'] obj* ['</rdf:RDF>']
```

Eine RDF-Beschreibung wird durch die Tags `<rdf:RDF>` und `</rdf:RDF>` gekennzeichnet. rdf: ist das Namensraum-Präfix, welches das RDF-Schema[23] referenziert, indem die Konzepte der RDF-M+S-Spezifikation beschrieben sind.

```
[6.2] obj           ::= description | container
```

Eine RDF-Beschreibung ist entweder eine `description`, die ein oder mehrere Statements beschreibt oder ein `container`, d.h. eine Sammlung von Inline-[24] Resourcen oder Resourcen-Referenzen.

Descriptions

Eine Description (vgl. Produktion `description`) beschreibt eine Sammlung von (einem oder mehreren) Statements[25], wobei jedes Statement eine Aussage über das gleiche Subjekt enthält.

```
[6.3]  description    ::= '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '/>'
                        |'<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '>'
                             propertyElt* '</rdf:Description>'
                        | typedNode
[6.13] typedNode      ::= '<' typeName idAboutAttr? bagIdAttr? propAttr* '/>'
                        |'<' typeName idAboutAttr? bagIdAttr? propAttr* '>'
                              propertyElt* '</' typeName '>'
[6.15] typeName       ::= Qname
[6.5]  idAboutAttr    ::= idAttr | aboutAttr | aboutEachAttr
[6.6]  idAttr         ::= ' ID="' IDsymbol '"'
[6.7]  aboutAttr      ::= ' about="' URI-reference '"'
[6.8]  aboutEachAttr  ::= ' aboutEach="' URI-reference '"'
                        | ' aboutEachPrefix="' string '"'
[6.9]  bagIdAttr      ::= ' bagID="' IDsymbol '"'
[6.10] propAttr       ::= typeAttr
                        | propName '="' string '"' (with embedded quotes escaped)
[6.11] typeAttr       ::= ' type="' URI-reference '"'
[6.14] propName       ::= Qname
[6.19] Qname          ::= [ NSprefix ':' ] name
[6.20] URI-reference  ::= string, interpreted per [URI]
[6.21] IDsymbol       ::= (any legal XML name symbol)
[6.22] name           ::= (any legal XML name symbol)
[6.23] NSprefix       ::= (any legal XML namespace prefix)
[6.24] string         ::= (any XML text, with "<", ">", and "&" escaped)
```

[23] http://www.w3.org/1999/02/22-rdf-syntax-ns

[24] Eine Resource wird als Inline-Resource bezeichnet, wenn diese innerhalb der RDF-Beschreibung definiert wird.

[25] Eine Description ist implizit ein RDF-Container, der die reifizierten Statements aller durch die *Description* beschriebenen Aussagen enthält.

Es gibt verschiedene syntaktische Ausprägungen einer Description, die jeweils als XML-Elemente realisiert sind. Allen gemeinsam ist die Verwendung folgender XML-Attribute. Durch XML-Attribute gemäß der Produktion `idAboutAttr`[26] wird das Subjekt definiert. Das XML-Attribut `about=URI-reference` referenziert mittels des URIs das Subjekt, über das eine Aussage gemacht wird. Wird stattdessen das XML-Attribut `ID=IDsymbol` verwendet, handelt es sich um eine Inline-Resource, über die eine Aussage gemacht wird. Wenn keines der beiden Attribute angeben ist, handelt es sich um eine anonyme Subjekt-Resource. Das Attribut `bagID=IDsymbol` definiert einen Bezeichner für die Description, welcher verwendet werden kann, wenn über die Description (d.h. den Container, der eine Sammlung von Statements enthält) eine Aussage gemacht werden soll. Wird keine `bagID` definiert, handelt es sich um eine anonyme Description. Durch XML-Attribute gemäß der Produktion `propAttr` werden Prädikat und Objekt des/der Statements angegeben. Ein XML-Attribut `propName=String` definiert Prädikat (propName)[27] und Objekt (String – entspricht einem Literal) eines Statements, wobei propName ein in einem RDF-Schema definierter Prädikat-Name ist. Durch Angabe mehrerer derartiger XML-Attribute werden mehrere Statements definiert. Das XML-Attribut `type=URI-reference` definiert den Typ des Subjekts.

Es gibt folgende syntaktische Ausprägungen von Descriptions:

Repräsentation von Prädikaten durch XML-Attribute. Diese Möglichkeit kann als Kurzschreibweise genutzt werden, wenn alle für Prädikate angegebenen Werte (Objekte), Literale sind. Bei dieser Schreibweise ist es nicht möglich, das gleiche Prädikat mehrfach (mit evt. verschiedenen Werten) zu verwenden.

Beispiel:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lit="http://www.schema.org/lit#">
  <rdf:Description
       rdf:about="http://www.galaxy.gal/HitchhikersGuide"
       lit:creator="Douglas Adams"/>
</rdf:RDF>
```

Ist in Triple-Notation:

[26] Auf das aboutEachAttr wird nicht näher eingegangen, da es in der zukünftigen Syntax (vgl. [7]) nicht mehr verwendet wird.

[27] *propName* ist ein QName (siehe auch XML-Spezifikation), d.h. ein Name inklusive XML-Namensraumpräfix.

```
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],"Douglas Adams"}
```

Repräsentation von Prädikaten durch XML-Elemente. Hiermit ist es möglich, ein Prädikat mehrfach anzugeben. Außerdem können als Werte (Objekte) für Prädikate Literale *und* Resourcen (inklusive Containern und weiterer Descriptions) angegeben werden. Diese Schreibweise wird auch **Stripped-Syntax** (vgl. [26, 113]) genannt, da Knotenrepräsentaten (Subjekt oder Objekt) und Prädikatrepräsentanten verschachtelt notiert werden (können), so daß ein Pfad durch einen RDF-Graphen beschrieben wird. Die Produktion `propertyElt` definiert, daß Prädikate statt durch XML-Attribute durch XML-Elemente beschrieben werden können.

```
[6.12] propertyElt    ::= '<' propName idAttr? '>' value '</' propName '>'
                        | '<' propName idAttr? parseLiteral '>'
                              literal '</' propName '>'
                        | '<' propName idAttr? parseResource '>'
                              propertyElt* '</' propName '>'
                        | '<' propName idRefAttr? bagIdAttr? propAttr* '/>'
[6.16] idRefAttr      ::= idAttr | resourceAttr
[6.17] value          ::= obj | string
[6.18] resourceAttr   ::= ' resource="' URI-reference '"'
[6.32] parseLiteral   ::= ' parseType="Literal"'
[6.33] parseResource  ::= ' parseType="Resource"'
```

Der Wert des (optionalen) XML-Attributs `ID` (siehe Produktion `idAttr`) eines `propName`-Elements ist der Bezeichner des Reifikanten des Statements. Optional kann das XML-Attribut `parseType` angegeben werden, mit dem die Interpretation des `propName`-Elementinhalts beeinflußt wird. Zulässige Attribut-Werte sind `Resource` und `Literal`. Der Wert `Resource` bedeutet, daß der Inhalt als (weitere) RDF-Beschreibung zu interpretieren ist. Der Wert `Literal` bedeutet, daß der Wert als Literal zu interpretieren ist. Damit ist es möglich, als Wert eines Prädikats statt Strings auch XML-Markup angegeben zu können. Der Wert des optionalen XML-Attributs `resource` eines `propName`-Elements ist der URI einer Resource, die als Objekt eines Statements verwendet werden soll (d.h. in diesem Fall wird keine Inline-Resource definiert).

Beispiel:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lit="http://www.schema.org/lit#">
```

```
  <rdf:Description rdf:about="http://www.galaxy.gal/HitchhikersGuide">
       <lit:creator>Douglas Adams</lit:creator>
       <lit:creator>Second Author</lit:creator>
       <lit:creator rdf:resource="http://www.autor.org/ThirdAuthor"
  </rdf:Description>
</rdf:RDF>
```

Ist in Triple-Notation:

```
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],"Douglas Adams"}
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],"Second Author"}
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],
              [http://www.autor.org/ThirdAuthor]}
```

XML-Element als Repräsentant für typisierten Knoten. Der XML-Element-Name ergibt sich aus dem Klassen-Namen (bzw. Konzept-Namen), der durch einen Benutzer in einem RDF-Schema definiert ist. Dies kann als Kurzschreibweise genutzt werden, bei welcher der Typ der Subjekt-Resource durch den XML-Elementnamen bezeichnet wird (bzw. hierüber ermittelt werden kann). Prädikate können als XML-Attribute (mit den gleichen Einschränkungen wie oben beschrieben) und als XML-Elemente (mit denen Pfade durch den RDF-Graphen beschrieben werden können) notiert werden.

Beispiel:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lit="http://www.schema.org/lit#">
  <lit:Book rdf:about="http://www.galaxy.gal/HitchhikersGuide"
            lit:creator="Douglas Adams"/>
</rdf:RDF>
```

Ist in Triple-Notation:

```
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],"Douglas Adams"}
{rdf:type, [http://www.galaxy.gal/HitchhikersGuide],
           [http://www.schema.org/lit#Book]}
```

Container

Zur Beschreibung von Sammlungen von Resourcen und/oder Literalen dienen Container. Es werden drei Arten von Container unterschieden:

1. Bag-Container können verwendet werden, wenn die Reihenfolge in der Resourcen/Literale angegeben werden, bei der Ermittlung der Semantik irrelevant ist.

2. Seq-Container können verwendet werden, wenn die Reihenfolge relevant ist.

3. Alt-Container können verwendet werden, wenn die Resourcen/Literale des Containers als Alternativen zu interpretieren sind.

```
[6.4] container       ::= sequence | bag | alternative
[6.25] sequence       ::= '<rdf:Seq' idAttr? '>' member* '</rdf:Seq>'
                          | '<rdf:Seq' idAttr? memberAttr* '/>'
[6.26] bag            ::= '<rdf:Bag' idAttr? '>' member* '</rdf:Bag>'
                          | '<rdf:Bag' idAttr? memberAttr* '/>'
[6.27] alternative    ::= '<rdf:Alt' idAttr? '>' member+ '</rdf:Alt>'
                          | '<rdf:Alt' idAttr? memberAttr? '/>'
[6.28] member         ::= referencedItem | inlineItem
[6.29] referencedItem ::= '<rdf:li' resourceAttr '/>'
[6.30] inlineItem     ::= '<rdf:li' '>' value </rdf:li>'
                          | '<rdf:li' parseLiteral '>' literal </rdf:li>'
                          | '<rdf:li' parseResource '>' propertyElt* </rdf:li>'
[6.31] memberAttr     ::= ' rdf:_n="' string '"' (where n is an integer)
[6.34] literal        ::= (any well-formed XML)
```

Die Mitglieder (`member`) eines Containers können entweder durch XML-Attribute oder durch XML-Elemente beschrieben werden. Werden XML-Attribute verwendet, können als Mitglieder des Containers nur Literale angegeben werden. Der XML-Attribut-Name entspricht dann `rdf:_n`, wobei n eine natürliche Zahl ist (über die im Falle eines Seq- oder Alt-Containers die Reihenfolge ermittelt wird). Alternativ kann das XML-Element `rdf:li` verwendet werden, womit zusätzlich zu Literalen auch Resourcen als Mitglieder eines Containers beschrieben werden können. Die Semantik der XML-Attribute `ID`, `resource` und `parseType` wurde bereits beschrieben.[28]

Beispiel:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lit="http://www.schema.org/lit#">
```

[28] Anmerkung: Durch wenige Änderungen der RDF-Syntax können Container als typisierte Knoten (typedNode) dargestellt werden. Im Diskussionsdokument *A Proposed Interpretation of RDF Containers* [117] wird eine entsprechende Änderung/Interpreation der Container-Syntax vorgeschlagen. Im Draft *RDF/XML Syntax Specification (Revised)* [7] wird dies berücksichtigt.

```
  <rdf:Description rdf:about="http://www.galaxy.gal/HitchhikersGuide">
    <lit:creator>
      <rdf:Seq ID="authors">
        <rdf:li>Douglas Adams</rdf:li>
        <rdf:li>Second Author</rdf:li>
        <rdf:li rdf:resource="http://www.autor.org/ThirdAuthor"/>
      </rdf:Seq>
    <lit:creator>
  </rdf:Description>
</rdf:RDF>
```

Ist in Triple-Notation:[29]

```
{lit:creator, [http://www.galaxy.gal/HitchhikersGuide],[#autors]}
{rdf:type,    [#autors], [http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq]}
{rdf:_1,      [#autors], "Douglas Adams"}
{rdf:_2,      [#autors], "Second Author"}
{rdf:_3,      [#autors], [http://www.autor.org/ThirdAuthor]}
```

Weitere Konzepte werden durch die RDF-Schema-Spezifikation definiert.

3.4 RDF-Schema

Die RDF-Schema-Spezifikation[30] definiert Vokabular, (in dieser Arbeit RDF-Meta-Schema genannt), mit dem RDF-Schemata erstellt werden können. Ein RDF-Schema definiert Vokabular das zur Beschreibung von Resourcen genutzt werden kann. Im Folgenden wird eine mengentheoretische Darstellung des RDF-Meta-Schema-Vokabulars vorgestellt. Eine Formalisierung in Prädikatenlogik erster Ordnung wird in Kapitel 4 vorgestellt.

1. rdfs:Class $\in$ $Classes$:
   $Kommentar$: rdfs:Class wird genutzt, um eine Resource als Typisierungskonzept zu definieren. Steht eine Resource r in Typ-Relation (Instanz-Relation) zu der Resource rdfs:Class ist r Element der Menge Classes. rdfs:Class selbst ist Element der Menge Classes.

   Beispiel:
   `{rdf:type,[http://../HitchhikersGuide],[lit:Book]}`

[29] Das Zeichen `#` steht für den relativen URI, der das Dokument referenziert, in der die RDF-Beschreibung selbst enthalten ist.

[30] Der Working Draft *RDF Vocabulary Description Language 1.0: RDF Schema* [28] ist eine Revision der RDF-Schema-Spezifikation, die jedoch noch nicht abgeschlossen ist. Im folgenden Text wird an den Stellen, an denen es notwendig ist, auf die Änderungen, die sich aus der Revision ergeben hingewiesen.

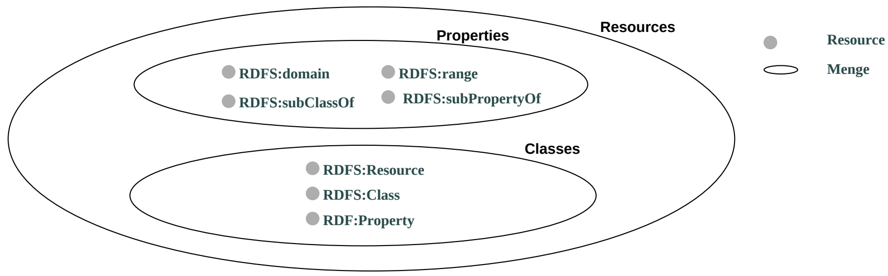

Abbildung 3.10: RDFS-Konzepte

2. rdfs:Resource $\in Classes$:
   Kommentar: Alle "Dinge" die durch RDF-Ausdrücke beschrieben werden können, sind (implizit) Resourcen, d.h. alle Resourcen stehen in Typ-Relation zur Resource rdfs:Resource und sind damit Elemente der Menge Resources. Die Resource rdfs:Resource ist Element der Menge Classes.

3. rdf:Property $\in Classes$:
   Kommentar: Alle Resourcen, die in Typ-Relation zur Resource rdf:Property stehen, sind Elemente der Menge Properties. Ein Element der Menge Properties kann in einem Statement an Prädikat-Position genutzt werden.

   Beispiel:
   `{rdf:type,[http://www.schema.org/lit#],[rdf:Property]}`

4. rdfs:domain $\in Properties$:[31]
   Kommentar: Für ein Property p kann ein Domain-Wertebereich definiert werden, d.h. es kann festgelegt werden, welche Typen von Resourcen an Subjekt-Position erlaubt sind.

   Beispiel:
   `{rdfs:domain,[lit:creator],[lit:Book]}`

5. rdfs:range $\in Properties$:
   Kommentar: Für ein Property p kann ein Wertebereich definiert werden. Der Range eines Properties legt fest, welche Typen an Objekt-Position erlaubt sind.

   Beispiel:
   `{rdfs:range,[lit:creator],[rdfs:Literal]}`[32]

[31] Diese Aussage ist eine Vereinfachung. rdfs:domain und rdfs:range sind Elemente der Menge rdfs:ConstraintProperties, welche in subClassOf-Relation zu der Menge Properties steht. Diese Zusammenhänge werden hier nicht dargestellt, da die RDF-Core-Working-Group beschlossen hat, diese Mengen aus dem RDF-Schema-Vokabular zu eliminieren (vgl.http://lists.w3.org/Archives/Public/w3c-rdfcore-wg/2001Nov/0294.html).

[32] Anmerkung: Es wäre sinnvoller Instanzen eines Konzepts/Typs Creator/Author als

6. rdfs:subClassOf $\in$ *Properties*:
   Kommentar: Ein Element A der Menge Classes kann in subClassOf-Relation zu einem Element B der Menge Classes stehen. Steht B in subClassOf-Relation zu A, dann ist ein Element b der Menge B auch Element der Menge A. Eine subClassOf-Relation definiert eine transitive Beziehung zwischen Mengen.

   Beispiel:
   `{rdfs:subClassOf,[lit:Proceedings],[lit:Book]}`

   Gegeben sei:
   `{rdf:type,[http://www2002.org/CDROM/],[lit:Proceedings]}`
   daraus folgt:
   `{rdf:type,[http://www2002.org/CDROM/],[lit:Book]}`

7. rdfs:subPropertyOf $\in$ *Properties*:

   Kommentar: Ein Element sp der Menge Properties kann in subPropertyOf-Relation zu einem Element p der Menge Properties stehen. Wenn dannein Statement $\{sp, [A], [B]\}$ existiert, kann $\{p, [A], [B]\}$ geschlossen werden.

   Beispiel:
   `{rdfs:subPropertyOf,[lit:author],[lit:creator]}`

   Gegeben sei:
   `{lit:Book,[lit:author],"Douglas Adams"}`
   daraus folgt:
   `{lit:Book,[lit:creator],"Douglas Adams"}`

8. rdfs:seeAlso $\in$ *Properties*
   rdfs:isDefinedBy $\in$ *Properties*
   rdfs:comment $\in$ *Properties*
   rdfs:label $\in$ *Properties*:
   Kommentar: Diese Properties werden in einer zukünftigen Version des RDF-Meta-Schemas nicht mehr vorhanden sein, da die Modelltheorie (siehe [76, 77]) keine Constraints bzgl. der Bedeutung dieser Properties definiert. In der hier gezeigten Mengendarstellung werden die Properties, deshalb nicht näher betrachtet.

rdfs:range zu spezifizieren. Literal wurde gewählt, um konsistent mit den einführenden Beispielen zu bleiben. Für die einführende Beispiele wurde rdfs:Literal gewählt, um dieses Konzept erklärend einzuführen.

3.5 Diskussion

XML dient dem interoperablen Austausch von strukturierten Daten und RDF soll zusätzlich dem interoperablen Austausch von semantisch interpretierbaren Daten ermöglichen (für Arbeiten die diesen Zusammenhang behandeln siehe [11, 16, 53, 137]).

Die in der RDF-M+S-Spezifikation vorgestellte XML-Syntax ist nur eines von vielen möglichen Austauschformaten für RDF-Beschreibungen. Die derzeit von RDF genutzte XML-Syntax wird durchaus kritisch diskutiert (siehe z.B. [107, 108, 106, 109, 110, 111]). Die XML-Syntax ist verwirrend und komplex. Dies wird besonders deutlich, wenn ein komplexer RDF-Graph in der gemischten Schreibweise (basic und abbreviated Syntax) serialisiert wird.[33] Außerdem basiert die von RDF genutzte XML-Syntax nicht auf einer XML-DTD oder einem XML-Schema (es ist nicht möglich, eine allgemeingültige DTD für RDF-Syntax zu entwerfen). Die Komplexität der RDF-Syntax resultiert unter anderem aus der Anforderung, daß RDF-Datenmodelle auch von älteren HTML- und XML-unterstützenden Systemen verarbeitbar sein sollen, bzw. daß die Verwendung von RDF deren Funktionalität zumindest nicht einschränken soll. Bereits 1999 wurde beschlossen (vgl. [174]), daß eine vereinfachte XML-Syntax entwickelt werden soll. Ziel der *RDF Core Working Group* [147] ist, die überarbeitete RDF-Syntax (weitgehend) kompatibel zur aktuell gültigen RDF-Syntax zu halten. Der Working-Draft *RDF/XML Syntax Specification (Revised)* [7] faßt die bisherigen diesbezüglichen Ergebnisse zusammen. Der Aufbau der Syntax hat sich allerding bisher nicht wesentlich verändert[34] und leidet deshalb weiter unter deren Komplexität/Unverständlichkeit.

Im Kapitel 7 dieser Arbeit wird ein alternatives Datenmodell und eine dazugehörige XML-Syntax vorgeschlagen. Das Datenmodell ist strukturell mächtiger und erlaubt eine verbesserte Behandlung von Reifikation, Container-Konstrukten und strukturellem Kontext. Die XML-Syntax ist strukturell wesentlich einfacher als die RDF-Syntax und hat zudem eine direkte Entsprechung im dazugehörigen Datenmodell. Die Entsprechung ist formal beweisbar (siehe Kapitel 7).

Die RDF-M+S- und die RDF-Schema-Spezifikation enthalten dagegen keine

[33] Um von Menschen besser lesbare RDF-Dokumente aus RDF-Graphen zu erzeugen, schlägt Jeremy J. Carroll in der Arbeit *Unparsing RDF/XML* [31] entsprechende Design-Patterns vor.

[34] Was sicherlich aus der Anforderung resultiert, zur aktuell gültigen RDF-Syntax möglichst kompatibel zu bleiben.

Transformationsgrammatik,[35] die definiert, wie ein RDF-Dokument in eine Triple-Menge oder in einen RDF-Graphen zu transformieren ist. Die EBNF zur RDF-Syntax gibt ausschließlich Bedingungen für die syntaktische Korrektheit eines RDF-Dokuments an. Die Interpretation, d.h. in welche Triple ein RDF-Dokument transformiert werden muß und was diese bedeuten, ist nicht eindeutig bzw. formal definiert. Das führt in der Praxis dazu, das verschiedene RDF-Parser verschiedene Triple-Mengen produzieren. Das kann dazu führen, daß ein und dasselbe RDF-Dokument semantisch unterschiedlich interpretiert wird. Um das zu vermeiden, enthält der Working-Draft zur Revised-RDF-Syntax [7], aufbauend auf dem *XML Information Set* [48], Transformationsregeln, die definieren, wie die einzelnen Bestandteile der RDF-Syntax in entsprechende Bestandteile eines RDF-Graphen zu transformieren sind. Zur Überprüfung der Korrektheit von RDF-Werkzeugen können die Testfälle herangezogen werden, die im Working Draft *RDF Test Cases*[71] definiert werden.

Zur Identifikation von Resourcen benutzt RDF URIs. Die Interpretation der durch den URI referenzierten Resource hängt von dem MIME-Type (vgl. [66, 125]) der Resource ab. Derzeit ist kein MIME-Type für RDF registriert (zum Registrierungsprozeß siehe [67]). Es ist allerdings der RFC 3023 [125] im Standardisierungprozeß, der den MIME-Type application/rdf+xml definiert, mit dem XML-Dokumente als RDF-Dokumente deklariert werden können. Der RFC 3023 definiert jedoch nicht, wie der Inhalt eines RDF-Dokuments semantisch zu interpretieren ist.

Um das zu ermöglichen, ist eine Sprache notwendig, mit der Semantik formal beschrieben werden kann. Die RDF-Schema-Spezifikation definiert Vokabular, das zur Definition von domänenabhängigen Vokabular genutzt werden kann. Das domänenabhängige Vokabular wird durch RDF-Syntax ausgedrückt. Ein RDF-Dokument, das domänenabhängiges Vokabular definiert, wird RDF-Schema genannt. Domänenabhängiges Vokabular dient zur Beschreibung von Resourcen und deren Zusammenhänge in einem bestimmten Anwendungsbereich. Aufbauend auf der RDF-Schema-Spezifikation entwickelt die W3C-Arbeitsgruppe Web-Ontology (WebOnt-WG) [185] derzeit eine Sprache zur Definition von Ontologien, die Web-Ontology-Language (WOL). Dem Draft *Requirements for a Web Ontology Language* [79] kann entnommen werden, was in diesem Zusammenhang

[35] Jeder RDF-Parser enthält jedoch zumindest implizit eine Transformationsgrammatik. Diese ist aber immer nur Interpretation der RDF-Spezifikationen durch den Autor des RDF-Parsers, weshalb dasselbe RDF-Dokument von verschiedenen RDF-Parsern verschieden interpretiert wird. Als Beispiel für eine deklarative Transformationsgrammatik kann das XSLT-Stylesheet dienen, das im Text *Transforming RDF with XSLT* [47] vorgestellt wird.

unter einer Ontologie[36] verstanden wird:

Definition: An ontology defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, databases, and applications that need to share domain information ... Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them (note that here ..., definition is not used in the technical sense understood by logicians). They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable.

Ziel der WebOnto-WG ist die Entwicklung einer Sprache (WOL), die formal Semantik beschreibt und maschinelles Schließen ermöglicht. Die Sprache soll über die grundlegende, in der RDF-Schema-Spezifikation definierte, Semantik hinausgehen. Die RDF-Schema-Spezifikation definiert eine Sprache, die zur Datentypisierung verwendet werden kann. Der Draft *RDF Datatyping* [78] definiert Methoden, wie typisierte Information in RDF ausgedrückt werden kann und wie durch die XML-Schema-Spezifikationen [61, 178, 20] vordefinierte Datentypen verwendet werden können. Zukünftige Sprachen sollen die WOL erweitern können, z.B. um zusätzliche, logikbasierte Fähigkeiten. Als Anwendungsfälle einer solchen Sprache werden Verbesserungen in der Nutzung von Web-basierten Anwendungen gesehen – z.B. im Zusammenhang mit dem Suchen in Web-Portalen, der Beschreibung von Multimedia-Sammlungen, dem Web-Site-Management, einer Design-Dokumentation, intelligenten Agenten oder Ubiquitous Computing (personalisierte allgegenwärtige Informationsdienste). Die W3C-Arbeitsgruppe *Semantic Web Advanced Development* [183] bemüht sich um die Entwicklung von entsprechenden Beispielanwendungen und Prototypen.

Eine W3C-Recommendation, welche die Semantik des RDF-Datenmodells und der in den RDF-Spezifikationen (RDF-M+S/RDFS) beschriebenen Konzepte formal definiert, gibt es derzeit jedoch nicht. Im folgenden Kapitel dieser Arbeit wird eine Interpretation der intendierten Semantik der RDF-M+S- sowie RDF-Schema-Spezifikation vorgenommen und durch Prädikatenlogik erster Ordnung formalisiert. Eine weitere Formalisierung zur Erfassung der Semantik von RDF-M+S/RDFS stellt der W3C-Draft *RDF Model Theory* (RDF-MT) [76, 77] vor.

[36] Zur Verdeutlichung eine Analogie: Eine DTD für ein XML-Dokument, entspricht einer Ontologien/einem RDF-Schema für RDF-Dokumente. Für verschiedene Anwendungsbereiche sollen dementsprechende Ontologien entworfen werden. Ein Mapping zwischen verschiedenen Ontologien könnte den Anwendungsbereich eines (intelligenten) Automaten erweitern. Die Arbeit *Learning to Map between Ontologies on the Semantic Web* [58] stellt einen entsprechenden Ansatz vor.

Die durch die beiden Formalisierungen festgelegten Semantiken weichen voneinander ab, wobei die Kompatibilität durch wenige Änderungen wiederhergestellt werden kann. Im folgenden Kapitel wird an den relevanten Stellen auf die Unterschiede hingewiesen und eine modifizierte Version der in dieser Arbeit vorgestellten Formalisierung gezeigt, so daß die Kompatibilität zur RDF-MT gewährleistet werden kann.

4 Formalisierung der RDF-Semantik mittels Prädikatenlogik

Die eXtensible Markup Language (XML) und RDF ermöglichen die Erstellung interoperabler Informationssysteme. Während XML die syntaktische Interoperabilität unterstützt, zielt RDF auf die semantische Interoperabilität. Das Description Framework (RDF) dient der Erfassung und Ausdrucksmöglichkeit von konzeptuellen Strukturen und Informationen die im Web zur Verfügung gestellt werden sollen. Die semantische Interoperabilität ist jedoch nur gewährleistet, wenn verschiedenen Anwendungen eine RDF-Beschreibung in der gleichen Art und Weise interpretieren. Wichtige Aspekte von RDF sind jedoch informal ausgedrückt, was bei der Interpretation von RDF-Beschreibungen zu Mißverständnissen führen kann. Wird RDF als Wissensrepräsentationssprache betrachtet, gilt die folgende Aussage von Sowa (vgl. *Knowledge Representation: Logical, Philosophical, and Computational Foundations* [161]) auch für RDF.

Logic, Ontology, and Computation: Knowledge representation is a multidisciplinary subject that applies theories and techniques from three other fields:

- Logic provides the formal structure and rules of inference.
- Ontology defines the kinds of things that exist in the application domain.
- Computation supports the applications that distinguish knowledge representation from pure philosophy.

Without logic, a knowledge representation is vague, with no criteria for determining whether statements are redundant or contradictory. Without ontology, the terms and symbols are ill-defined, confused, and confusing. And without computable models, the logic and ontology cannot be implemented in computer programs. Knowledge representation is the application of logic and ontology to the task of constructing computable models for some domain.

Um Mißverständnisse zu vermeiden, ist es deshalb sinnvoll, die RDF-Sprachmittel formal zu definieren. RDF-Beschreibungen können in verschiedenen Repräsentationen dargestellt werden. Eine Repräsentationsmöglichkeit für eine RDF-Beschreibung ist ein semantisches Netz. Die kann als erste Annäherung an eine formalen Definition der RDF-Semantik angesehen werden. Reimer zeigt in [149], daß ein semantisches Netz in Prädikatenlogik erster Ordnung ausgedrückt werden kann. Prädikatenlogik erster Ordnung ist ein gut untersuchtes mathematisches Ausdrucksmittel für Semantik, welches automatisches Schlußfolgern (engl: inferencing/reasoning) aufgrund von syntaktischen Umformungen einer Wissensbasis ermöglicht. Dieses Kapitel zeigt eine Formalisierung der Semantik von RDF[1] in Prädikatenlogik erster Ordnung. Diese Formalisierung hat in der RDF-Community bereits Beachtung gefunden (vgl. z.B. [94, 13]).

Das Kapitels ist wie folgt gegliedert: In Abschnitt 4.1 wird ein kurzer Überblick über Arbeiten gegeben, die sich ebenfalls mit der formalen Definition der RDF-Semantik beschäftigen. Im Abschnitt 4.2 wird ein Zusammenhang zwischen RDF und Wissensrepräsentationssystemen hergestellt. Der Abschnitt 4.3 zeigt die im Rahmen dieser Arbeit gefundenen Regeln und Fakten, die die RDF-Semantik formal in Prädikatenlogik erster Ordnung wiedergeben. Auf Änderungen der RDF-Semantik, die sich aus der in Entwicklung befindlichen RDF-Modelltheorie-Spezifikation (RDF-MT) [76, 77] ergeben, wird ebenfalls in diesem Abschnitt eingegangen. Eine an die RDF-MT angepaßte Version der Formalisierung zeigt der Abschnitt 4.4. Im Abschnitt 4.5 werden mögliche Konsequenzen und Anwendungen der gezeigten Formalisierung diskutiert.

4.1 Verwandte Arbeiten

Ein Beispiel für einen Ansatz, der logikbasierte Formalismen und RDF in Beziehung setzt, wird in *The RDF Schema Specification Revisited* [129] gezeigt. Wegen der erwarteten Schwierigkeiten, die sich aus der Repräsentation der behaupteten Selbstreferenzialität von RDF ergeben, wird die klassische Unterscheidung zwischen Datenmodell und Metadatenmodell durch die Einführung von epistemischen Primitiven (vergl. [149]) eingeführt. Diese selbst sind jedoch nicht im vorgeschlagenen Formalismus ausdrückbar (wie z.B. das verwendete $type\_of$ Property). Dieser Ansatz unterschätzt jedoch die Ausdrucksmächtigkeit des RDF-Modells.

[1] Die in den RDF-Spezifikationen angegebenen Definitionen sind weitgehend informal ausgedrückt, d.h. jede Formalisierung kann zwangsläufig nur eine Auslegung/Interpretation der von den Autoren (der RDF-Spezifikationen) intendierten Semantik sein.

Aus dem im Rahmen dieser Arbeit erstellten logischen Regelwerk geht hervor, daß die strukturelle und semantische Einfachheit bzw. Klarheit von RDF auf der Eigenschaft beruht, das Aussagen über Instanzen, Konzepte, Meta-Konzepte, Meta-Meta-Konzepte, etc. gemacht werden können, ohne einen weiteren expliziten Level von (Meta-)Konstrukten einführen zu müssen, der nicht im Ursprungsmodell ausdrückbar ist.

Die in [129] in Frage gestellte Selbstreferenzierung vor RDF ist eine Selbstbeschreibbarkeit, die gewünscht ist. Im Kern besteht RDF aus einer sehr kleinen Menge von Basiselementen (geordnete Sequenzen – triple) und Basisaxiomen (die im Rahmen dieser Arbeit vorgestellten logischen Regeln und Fakten). Der Eindruck der Selbstreferenzialität (in [129] werden z.B. die Properties `subClassOf` und `type` erwähnt) wird aufgelöst, wenn der Verwendungszusammenhang eines Triple-Bestandteils betrachtet wird. Der Verwendungszusammenhang wird durch die Position im Triple bestimmt. Derselbe Triplebestandteil hat in Abhängigkeit des Vorkommens an verschiedenen Positionen im Triple auch verschiedene Bedeutung.

Ein sehr interessanter Ansatz, der wie die im Rahmen dieser Arbeit vorgestellten RDF-Formalisierung Logik zur Erklärung der RDF-Semantik anwendet, ist das Tutorial von Champin [32]. Der im Rahmen dieser Arbeit vorgestellte Ansatz ist jedoch vollständiger[2] und es wird versucht, klar zwischen dem Repräsentationsformat (z.B. Triple) und dem Wissensmodell (z.B. das Property `instanceOf` zum Ausdruck der Vererbung oder andere Properties zum Ausruck von Constraint-Verletzungen[3], etc.) zu unterscheiden. Außerdem vermeidet das im Rahmen dieser Arbeit vorgestellte logische Regelwerk die Verwendung von negativen Fakten.

Decker et al. stellen in *A Query and Inference Service for RDF* [52] einen weiteren Ansatz vor, der die Verarbeitung von RDF-Triple durch Logik ermöglicht. In der Arbeit wird der Interpreter SiLRI (Simple Logic-based RDF Interpreter) präsentiert. Die Intention der Arbeit ist jedoch nicht, die genaue Semantik von RDF herauszuarbeiten und diese in Logik bereitzustellen, sondern die direkte Verwendung von Triplen als Faktenbasis zu zeigen.

Der in diesem Kapitel gezeigte Ansatz ist genereller: zusätzlich zu der ein-

[2]Diese Aussage bezieht sich auf die Version der Arbeit von Champin aus dem Jahre 2000, die mittlerweile (zeitlich nach der Diskussion des in diesem Kapitel erläuterten Ansatzes in der RDF-Insterest-Mailingliste) ergänzt wurde.

[3]Zum Begriff Constraint: Der Begriff Constraint wird in dieser Arbeit im Sinne der RDF-Schema-Spezifikation[27] verwendet. Ein Constraint beschreibt valide Werte eines Properties.

fachen Transformation von RDF-Triplen in Fakten einer Wissensbasis wird ein logisches Regelwerk zur Verfügung gestellt, welches RDF-Konzepte und -Constraints semantisch erfaßt. Die (transformierten) Fakten und die logischen Regeln können in eine logikbasierte Inferenzmaschine geladen werden, um die intendierte Semantik von RDF-Dokumenten (RDF-Schemata und -Instanzen) auszudrücken.

Um das im folgenden vorgestellte logische Regelwerk zu testen, wurde der Interpreter SiLRI verwendet. Eine SiLRI-konforme Formulierung des Regelwerks in Datalog wird im Anhang 9.7 gezeigt.

Das Regelwerk wurde außerdem mit SWI-Prolog [175] getestet. Eine Prolog-konforme Formulierung des Regelwerks wird im Anhang 9.6 gezeigt. Das SGML-Modul von SWI-Prolog enthält einen RDF-Parser. Dieser ermöglicht es, RDF-Syntax einzulesen und die Statements, die durch eine RDF-Beschreibung ausgedrückt werden, als Fakten einer Wissensbasis zu speichern. Eine Wissensbasis, die nach dem Einlesen einer RDF-Beschreibung zur Verfügung steht, kann mit Prolog-Prädikaten gemäß der hier vorgestellten Fakten und Regeln befragt werden.

4.2 Zusammenhang von RDF und Wissensrepräsentation

In diesem Abschnitt wird erläutert, daß die graphische Repräsentation von RDF den semantischen Netzen zugeordnet werden kann. Semantische Netze können durch Prädikatenlogik erster Ordnung formal beschrieben werden. Sowohl semantische Netze als auch die Prädikatenlogik erster Ordnung sind gut untersuchte Wissensrepräsentationsformate (siehe z.B. [149, 161]). Die auf diesen Gebieten gewonnenen Forschungsergebnisse und Erfahrungen lassen sich durch die Zuordnung des RDF-Graphs zu den semantischen Netzen auch für RDF nutzen.

4.2.1 Semantische Netze

Konzeptuelle Graphen definieren Wissensrepräsentationssprachen basierend auf Konzepten aus der Sprachverarbeitung, der Psychologie und der Philosophie (vgl. [160] S. 69). Unter einem Konzept werden alle konkreten und abstrakten Dinge verstanden, die Beschreibungsgegenstand sind und worüber deshalb Aussagen vorliegen. Reimer definiert den Begriff Konzept folgendermaßen:

Definition - Konzept: (vgl. [149] S. 17)
Ein Konzept ist ein 3-Tupel (Konzeptname, Extension, Intension). Die Extension ist die Menge aller Objekte, die zu dem Konzept gehört, während die Intension die Merkmale angibt, die ein Objekt aufweisen muß, um zu dem Konzept zu gehören. Diese Merkmale werden auch Konzeptmerkmale genannt.

Zusammenfassend schreibt Reimer weiter (vgl. [149] S. 79): Modelle, bei denen Konzepte semantisch miteinander in Beziehung stehen, basieren auf sogenannten assoziativen Beziehungen, welche formal als zweistellige Relationen aufgefaßt werden können. Assoziationsmodelle lassen sich bildlich als Graphen veranschaulichen, in denen Knoten für Konzepte und Kanten für assoziative Beziehungen zwischen Konzepten stehen. Zwei Arten von Graphen können dabei unterschieden werden : (1) Graphen, in denen Kanten nur eines Kantentyps für assoziative Beziehungen verwendet werden. Ein Graph dieser Art wird *assoziatives Netzwerk* genannt. (2) Graphen die verschiedene Kantentypen für verschiedene Typen assoziativer Beziehungen erlauben. Ein Graph dieser Art wird *semantisches Netzwerk* genannt.

Ein RDF-Graph besteht aus Knoten und gerichteten Kanten. Knoten werden entweder mit einem URI (Konzept Resource) oder einem Atom-Wert (Konzept Literal) markiert. Kanten werden durch einen URI markiert und typisiert. Wegen des Vorhandenseins von Knoten und gerichteten typisierten Kanten kann ein RDF-Graph als semantisches Netzwerk aufgefaßt werden. Die Interpretation der Semantik eines RDF-Graphen wird in den RDF-Spezifikationen nicht genau festgelegt. Oft wird die Semantik semantischer Netze nicht formal spezifiziert, was dazu führt, daß verschiedene Benutzer das gleiche semantische Netz unterschiedlich interpretieren. Ohne formal spezifizierte Semantik bleibt unklar, wie ein semantisches Netz zu interpretieren ist, welche Verbindungen zwischen Knoten zulässig sind und wie diese Verbindungen interpretiert werden sollen. Es ist jedoch möglich, semantische Netze in ihrer Semantik formal zu definieren. Reimer zeigt in [149] (vgl. S. 82ff) wie für ein semantisches Netz die Semantik durch eine Reihe von Vereinbarungen festgelegt werden kann. In Anlehnung an Reihmer sollen folgende Vereinbarungen zur Festlegung der Semantik eines RDF-Graphen angenommen werden:

Definition - RDF-Graph: (1) Zwei Knoten mit unterschiedlichen Markierungen (engl. label) werden semantisch unterschieden. (2) Zwei Knoten mit der gleichen Markierung sind nicht erlaubt. (3) Ein Knoten, der Bestandteil der Extension des Literal-Konzepts ist, wird durch eine Markierung repräsentiert, die von einem Rechteck umgeben ist. Ein solcher Knoten wird Literal-Knoten genannt. (4) Ein Knoten, der Bestandteil der Extension des Resource-Konzepts ist, wird durch eine Markierung repräsentiert, die von einem Oval umgeben ist. Ein solcher Knoten wird Resource-Knoten genannt. (5) Resource-Knoten oder Literal-Knoten spezifizieren implizit die Existenz der/des repräsentierten Resource/Literals. (6) Eine gerichtete Kante verbindet einen Resource-Knoten entweder mit einem Literal-Knoten oder mit einem weiteren Resource-Knoten. Eine gerichtete Kante wird mit einem URI markiert. Ein Statement wird durch die Verbindung von zwei Knoten mit einer gerichtete Kante repräsentiert. (7) Alle Statements eines RDF-Graphen sind implizit konjunktiv verknüpft.

4.2.2 Prädikatenlogik

Die Prädikatenlogik erster Ordnung[4] ermöglicht das logische Schließen. Aus einer gegebenen Formel- und Faktenmenge können mittels Inferenzregeln neue Fakten abgeleitet werden, die implizit in einer Formelmenge enthaltene Aussagen explizit darstellen. Hervorzuheben ist, daß der Vorgang des Schlußfolgerns auf rein syntaktischer Basis abläuft. Somit besteht die Möglichkeit, logisches Schlußfolgern automatisch durch einen Rechner durchführen zu lassen.

Semantische Netze können in Prädikatenlogik erster Ordnung ausgedrückt werden (vgl. [149] S. 50). Dies gilt ebenso für RDF-Graphen, wenn man die in Abschnitt 4.2.1 gemachten Vereinbarungen für RDF-Graphen annimmt. Eine Linearisierung eines RDF-Graphen mittels Prädikatensymbolen kann dann wie folgt vorgenommen werden. Eine gerichtete Kante, die zwei Knoten miteinander verbindet kann als dreistellige Relation[5] $arc(n1, p, n2)$ dargestellt werden.

Der Graph aus Abbildung 4.1 kann in Prädikatenlogik folgendermaßen repräsentiert werden.[6]

[4] Für eine Einführung in die mathematische Logik und die Prädikatenlogik siehe z.B. [8, 80, 122, 153, 156].

[5] Alternative Ansätze (vgl. z.B. [52, 114]) benutzen eine zweistellige Relation, wobei der Name der Relation der Kantenmarkierung entspricht. Bei Verwendung der Prädikatenlogik der ersten Ordnung hat dieser Ansatz jedoch den Nachteil, daß keine Abfrage über die Kantenmarkierungen gemacht werden können.

[6] Zur Erreichung einer übersichtlicheren Darstellung, wird die Semantik der RDF-Konzepte in diesem Beispiel nicht berücksichtigt.

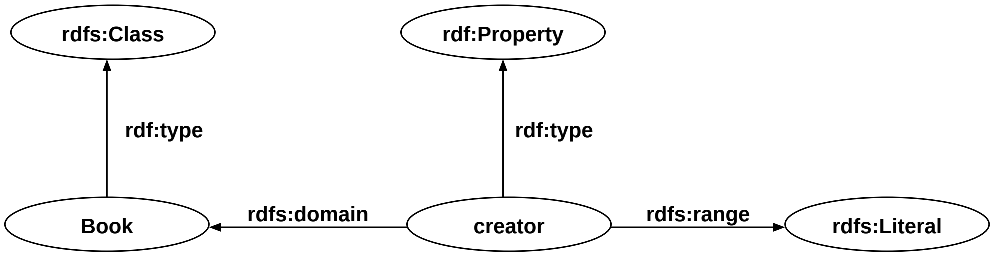

Abbildung 4.1: Beispiel eines RDF-Graphen, der durch implizit konjunktiv verknüpfte Relationen repräsentiert wird

arc('Book', 'rdf:type', 'rdfs:Class')
arc('creator', 'rdfs:domain', 'Book')
arc('creator', 'rdfs:range', 'rdfs:Literal')
arc('creator', 'rdf:type', 'rdfs:Literal')

Die RDF-Spezifikationen legen für bestimmte Konzepte, beispielweise für die Konzepte `rdf:type` und `rdfs:Class`, Semantik fest, die nicht durch das Prädikatensymbol arc ausgedrückt werden kann. Entsprechende Regeln und Fakten, welche die Semantik von RDF-Beschreibungen über das Prädikatensymbol `arc` hinaus erfassen, sind jeder Logik-Repräsentation von RDF-Graphen hinzuzufügen. Im folgenden Kapitel werden Regeln und Fakten definiert, welche zusammen mit dem Prädikatensymbol arc als RDF-Beschreibung interpretiert werden können.

4.3 Interpretation der Semantik von RDF in Prädikatenlogik

Das Prädikatensymbol arc('s', 'p', 'o') sei wahr, wenn eine gerichtete Kante von Knoten s nach Knoten o existiert und diese mit p beschriftet ist.[7]

Das Prädikatensymbol $rfc2396Conform$('x') sei wahr, wenn ein String x gemäß des RFC2396 [17] aufgebaut ist.

Die folgende Regel hebt die Perspektive von der Repräsentationsebene (arc als direkte Entsprechung einer Kante im Repräsentationsformat Graph) auf die Wissensebene (*statement* als Entsprechung der Bedeutung: Fakt ist in die Wissensbasis eingefügt (engl. asserted)).

[7] Ist dieses Prädikatensymbol erfüllt, gibt es eine direkte Entsprechung eines Triples $\{pred, sub, obj\}$ aus der Menge der Triple, die ein RDF-Modell repräsentieren.

$$\forall s,p,o : arc(s,p,o) \wedge rfc2396Conform(p)^{8} \Rightarrow statement(s,p,o)^{9} \qquad (4.1)$$

In den folgenden Unterabschnitten werden die für diese Arbeit relevanten RDF-Konzepte und deren Formulierung als logische Regeln gezeigt. Text, dem das (fettgeschriebene) Wort *Definition* vorangestellt ist, wurde direkt aus [101] (RDF-Definition) oder [27] (RDFS-Definition) übernommen. Es wurde keine Übersetzung des englischen Textes der RDF-Spezifikationen ins Deutsche vorgenommen, da die im folgenden aufgestellten logischen Regeln eine Interpretation des Textes der RDF-Spezifikationen darstellen und die normierenden informalen Texte der RDF-Spezifikationen erhalten bleiben sollen (eine Übersetzung ins Deutsche wäre bereits eine Interpretation der informalen Definitionstexte und damit der RDF-Semantik).

[8] Die RDF-M+S definiert: Jede benannte Resource hat einen URI. Aufgrund dieser Definition könnte angenommen werden, daß das Prädikatensymbol rfc2396Conform(x) grundsätzlich auch auf den subject-String angewendet werden muß. Die RDF-Spezifikation erzwingt dies allerding nur für *benannte* Resourcen. Anonyme Resourcen, die in der graphischen und der XML-Repräsentation vorkommen, führen zu generierten Namen in der Triple-Notation eines RDF-Modells. Diese Namen sind aber nicht notwendigerweise URIs (die meisten derzeit vorhandenen RDF-Parser generieren keine validen URIs für anonyme Resourcen). Um die generierten Namen dennoch in Triplen zuzulassen, wurde in der logischen Regel auf die Überprüfung der Konformität eines subject-Strings mit einem URI verzichtet. Hinweis: das RDF-Modell beschränkt den Namensraum möglicher Bezeichner für Resourcen nicht (hier ist nicht der XML-Namensraum gemeint). In einem vollständigen mengentheoretischen Modell gäbe es ein Alphabet, um anzuzeigen, daß bestimmte Entitäten Elemente der Menge Resource wären. Der Namensraum für Resourcen ist implizit aufgeteilt in URIs, Literale und "generierte" Resourcennamen. Die RDF-M+S bietet keinen Mechanismus, um zu verhindern, daß an Objekt-Position als Literal ein URI geschrieben wird oder das ein URI-konformer String als Name für eine anonyme Resource verwendet wird. Es ist allerdings auch kaum zu argumentieren, daß *eine einzige* Restriktion auf alle Bezeichner angewendet werden muß. Relevant ist die Semantik des Bezeichners – und diese Semantik ist eine Konsequenz aus dem Verwendungszusammenhang, in dem ein Bezeichner benutzt wird. Z.B. ist ein Bezeichner, der an Prädikat-Position in einem RDF-Statement steht, immer ein URI zur Referenzierung eines RDF-Properties.

[9] Marchiori et al schlagen in [114] vor, ein RDF-Statement in ein zweistelliges Prädikatensymbol zu transformieren. Dabei entspricht ein Predicate-Name einem Prädikatensymbolnamen und die dazugehörigen subject- und object-Teile werden als Argumente des Prädikatensymbols eingesetzt. Dies resultiert jedoch in dem Problem, daß nicht mehr Prädikatenlogik erster Ordnung eingesetzt werden kann, wenn nach Predicates gesucht werden soll, bei denen der Name nicht bekannt ist. Hierfür wäre es notwendig, für das Prädikatensymbol eine Variable einzusetzen. Dies ist jedoch nur mit Prädikatenlogik höherer Ordnung lösbar. Aus diesem Grund wird in der hier vorgestellten Regel- und Fakten-Menge das Prädikatensymbol *statement* eingeführt. Suchen nach bestimmten Predicates ist damit auch mit Prädikatenlogik erster Ordnung möglich, da an Predicate-Position in diesem Symbol eine Variable eingesetzt werden kann.

4.3.1 Grundlegende Datenstruktur und Reifikation

Resource

RDFS-Definition: All things being described by RDF expressions are called resources, and are considered to be instances of the class `rdfs:`[10] `Resource`. The RDF class `rdfs:Resource` represents the set called 'Resources' in the formal model for RDF presented in section 5 of the Model and Syntax specification [101].

Elemente der *Menge* `Resources` (siehe Definition 1. Abschnitt 3.2) sind Konzepte in der RDF-Wissensbasis. Gemäß Reimer (vgl. [149]) sind Konzepte generell alle konkreten und abstrakten Dinge die beschrieben werden sollen. In RDF sind die Resourcen die Dinge, die beschrieben werden. RDF unterscheidet anonyme und benannte Resourcen. Benannte Resourcen werden über URIs identifiziert. Der generelle Aufbau von URIs wird im RDF2396 [17][11] definiert. Der Aufbau des Bezeichners einer anonymen Resource[12] ist nicht festgelegt.

Das erste und zweite Argument eines Statements, respektive eines Triples, sind Resourcen, welche durch URIs identifiziert werden. Siehe auch Definition 4 im Abschnitt3.2.[13]

$$\forall s, p, o : statement(s, p, o) \Rightarrow res(s) \wedge uri(p) \wedge obj(o) \tag{4.2}$$

$$\forall r : uri(r) \Rightarrow res(r) \tag{4.3}$$

$$\forall r : res(r) \wedge rfc2396Conform(r) \Rightarrow namedRes(r) \tag{4.4}$$

[10] Der XML-namespace *rdf:* ist Platzhalter für den Namespace-URI *http://www.w3.org/1999/02/22-rdf-syntax-ns#* und *rdfs:* für *http://www.w3.org/2000/01/rdf-schema#*. Hinweis: die Spezifikationen RDF-M+S und RDF-S sind in der Zuordnung von Namesräumen nicht eindeutig sind.

[11] Es sei darauf hingewiesen, daß der RFC3396 [17] den URI als Referenz auf ein Konzept definiert. Die tatsächlich referenzierte Entität kann bei fortschreitender Zeit eine andere sein bzw. in Abhängigkeit vom angefragten MIME-Type werden verschiedenen Entitäten mit dem gleichen URI referenziert. Deshalb sind die durch URIs beschriebenen Referenzen nicht notwendigerweise eindeutig.

[12] Im dem RDF zugrundeliegendem mengentheoretischen Modell benötigt jede Resource einen Bezeichner, um zu existieren, d.h. jede Resource hat einen Bezeichner. Es gibt allerdings anonyme Resourcen in der graphischen- und der XML-Repräsentation, weshalb es nötig ist zwischen benannte Resourcen (mit einem URI als Bezeichner) und anonymen Resourcen (mit einem im Aufbau nicht weiter beschränkten Bezeichner) zu unterscheiden.

[13] Für das logische Prädikat `statement` wird in Anlehnung an die N3-Notation [14] eine andere Ordnung als in Definition 4, Abschnitt3.2, angenommen - `s,p,o` statt `p,s,o`.

Literal

RDF-Definition: The most primitive value type represented in RDF, typically a string of characters. The content of a literal is not interpreted by RDF itself and may contain additional XML markup. Literals are distinguished from Resources in that the RDF model does not permit literals to be the subject of a statement.

Elemente der *Menge* `Literals` (siehe Definition 2, Abschnitt 3.2) sind atomare Werte (Strings), welche immer die Objekt-Position im Statement einnehmen. In einem semantischen RDF-Graph ist es nicht erlaubt, ausgehend von einem Knoten mit Typ Literal eine Kante einzutragen. Deshalb kann keine Aussage über ein Konzept vom Typ Literal gemacht werden. Ein Knoten im semantischen RDF-Graph, der ein Literal repräsentiert, hat einen Bezeichner, der dem atomaren Wert entspricht, den er repräsentiert. Knoten vom Typ Literal werden graphisch als Viereck dargestellt.

$$\forall o : obj(o) \wedge \neg res(o) \Rightarrow lit(o) \tag{4.5}$$

Regel 4.5 könnten möglicherweise in Frage gestellt werden, da ein Literal durchaus ein String gemäß RFC2396 sein kann und deshalb als ein URI interpretiert werden könnte. Wenn solch ein "URI-String" jedoch keine Resource in einem RDF-Modell benennt, dann kann die Eigenschaft `uri` für diesen String nicht geschlossen werden.

$$\forall o : lit(o) \Rightarrow instanceOf(o, \text{'rdfs:Literal'}) \tag{4.6}$$

Wenn das logische Prädikat `lit` für einen String wahr ist, dann ist dieser String auch vom Typ Literal (also eine Instanz des Konzepts Literal). Das Prädikatensymbol `instanceOf` wird durch Regel 4.10 eingeführt.

Property

RDF-Definition: A specific attribute with defined meaning that may be used to describe other resources. A property plus the value of that property for a specific resource is a statement about that resource. A property may define its permitted values as well as the types of resources that may be described with this property.

Zur Erfassung dieses Konzepts ist keine Regel notwendig. Properties sind eine Projektion des zweiten Arguments des logischen Prädikats `statement`. Hinweis: Elemente der Menge Properties sind auch Elemente der Menge Resources (siehe Definition 3. Abschnitt 3.2), und damit gelten alle Regeln für Resourcen auch für Properties.

Statement

RDF-Definition: A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by an URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a literal may have content that is XML markup but is not further evaluated by the RDF processor.

Die entsprechende Semantik wird bereits durch die weiter oben eingeführte Regel 4.1 erfaßt.

Reifikation

Die Reifikation dient dazu, Aussagen über Aussagen bzw. Statements über Statements machen zu können. Die Definition 9, Abschnitt 3.2, legt fest, welche Merkmale die Reifikation in RDF hat. Die Reifizierung eines Statements führt zu einem weiteren Knoten im RDF-Graph, der Repräsentant des Statements ist. Die Resource, die dieser Knoten repräsentiert ist von Typ Statement. Dies wird im RDF-Graph durch eine von diesem Knoten ausgehende Kante mit dem Bezeichner `rdf:type` ausgedrückt. Weitere drei von diesem Knoten ausgehende Kanten tragen die Bezeichner `rdf:subject`, `rdf:predicate` und `rdf:object`.

Die Werte dieser Prädikate stellen die einzelnen Teile eines Statements dar. Regel 4.7 erfaßt die Semantik der Reifizierung eines Statements und Regel 4.8 findet die Resource, die Repräsentant eines reifizierten Statements ist.

$$\begin{aligned}
\forall s, s', p, o : res(s) \wedge uri(p) \wedge \\
statement(s', \text{'rdf:type'}, \text{'rdf:statement'}) \wedge \\
statement(s', \text{'rdf:subject'}, s) \wedge \\
statement(s', \text{'rdf:predicate'}, p) \wedge \\
statement(s', \text{'rdf:object'}, \text{o}) \\
\Rightarrow reifies(s', s, p, o)
\end{aligned} \tag{4.7}$$

$$\forall s, s', p, o : reifies(s', s, p, o) \Rightarrow reifyingStatement(s') \tag{4.8}$$

RDF-Definition: A statement and its corresponding reified statement exist independently in an RDF graph and either may be present without the other. The RDF graph is said to contain the fact given in the statement if and only if the statement is present in the graph, irrespective of whether the corresponding reified statement is present.

$$\begin{aligned}
\forall s, s', p, o : reifies(s', s, p, o) \wedge statement(s, p, o) \\
\Rightarrow reifiesFact(s', s, p, o)
\end{aligned} \tag{4.9}$$

RDF sieht vor, daß die Reifikation eines Statements genutzt werden kann, ohne daß das entsprechende Statement auch im RDF-Modell vorkommen muß. Nur, wenn auch das entsprechende Statement im RDF-Modell vorkommt, handelt es sich um ein Fakt, das in die Wissensbasis eingetragen werden darf. Regel 4.9 findet alle Reifikationen, für die auch Fakten in der Wissensbasis existieren.

4.3.2 RDF-Schema–Regeln

type

RDFS-Definition: This indicates that a resource is a member of a class, and thus has all the characteristics that are to be expected of a member of that class. When a resource has an `rdf:type` property whose value is some specific class, we say that the resource is an instance of the specified class. The value of an `rdf:type` property for some resource is another resource which must be an instance of `rdfs:Class`. The resource known as `rdfs:Class` is itself a resource of `rdf:type rdfs:Class`. Individual classes (for example, 'Dog') will always have an `rdf:type` property whose value is `rdfs:Class` (or some subclass of `rdfs:Class` ...). A resource may be an instance of more than one class.

Siehe auch Definition 6, Abschnitt 3.2.

$$\forall i, c : statement(i, \text{'rdf:type'}, c) \Rightarrow instanceOf(i, c) \tag{4.10}$$

Zusätzliche instanceOf-Beziehungen können aus den weiter unten eingeführten Regeln abgeleitet werden, welche die Semantik des Properties subClassOf erfassen.

In der RDFS-Definition für das Property type wird desweiteren gefordert, daß als Wert eine Resource angegeben werden muß, die in instanceOf-Beziehung zum Konzept Class stehen muß. Dieses Constraint wird durch das weiter unten angegebene Fakt 4.64 erfaßt. Weitere Fakten, die Constraints erfassen, werden in Abschnitt 4.3.5 eingeführt. Constraint-Verletzungen können durch die Anwendung der für die Properties domain und range weiter unten eingeführten Regeln gefunden werden.

Class

RDFS-Definition: This corresponds to the generic concept of a Type or Category, similar to the notion of a Class in object-oriented programming languages such as Java. When a schema defines a new class, the resource representing that class must have an `rdf:type` property whose value is the resource `rdfs:Class`. RDF classes can be defined to represent almost anything, such as Web pages, people, document types, databases or abstract concepts.

Der Satz *"...similar to the notion of a Class in object-oriented programming languages such as Java"* ist irreführend, da Vererbung von Eigenschaften im RDF-Modell nicht explizit vorgesehen ist. Das RDF-Modell verbietet jedoch nicht (monotone) Vererbung durch zusätzliche Constraint-Regeln auszudrücken.

RDFS-Definition: The resource known as `rdfs:Class` is itself a resource of `rdf:type` `rdfs:Class`.

Das Konzept Class ist eine Instanz von sich selbst. Eine Regel ist für diese Semantik nicht notwendig. Das weiter unten aufgeführte Fakt 4.29 drückt diese Beziehung aus. In Abschnitt 4.3.4 und 4.3.5 werden die Aussagen zusammengefaßt, die als Fakten in die Wissensbasis eingetragen werden.

subClassOf

RDFS-Definition: This property specifies a subset/superset relation between classes. The `rdfs:subClassOf` property is transitive. If class `A` is a subclass of some broader class `B`, and `B` is a subclass of `C`, then `A` is also implicitly a subclass of `C`. Consequently, resources that are instances of class `A` will also be instances of `C`, since `A` is a sub-set of both `B` and `C`.

$$\begin{aligned} \forall c1, c2 : statement(c1, \text{'rdfs:subClassOf'}, c2) \\ \Rightarrow subClassOf(c1, c2) \end{aligned} \tag{4.11}$$

Regel 4.11 führt das Wissensprädikat subClassOf ein.

$$\begin{aligned} \forall c1, c2, c3 : subClassOf(c1, c2) \wedge subClassOf(c2, c3) \\ \Rightarrow subClassOf(c1, c3) \end{aligned} \tag{4.12}$$

Regel 4.12 erfaßt die transitive Beziehung von subClassOf-Hierachien.

$$\begin{aligned} \forall i, c1, c2 : instanceOf(i, c1) \wedge subClassOf(c1, c2) \\ \Rightarrow instanceOf(i, c2) \end{aligned} \tag{4.13}$$

Eine Instanz einer Klasse, die eine Unterklasse einer Oberklasse ist, ist auch Instanz der Oberklasse. Regel 4.13 drückt diese Transitivitätsbeziehung aus.

RDFS-Definition: A class may be a subclass of more than one class.

Für diese Aussage wird keine Regel benötigt. Die in diesem Kapitel eingeführte Regel- und Fakten-Menge stellen dies bereits sicher.

RDFS-Definition: A class can never be declared to be a subclass of itself, nor of any of its own subclasses.

Mit dieser Definition wird festgelegt, daß Zirkelbeziehungen in subClassOf-Hierarchien nicht erlaubt sind.[14] Die folgende Regel wird eingeführt, um Zirkelbeziehungen in subClassOf-Hierarchien zu finden und entsprechende Constraint-Verletzungen anzuzeigen. Generell sollen mit der vorliegenden Regel- und Fakten-Menge Constraint-Verletzungen gefunden werden. Es wird jedoch nicht festgelegt, wie diese aufgelöst werden sollen bzw. wie darauf reagiert werden soll.

$$\forall c : subClassOf(c, c) \Rightarrow violation('subClassCycle', c) \tag{4.14}$$

Weitere Constraints, die auf die vererbten Properties wirken, ergeben sich aus den domain- und range-Constraints, die weiter unten erläutert werden.

subPropertyOf

RDFS-Definition: The property `rdfs:subPropertyOf` is an instance of `rdf:Property` that is used to specify that one property is a specialization of another. A property may be a specialization of zero, one or more properties. If some property `P2` is a `subPropertyOf` another more general property `P1`, and if a resource `A` has a `P2` property with a value `B`, this implies that the resource `A` also has a `P1` property with value `B`.

$$\begin{aligned} \forall p1, p2 : statement(p1, \text{'rdfs:subPropertyOf'}, p2) \\ \Rightarrow subPropertyOf(p1, p2) \end{aligned} \tag{4.15}$$

Regel 4.15 führt das Prädikatensymbol subPropertyOf auf der Wissensrepräsentationsebene ein.

[14] Dieser Constraint verhindert jedoch, daß Gleichheit durch Zirkelbeziehungen ausgedrückt werden kann.

$$\forall p1, p2, p3 : subPropertyOf(p1, p2) \wedge subPropertyOf(p2, p3) \Rightarrow subPropertyOf(p1, p3) \tag{4.16}$$

Regel 4.16 erfaßt die transitive Beziehung zwischen Properties, die in subPropertyOf-Hierarchien angeordnet sind.

$$\forall s, p1, p2, o : statement(s, p1, o) \wedge subPropertyOf(p1, p2) \Rightarrow statement(s, p2, o) \tag{4.17}$$

Wird einer Resource ein Property mit einem bestimmten Wert zugeordnet und steht dieses Property in subPropertyOf-Beziehung zu einem weiteren Property, so wird auch das hierarchisch höher eingeordnete Property der Resource mit dem gleichen Wert zugeordnet. Regel 4.17 erfaßt diese Semantik.[15]

RDFS-Definition: A property can never be declared to be a subproperty of itself, nor of any of its own subproperties.

Ebenso wie bei subClassOf-Hierarchien sind Zirkelbeziehungen bei subPropertyOf-Hierarchien nicht erlaubt. Regel 4.18 findet Zirkelbeziehungen in subPropertyOf-Hierarchien und zeigt diese Constraintverletzung durch das Prädikatensymbol $violation('subPropertyCycle', p)$ an.

[15] Diese durch die RDF-Spezifikation festgelegte Semantik kann zu Problemen führen. (1) Wenn zwei Properties p und q mit den Werten v_p und v_q einer Resource r zugeordnet werden, wobei $v_p \neq v_q$ gelten soll und beide Properties in subPropertyOf-Beziehung zu sp stehen, welcher Wert gilt dann für v_{sp}? (2) Wenn p in subPropertyOf-Beziehung zu sp steht und sp zu ssp und beide, sp und ssp werden der Resource r zugeordnet, wobei die Werte von sp und ssp sich unterscheiden, welcher Wert gilt dann für ssp? Der vererbte oder der explizit für ssp angegebene Wert? Dies gleicht den Problemen, die sich durch mehrfache Vererbung und Überladen in Klassenhierarchien ergeben. Diese Probleme existieren für die instanceOf- und subClassOf-Beziehungen nicht, da in diesen Fällen keine neuen Statements generiert werden müssen. Die von der RDF-Spezifikation festgelegte Semantik für subPropertyOf-Beziehungen ist in dieser Hinsicht negativ zu bewerten. Eine angemessene Lösung könnte sein, Wertkonflikte in subPropertyOf-Beziehungen zu finden und in diesen Fällen keine neuen Statements zu generieren.

$$\forall p : subPropertyOf(p,p) \Rightarrow violation('subPropertyCycle', p1)^{16} \qquad (4.18)$$

Aus den Definitionen für das Property subPropertyOf ergibt sich, daß bei Verwendung von Properties, die in subPropertyOf-Beziehung stehen, neue Statements der Wissensbasis hinzugefügt werden müssen.[17]

domain

RDFS-Definition: An instance of ConstraintProperty that is used to indicate the class(es) on whose members a property can be used. A property may have zero, one, or more than one class as its domain. If there is no domain property, it may be used with any resource. If there is exactly one domain property, it may only be used on instances of that class (which is the value of the domain property). If there is more than one domain property, the constrained property can be used with instances of any of the classes (that are values of those domain properties). ... The `rdfs:domain` of `rdfs:domain` is the class `rdf:Property`. This indicates that the domain property is used on resources that are properties. The `rdfs:range` of `rdfs:domain` is the class `rdfs:Class`. This indicates that any resource that is the value of a domain property will be a class. Note: This specification does not constraint the number of `rdfs:domain` properties that a property may have. If there is no domain property, we know nothing about the classes with which the property is used. If there is more than

[16] Eine Regel zum Registrieren von Statements, für die eine subPropertyCycle-Verletzung vorliegt, kann wie folgt angegeben werden:
$\forall s,p,o \ : \ violation('subPropertyCycle', s) \wedge statement(s, \text{'rdfs:subPropertyOf'}, o) \Rightarrow violation('subPropertyCycle', s, \text{'rdfs:subPropertyOf'}, o)$.

[17] Die beiderseitige Abhängigkeit zwischen den Prädikatensymbolen subPropertyOf und statement kann zu Problemen führen, wenn die hier vorgestellten Regeln und Fakten mit der Prolog-üblichen SLD-Resolution verwendet werden sollen. In diesem Fall können die Regeln reduziert und umformuliert werden, so daß zumindest herausgefunden werden kann, ob jedes Statement, das in der Wissensbais eingefügt sein sollte, auch tatsächlich modelliert ist. Zur SLD-Resolution (s. [130] S. 23-24): *"Resolution ([150]) ist eine Ableitungsregel, die zum automatischen Theorembeweisen entwickelt wurde. Ein Sonderfall der Resolution ist die SLD-Resolution, die für Horn-Klauseln anwendbar ist. Der Terminus SLD-Resolution seht für selektive, lineare Resolution von bestimmten (engl.: definite) Klauseln ([105]). Die SLD-Resolution ist Ziel-gesteuert (rückwärtsverkettend) und basiert auf indeterministischer Tiefensuche."*

one `rdfs:domain` property, the constrained property can be used with resources that are members of any of the indicated classes. Note that unlike range this is a very weak constraint.

Im folgenden werden Regeln zum Finden von Domain-Constraint-Verletzungen eingeführt:

$$\begin{aligned} \forall p, i, c : statement(p, \text{'rdfs:domain'}, c) \wedge instanceOf(i, c) \\ \Rightarrow domain(i, p) \end{aligned} \tag{4.19}$$

$$\begin{aligned} \forall p, c : statement(p, \text{'rdfs:domain'}, c) \\ \Rightarrow domainConstrainedProp(p) \end{aligned} \tag{4.20}$$

$$\begin{aligned} \forall s, p, o : statement(s, p, o) \wedge \\ domainConstrainedProp(p) \wedge \\ \neg domain(s, p) \\ \Rightarrow violation('domain', s, p, o) \end{aligned} \tag{4.21}$$

Der Constraint, daß nur Konzepte von Typ Class als Wert für ein domain-Property angegeben werden dürfen, ist durch das weiter unten eingeführte Constraint-Fakt 4.70 erfaßt. Der Constraint, daß das domain-Property nur Konzepten vom Typ Property zugeordnet werden darf, ist durch das Constraint-Fakt 4.69 erfaßt.

range

RDFS-Definition: An instance of ConstraintProperty that is used to indicate the class(es) that the values of a property must be members of. The value of a range property is always a Class. Range constraints are only applied to properties. A property can have at most one range property. It is possible for it to have no range, in which case the class of the property value is unconstrained.... The `rdfs:domain` of `rdfs:range` is the class `rdf:Property`. This indicates that the range property applies to resources that are themselves properties. The `rdfs:range` of `rdfs:range` is the class `rdfs:Class`. This indicates that any resource that is the value of a range property will be a class.

Regeln zur Feststellung, ob höchstens ein range-Constraint existiert:

$$\forall s, c : statement(p, \text{'rdfs:range'}, c) \Rightarrow isRange(c, p) \tag{4.22}$$

$$\begin{aligned} \forall p, c1, c2 : isRange(c1, p) \wedge isRange(c2, p) \wedge (c1 \neq c2) \\ \Rightarrow violation('rangeCardinality', p)^{18} \end{aligned} \tag{4.23}$$

Regeln zur Feststellung von range-Verletzungen:

$$\forall p, c : isRange(c, p) \Rightarrow hasRange(p) \tag{4.24}$$

$$\begin{aligned} \forall s, p, o, c : statement(s, p, o) \wedge instanceOf(o, c) \wedge isRange(c, p) \\ \Rightarrow range(o, p) \end{aligned} \tag{4.25}$$

[18] Die Einschränkung, daß maximal eine Klasse als Wert für das range-Property angegeben werden darf, ist sehr umstritten. Diesbezügliche Diskussionen in der RDF-Interest-Group Mailling-Liste könnten eingesehen werden unter http://lists.w3.org/Archives/Public/www-rdf-interest/2000Sep/0107.html (Diskussionsstartpunkt). Desweiteren wird die Brauchbarkeit der domain- und range-Constraints in der Mailling-Liste diskutiert. Eine Möglichkeit zu erreichen, daß trotz der Kardinalitätsbeschränkung mehr als eine Klasse als möglicher Wert für ein Predicate angegeben werden darf, ist die Modellierung von Klassenhierarchien. Als Wert für den range eines Properties kann eine Oberklasse angegeben werden, die als Repräsentant für verschiedene Klassen fungiert. Z.B. kann die Klasse Motorfahrzeug als Oberklasse der Klassen Auto, LKW und Motorrad modelliert werden. Als range-Constraint wird einem Property die Klasse Motorfahrzeug zugewiesen. Damit sind Instanzen der Klassen Auto, LKW und Motorrad valide Werte für das Property. Eine Regel zum Registrieren von Statements, für die eine range-Kardinalitätsverletzung vorliegt, kann wie folgt angegeben werden: $\forall s, p, o : violation('rangeCardinality', s) \wedge statement(s, \text{'rdfs:range'}, o) \Rightarrow violation('rangeCardinality', s, \text{'rdfs:range'}, o)$.

$$\forall s, p, o : statement(s, p, o) \wedge hasRange(p) \wedge \neg range(o, p) \Rightarrow violation('range', s, p, o) \quad (4.26)$$

Das Prädikatensymbol $violation('range', s, p, o)$ findet ebenso wie das Prädikatensymbol $violation('domain', s, p, o)$ Constraint-Verletzungen. Außerdem werden durch die Regeln für range-Constraints Kardinalitätsverletzungen gefunden. Die Regeln wurden in der angegebenen Art modelliert, da die RDF-Spezifikationen nicht definieren, wie auf Constraint-Verletzungen reagiert werden soll. Es kann jedoch angenommen werden, daß als kleinste gemeinsame Basis bei der Interpretation der RDF-Semantik Constraint-Verletzungen gefunden werden sollten. Die Reaktion auf Constraint-Verletzungen kann in Abhängigkeit von der Applikation variieren.

Der Constraint, daß nur Konzepte von Typ Class als Wert für ein range-Property angegeben werden dürfen, ist durch das weiter unten eingeführte Constraint-Fakt 4.72 erfaßt. Der Constraint, daß das range-Property nur Konzepten vom Typ Property zugeordnet werden darf, ist durch das Constraint-Fakt 4.71 erfaßt.

4.3.3 Container-Regeln

Seq

RDFS-Definition: This corresponds to the class called 'Sequence' in the formal model for RDF presented in section 5 of the Model and Syntax specification [101]. It is an instance of `rdfs:Class` and `rdfs:subClassOf` `rdfs:Container`.

RDF-Definition: An ordered list of resources or literals. Sequence is used to declare that a property has multiple values and that the order of the values is significant. Sequence might be used, for example, to preserve an alphabetical ordering of values. Duplicate values are permitted.

RDF-Definition: To represent a collection of resources, RDF uses an additional resource that identifies the specific collection (an instance of a collection, in object modeling terminology). This resource must be declared to be an instance of one of the container object types ... The type property, ... , is used to make this declaration. The membership relation between

> this container resource and the resources that belong in the collection is defined by a set of properties defined expressly for this purpose. These membership properties are named simply '_1', '_2', '_3', etc.

Siehe auch die Definitionen 10 und 11 im Abschnitt 3.2.

$$\forall s, o : statement(s, \text{'rdf:type'}, \text{'rdf:Seq'}) \wedge statement(s, \_1, o) \Rightarrow seq(s, o) \tag{4.27}$$

$$\forall s, o1, o2 : seq(s, o1) \wedge statement(s, \_2, o2) \Rightarrow seq(s, o1, o2) \tag{4.28}$$

$$\vdots$$

Die Regeln für die Container-Konzepte Bag und Alt können analog formuliert werden.

Die RDF-Spezifikationen geben zu folgenden Fragestellungen keine Auskunft: Es ist nicht definiert, wie ein fehlendes $\_X$ in einer Sequenz (z.B., $\_1, \_3, \_4$) interpretiert werden soll. Ebenfalls ist nicht klar, wie zwei gleiche $\_X$ Predicates interpretiert werden sollen (Welcher Wert ist der relevante?). Desweiteren können Unklarheiten entstehen, wenn ein RDF-Modell modifiziert wird: wie soll mit entfernten oder eingefügten Container-Elementen verfahren werden. Eine Lösung könnte die "Neunumerierung" der $\_X$-Predicates sein. In bestimmten Fällen könnte die Neunumerierung jedoch die ursprüngliche Semantik eines Containers verändern.[19]

4.3.4 Fakten zur Erfassung vordefinierter Klassen

Die Konzept-Fakten ergeben sich aus den in den Abbildungen 4.2 und 4.4 gezeigten RDF-Graphen auf die die Regel 4.1 angewendet wird.[20]

type-Beziehungen

RDF-Schema-Klassen Die folgenden Konzepte repräsentieren die Basisklassen, welche in der RDF-Schema-Spezifikation definiert sind. Die Basisklassen

[19] Die unklare Semantik der Container-Konzepte hat zu Mißverständnissen geführt. Desweiteren ist es strittig, ob es sinnvoll ist, die Container-Konzepte in das grundlegende RDF-Modell aufzunehmen, da diese das Verständnis von RDF erschweren und die XML/RDF-Syntax komplizieren.

[20] Die Abbildungen 4.2 und 4.4 sind der RDF-Schema-Spezifikation entnommen worden.

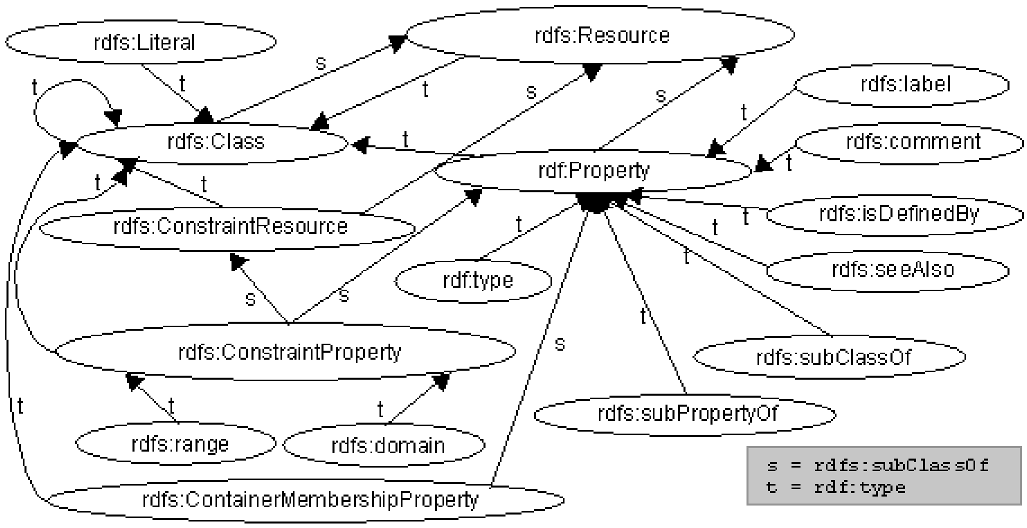

Abbildung 4.2: Klassenhierarchie der RDF-Schema-Konzepte (vgl. [27])

sind in jedem RDF-Modell (implizit) enthalten, das RDF-Schema-Konstrukte verwendet.

$$statement(\text{'rdfs:Class', 'rdf:type', 'rdfs:Class'}) \tag{4.29}$$

$$statement(\text{'rdfs:Resource', 'rdf:type', 'rdfs:Class'}) \tag{4.30}$$

$$statement(\text{'rdfs:ConstraintResource', 'rdf:type', 'rdfs:Class'}) \tag{4.31}$$

$$statement(\text{'rdfs:Literal', 'rdf:type', 'rdfs:Class'}) \tag{4.32}$$

$$statement(\text{'rdf:Property', 'rdf:type', 'rdfs:Class'}) \tag{4.33}$$

$$statement(\text{'rdfs:ConstraintProperty', 'rdf:type', 'rdfs:Class'}) \tag{4.34}$$

$$statement(\text{'rdfs:ContainerMembershipProperty', 'rdf:type', 'rdfs:Class'}) \tag{4.35}$$

RDF-Schema-Properties Jedes RDF-Modell, das RDF-Schema-Konstrukte verwendet, enthält (implizit) folgende Basis-Properties. Die Basis-Properties sind Instanzen der Property-Klasse, die zur Definition von Beziehungen zwischen Klassen und von Beziehungen zwischen Instanzen von Klassen genutzt werden können.

$$statement(\text{'rdf:type'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.36}$$
$$statement(\text{'rdfs:subPropertyOf'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.37}$$
$$statement(\text{'rdfs:subClassOf'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.38}$$
$$statement(\text{'rdfs:domain'}, \text{'rdf:type'}, \text{'rdfs:ConstraintProperty'}) \tag{4.39}$$
$$statement(\text{'rdfs:range'}, \text{'rdf:type'}, \text{'rdfs:ConstraintProperty'}) \tag{4.40}$$

Für die Property-Instanz-Beziehungen label, comment, isDefinedBy und seeAlso werden keine Fakten angegeben, da diese Beziehungen im Rahmen dieser Arbeit keine Relevanz haben. Diese Fakten können jedoch ebenfalls einfach durch Anwendung der Regel 4.1 ermittelt werden.

RDF-M+S-Klassen Die RDF-Schema-Spezifikation definiert ebenfalls die Konzepte der RDF-M+S-Spezifikation (siehe Abschnitt 6. Model and Syntax concepts und Anhang A: XML Serialization)[21]. Da die RDF-M+S-Spezifikation keine graphische Aufbereitung der Klassenhierarchie bereitstellt, wurde aus den in der RDF-Schema-Spezifikation gegebenen Definitionen Abbildung 4.3 abgeleitet. Durch Anwendung der Regel 4.1 auf den in der Abbildungen 4.3 gezeigten RDF-Graph ergeben sich folgende Fakten:

$$statement(\text{'rdf:Statement'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.41}$$
$$statement(\text{'rdfs:Container'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.42}$$
$$statement(\text{'rdf:Bag'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.43}$$
$$statement(\text{'rdf:Seq'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.44}$$
$$statement(\text{'rdf:Alt'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.45}$$

RDF-M+S-Properties

$$statement(\text{'rdfs:subject'},\text{'rdf:type'},\text{'rdf:Property'}) \tag{4.46}$$
$$statement(\text{'rdfs:predicate'},\text{'rdf:type'},\text{'rdf:Property'}) \tag{4.47}$$
$$statement(\text{'rdfs:object'},\text{'rdf:type'},\text{'rdf:Property'}) \tag{4.48}$$
$$statement(\text{'rdfs:value'},\text{'rdf:type'},\text{'rdf:Property'}) \tag{4.49}$$

[21] Aus der Tatsache, daß die RDF-Schema-Spezifikation die Konzepte der RDF-M+S-Spezifikation definiert (in der RDF-M+S-Spezifikationen wird das RDF-Modell ausschließlich anhand von Mengentheorien vorgestellt), resultiert in der Praxis Verwirrung. Die RDF-M+S-Konzepte können nicht getrennt von den RDF-Schema-Konzepten betrachtet werden, und die Spezifikationen sind ungenau bei der Verwendung der XML-Namensräumen.

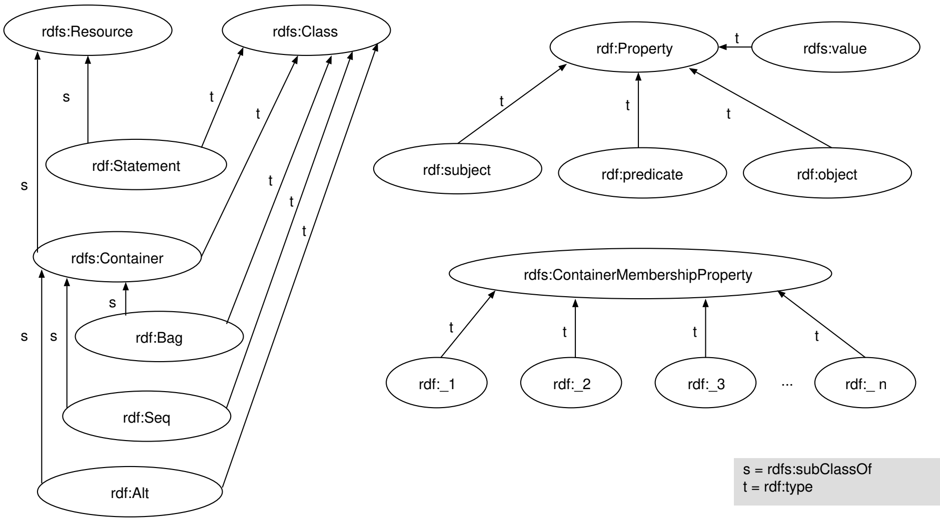

Abbildung 4.3: Klassenhierachie der RDF-M+S-Konzepte

RDF-M+S-ContainerMembership-Properties Die definierte Semantik der ContainerMembership-Properties ist bei dem Eintragen der Properties _1, _2, ..., _n in eine Wissensbasis nicht unproblematisch, da beim Einlesen eines RDF-Modells nicht von vornherein klar ist, wieviele Elemente die Container eines RDF-Modells haben werden und damit auch nicht bekannt ist, welche ContainerMembership-Fakten bei einem generellen Modell zur Verfügung gestellt werden sollen. Aus dem Grund werden im folgenden nur zwei ContainerMembership-Fakten angegeben, die stellvertretend für alle ContainerMembership-Properties stehen sollen, die in einem RDF-Modell vorkommen.[22]

[22] Als Lösungen für konkrete Implementierungen kann z.B. auf folgende Arten vorgegangen werden: (1) ContainerMemberShip-Fakten bei Bedarf dynamisch in die Faktenbasis aufnehmen oder (2) Wenn abschätzt werden kann, daß eine bestimmte Menge von Elementen in Containern nicht überschritten wird, kann von vornherein eine ausreichende Anzahl ContainerMemberShip-Fakten in die Wissensbasis eingetragen werden.

Das in den RDF-M+S-Spezifikation vorgeschlagene Design zur Darstellung von Zugehörigkeiten von Elementen zu Containern ist, wie angesprochen, problematisch. Ein diesbezüglich besseres Design wäre die Verwendung der Reifikation zur Anzeige der sequentiellen Ordnung der Elemente eines Containers. Dazu müßte das RDF-Modell abgeändert werden. Es könnte ein Property hasElement eingeführt werden, daß die Zugehörigkeit einer Resource zu einem Container anzeigt (exemplarisch: statement(s,hasElement,o)). Solche Statements könnten reifiziert werden, um dann über ein Property (z.B. hasSeqNo) die sequentielle Ordnung zu definieren (exemplarisch: statement(r´,hasSeqNo,1)) Bei diesem Vorgehen bräuchten keine n Prädikatensymbole zur Anzeige einer sequentiellen Ordnung in einer Wissensbasis eingeführt werden.

$$statement(\text{'rdf:\_1','rdf:type','rdfs:ContainerMembershipProperty'}) \tag{4.50}$$

$$\ldots$$

$$statement(\text{'rdf:\_n','rdf:type','rdfs:ContainerMembershipProperty'}) \tag{4.51}$$

subClassOf-Beziehungen

RDF-Schema-subClassOf-Beziehungen

$$statement(\text{'rdfs:Class','rdf:subClassOf','rdfs:Resource'}) \tag{4.52}$$

$$statement(\text{'rdf:Property','rdf:subClassOf','rdfs:Resource'}) \tag{4.53}$$

$$statement(\text{'rdfs:ConstraintResource','rdf:subClassOf','rdfs:Resource'}) \tag{4.54}$$

$$statement(\text{'rdfs:ConstraintProperty','rdf:subClassOf',} \\ \text{'rdfs:ConstraintResource'}) \tag{4.55}$$

$$statement(\text{'rdfs:ConstraintProperty','rdf:subClassOf','rdf:Property'}) \tag{4.56}$$

$$statement(\text{'rdfs:ConstainerMembershipProperty','rdf:subClassOf',} \\ \text{'rdf:Property'}) \tag{4.57}$$

RDF-M+S-subClassOf-Beziehungen

$$statement(\text{'rdf:Statement','rdf:subClassOf','rdfs:Resource'}) \tag{4.58}$$

$$statement(\text{'rdfs:Container','rdf:subClassOf','rdfs:Resource'}) \tag{4.59}$$

$$statement(\text{'rdf:Seq','rdf:subClassOf','rdfs:Container'}) \tag{4.60}$$

$$statement(\text{'rdf:Bag','rdf:subClassOf','rdfs:Container'}) \tag{4.61}$$

$$statement(\text{'rdf:Alt','rdf:subClassOf','rdfs:Container'}) \tag{4.62}$$

4.3.5 Fakten zur Erfassung erlaubter Typebeziehungen

$$statement(\text{'rdf:type','rdfs:domain','rdfs:Resource'}) \tag{4.63}$$

$$statement(\text{'rdf:type','rdfs:range','rdfs:Class'})^{23} \tag{4.64}$$

[23] Dieses Fakt erfaßt den Constraint, daß das Property type als Wert eine Klasse haben muß, deshalb ist in Regel 4.10 dieser Constraint nicht gesondert zu berücksichtigen.

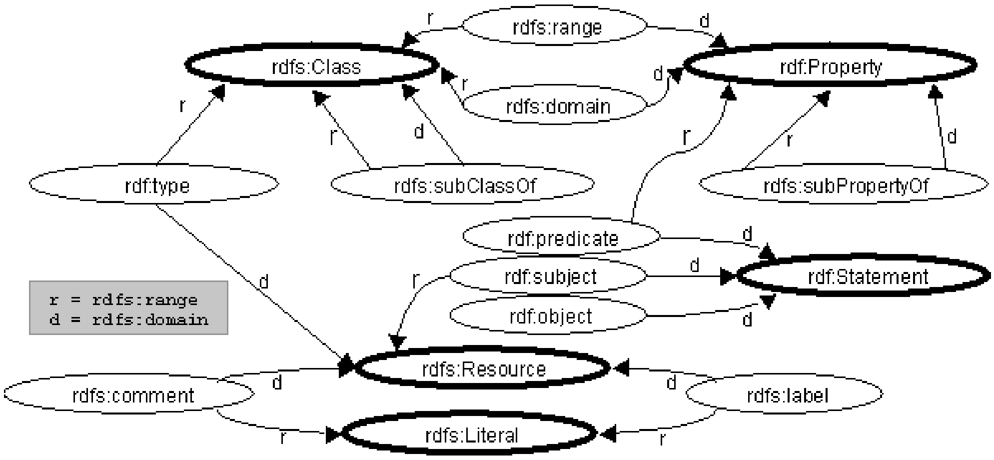

Abbildung 4.4: RDF-Schema-Constraints (vgl. [27])

$$statement(\text{'rdfs:subClassOf','rdfs:domain','rdfs:Class'}) \tag{4.65}$$
$$statement(\text{'rdfs:subClassOf','rdfs:range','rdfs:Class'}) \tag{4.66}$$

$$statement(\text{'rdfs:subPropertyOf','rdfs:domain','rdf:Property'}) \tag{4.67}$$
$$statement(\text{'rdfs:subPropertyOf','rdf:range','rdf:Property'}) \tag{4.68}$$

$$statement(\text{'rdfs:domain','rdfs:domain','rdf:Property'}) \tag{4.69}$$
$$statement(\text{'rdfs:domain','rdfs:range','rdfs:Class'}) \tag{4.70}$$

$$statement(\text{'rdfs:range','rdfs:domain','rdf:Property'}) \tag{4.71}$$
$$statement(\text{'rdfs:range','rdfs:range','rdfs:Class'}) \tag{4.72}$$

$$statement(\text{'rdf:subject','rdfs:domain','rdf:Statement'}) \tag{4.73}$$
$$statement(\text{'rdf:subject','rdfs:range','rdfs:Resource'}) \tag{4.74}$$
$$statement(\text{'rdf:predicate','rdfs:domain','rdf:Statement'}) \tag{4.75}$$
$$statement(\text{'rdf:predicate','rdfs:range','rdf:Property'}) \tag{4.76}$$

$$statement(\text{'rdf:object','rdfs:domain','rdf:Statement'})^{24} \tag{4.77}$$

Für die Domain-Range-Beziehungen von *comment*, *label* und *value* werden keine Fakten angegeben, da diese im Rahmen dieser Arbeit keine Relevanz haben. Diese Fakten können jedoch ebenfalls durch Anwendung der Regel 4.1 ermittelt werden.

4.4 Aktuelle und zukünftige Entwicklungen: RDF-Modell-Theorie

Das W3C hat ebenfalls erkannt, daß die RDF-Semantik bisher unzureichend formalisiert ist und entwickelt deshalb die *RDF Model Theory* [76, 77]. Derzeit[25] hat die RDF-Modell-Theorie (RDF-MT) den Status eines Drafts. Ziel ist die Entwicklung einer Modell-Theorie, die die Semantik von RDF und RDFS eindeutig festlegt, so daß logisches Schließen von Wissen aus RDF-Beschreibungen ermöglicht wird. Grundlage sind jedoch nicht nur die RDF-Spezifikationen, sondern auch der aktuelle Konsens, der in der RDF-Core-Working-Group bzgl. der RDF-Semantik gefunden wurde (vgl. [76]): *It reflects the current understanding of the RDF Core working group at the time of writing. In some particulars this differs from the account given in Resource Description Framework (RDF) Model and Syntax Specification ...*

In der Arbeit *Logical Interpretations of RDFS - A Compatibility Guide* [42] vergleichen Wolfram Conen und der Autor der hier vorliegenden Arbeit die durch die RDF-Modell-Theorie ausgedrückte Semantik (RDF-MT-Semantik) mit der bereits vorgestellten Interpretation der RDF-Semantik in Prädikatenlogik erster Ordnung (RDF-FOL-Semantik – siehe vorangegangene Abschnitte). Die RDF-MT-Semantik ist durch eine Mengentheorie definiert. Um die RDF-MT-Semantik mit der RDF-FOL-Semantik vergleichen zu können, wird die durch

[24] Für das Property rdf:object gilt der Constraint, daß entweder als Wert (range) eine Instanz von rdfs:Resource oder rdfs:Literal angegeben werden darf. Diese Beziehung ist in dem RDF-Graph, den Abbildung 4.4 zeigt, nicht explizit enthalten. Die RDF-Schema-Spezifikation legt jedoch fest, daß für Properties, für die kein range-Constraint angegeben wird, als Wert für range rdfs:Resource angenommen wird.

[25] In der vorliegenden Arbeit wird die RDF-Modell-Theorie in der Fassung vom 29 April 2002 zugrundegelegt.

die RDF-Modell-Theorie festgelegte Semantik ebenfalls in Prädikatenlogik erster Ordnung ausgedrückt. Dies kann, neben der Vergleichbarkeit, zur Erhöhung der Verständlichkeit der RDF-MT-Semantik beitragen und zur Entwicklung von verifizierbaren Implementatierungen führen. Der Vergleich kann Entwicklern helfen, die Konsequenzen, die durch die Änderungen der RDF-Semantik entstehen, zu verstehen, so daß diese ihre RDF-Anwendungen (wenn die Veränderung der Semantik dies nicht unmöglich macht) entsprechend anpassen können.

Im folgenden wird kurz auf die wesentlichen Unterschiede in der Semantik der beiden Interpretationen eingegangen. Für eine detaillierte und formale Betrachtung der Unterschiede siehe [42]. Die wesentlichen Änderungen betreffen die (1) Behandlung der Integritäts-Constraints (domain/range-Constraints) und (2) die Elimination der Constraints für Zirkelbezüge in subClassOf- und subPropertyOf-Beziehungen.

4.4.1 Semantik der Domain/Range-Constraints

Die MT-Interpretation der RDF-Semantik deutet die "Integritäts-Constraints" Domain und Range *nicht als Constraints* (die es ermöglichen, Integritätsverletzungen zu finden bzw. zur Validierung von RDF-Beschreibungen genutzt werden können), womit RDFS weniger als Schemasprache aufzufassen ist, sondern eher als Datentypisierungssprache (bei der die Domain/Range-"Constraints" zur Definition/Ableitung des Typs eines Subjekts oder Objekts genutzt werden).

Abbildung 4.5 verdeutlicht den Unterschied. Eine RDF-Beschreibung kann als Wissenbasis interpretiert werden, die vordefinierte Fakten und Regeln (in der Abbildung grau schattiert dargestellt) und durch RDF-Graphen beschriebenes Wissen (Eingabefakten – in der Abbildung transparent dargestellt) enthält. Vordefinierte Regeln lassen sich unterscheiden in (1) deduktive Regeln und (2) Integritäts- und Validitäts-Regeln unterscheiden. Aus den Eingabefakten kann unter Anwendung der deduktiven Regeln neues Wissen abgeleitet werden. Durch Anwendung der Integritäts und Validitäts-Regeln lassen sich Constraint-Verletzungen finden. Die durch den RDF-MT-Draft definierte RDF-Semantik enthält keine Regeln mehr, die zur Überprüfung der Integrität (bzgl. Domain- und Range-Constraints, Zirkelbeziehungen in subClassOf- und subPropertyOf-Hierarchien) einer RDF-Beschreibung herangezogen werden können. Die Integritäts-Regeln der RDF-FOL-Interpretation werden durch deduktive Regeln ersetzt.

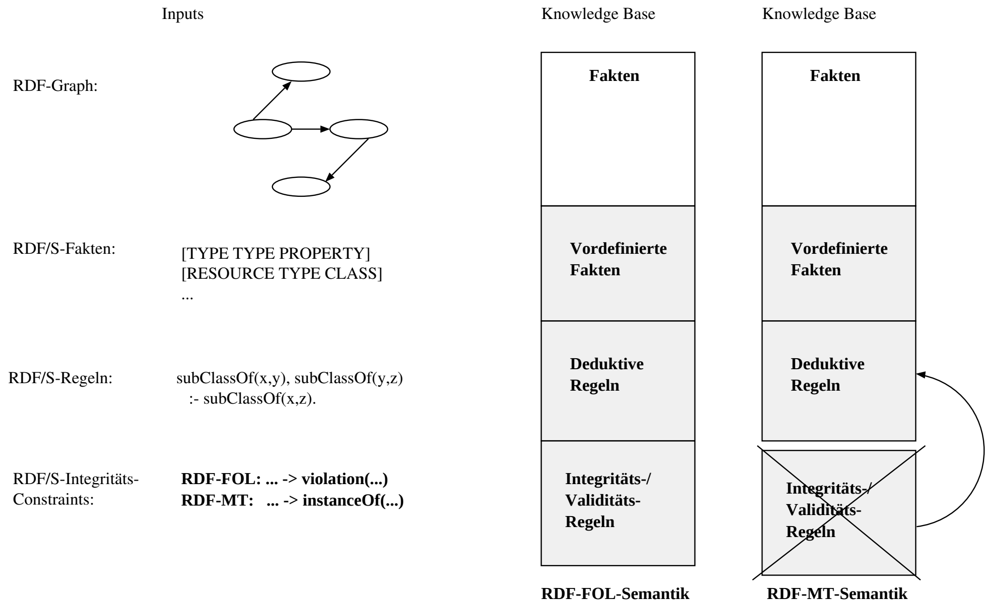

Abbildung 4.5: RDF-FOL-Semantik und RDF-MT-Semantik

Im einzelnen sind die Unterschiede wie folgt. Zunächst sind die Regeln (s.a. Abschnitt 4.3.1):

$$\forall p, i, c : statement(p, \text{'rdfs:domain'}, c) \wedge instanceOf(i, c) \Rightarrow domain(i, p) \tag{4.19}$$

$$\forall p, c : statement(p, \text{'rdfs:domain'}, c) \Rightarrow domainConstrainedProp(p) \tag{4.20}$$

$$\forall s, p, o : statement(s, p, o) \wedge domainConstrainedProp(p) \wedge \neg domain(s, p) \Rightarrow violation('domain', s, p, o) \tag{4.21}$$

zu ersetzen durch die folgenden Regeln:

$$\forall p, c : statement(p, \text{'rdfs:domain'}, c) \Rightarrow domain(c, p) \tag{4.78}$$

$$\forall s, p, o, c : statement(s, p, o) \wedge domain(c, p) \Rightarrow instanceOf(s, c) \tag{4.79}$$

Das bedeutet, daß statt der Ableitung von Domain-Constraint-Verletzungen instanceOf-Beziehungen abgeleitet werden.

Außerdem sind die Regeln (4.22)-(4.26) zur Erfassung der Range-Constraints (s.a. Abschnitt 4.3.1):

$$\begin{aligned} \forall s, c : statement(p, \text{'rdfs:range'}, c) \\ \Rightarrow isRange(c, p) \end{aligned} \tag{4.22}$$

$$\begin{aligned} \forall p, c1, c2 : isRange(c1, p) \wedge isRange(c2, p) \wedge (c1 \neq c2) \\ \Rightarrow violation('rangeCardinality', p) \end{aligned} \tag{4.23}$$

$$\begin{aligned} \forall p, c : isRange(c, p) \\ \Rightarrow hasRange(p) \end{aligned} \tag{4.24}$$

$$\begin{aligned} \forall s, p, o, c : statement(s, p, o) \wedge instanceOf(o, c) \wedge isRange(c, p) \\ \Rightarrow range(o, p) \end{aligned} \tag{4.25}$$

$$\begin{aligned} \forall s, p, o : statement(s, p, o) \wedge hasRange(p) \wedge \neg range(o, p) \\ \Rightarrow violation('range', s, p, o) \end{aligned} \tag{4.26}$$

durch folgende Regeln zu ersetzen:

$$\forall p, c : statement(p, \text{'rdfs:range'}, c) \Rightarrow range(c, p) \tag{4.80}$$

$$\forall s, p, o, c : statement(s, p, o) \wedge \neg lit(o) \wedge range(c, p) \Rightarrow instanceOf(o, c) \tag{4.81}$$

In der RDF-FOL-Interpretation gibt es explizit formulierte Regeln (vgl. (4.21), (4.23) und (4.26)), die genutzt werden können, um Integritätsverletzungen zu finden. In der RDF-MT-Interpretation dagegen sind die Domain/Range-"Constraints" deduktiv anzuwenden, d.h. der Typ einer Resource an Subjekt- oder Objekt-Position eines Statements ist abzuleiten (vgl. (4.79),(4.81)). Darüberhinaus gibt es in der RDF-MT-Interpretation keine Beschränkung der Kardinalität des Range-Properties, d.h. in der RDF-MT-Interpretation können für ein Property mehrere Klassen als Range angegeben werden.

4.4.2 Semantik der subClassOf/subPropertyOf-Constraints

Zirkel in subClassOf/subPropertyOf-Beziehungen sind gemäß der RDF-FOL-Interpretation nicht erlaubt. Die Regeln zur Erfassung dieser Constraints sind nicht mehr Bestandteil der RDF-MT-Semantik. Zu entfernen sind also:

$$\forall c : subClassOf(c, c) \Rightarrow violation('subClassCycle', c) \tag{4.14}$$

$$\forall p : subPropertyOf(p, p) \Rightarrow violation('subPropertyCycle', p) \tag{4.18}$$

4.4.3 Konsequenzen für RDF-Applikationen

Im folgenden wird auf die wesentlichen Konsequenzen eingegangen, die sich aus der RDF-MT-Semantik im Vergleich zur RDF-FOL-Semantik ergeben.

In der RDF-MT-Semantik führen Domain/Range-"Constraints"[26] zur Ableitung von Typinformation. Der Typ einer Resource kann jedoch bereits durch die Verwendung von `rdf:type`, `rdfs:subClass` und entsprechender Subproperties (z.B. `x rdfs:subProperty rdf:type` oder `x rdfs:subProperty rdfs:subClass`) ausgedrückt werden. Es ist nicht mehr möglich, die Verwendung eines bestimmten Types zu erzwingen. Durch die RDF-MT-Interpretation der Domain/Range-Constraints geht deshalb Ausdrucksmächtigkeit verloren.

Der Constraint, daß ein Property nur einen einzigen Range haben darf, ist nicht mehr Bestandteil der RDF-MT-Semantik. Für ein Property können gemäß der RDF-MT-Semantik mehrfache Domain/Range-Constraints angegeben werden, die konjunktiv (UND-verknüpft) zu interpretieren sind.

Zirkel in subClassOf- und subPropertyOf-Beziehungen sind in der RDF-MT-Interpretation erlaubt. Dies ermöglicht es, Gleichheit auszudrücken, d.h. das zwei Klassennamen die gleiche Klasse bezeichnen: $A \subseteq B, B \subseteq A \Rightarrow A = B$.

Die Änderungen bzgl. der Domain/Range-Constraints und der Zirkelbezüge in subClassOf- und subPropertyOf-Beziehungen sind *nicht kompatibel* zur ursprünglich intendierten RDF-Semantik, so daß potentiell alle[27] bereits vorhandenen RDF-Anwendungen entsprechend angepaßt werden müssen, wenn der aktuelle MT-Draft zum Standard würde (was zu erwarten ist). Eine Anpassung der gezeigten RDF-FOL-Semantik an die RDF-MT-Semantik ist jedoch möglich. Im wesentlichen erfolgt dies, wie gezeigt, durch Weglassen entsprechender Regeln bzw. durch Hinzufügen deduktiver Regeln. In diesem Abschnitt wurden nur die wesentlichen Änderungen hervorgehoben. Für eine detailliertere Betrachtung siehe [42].

[26] Der Begriff Domain/Range-Constraint wird verwendet, da in der RDF-Schema-Spezifikation [27] die Properties domain und range als Constraints bezeichnet werden. Die RDF-MT interpretiert die Properties domain und Range jedoch nicht als Constraints, d.h. domain und range werden nicht zur Integritätüberprüfung eingesetzt.

[27] Diese Aussage gilt nur insofern, als für eine RDF-Anwendung Integritäts-Constraints und Zirkelbeziehungen relevant sind.

4.5 Diskussion

Die RDF-Spezifikationen definieren grundlegende Sprachmittel (Konzepte und Constraints) zur Formulierung von RDF-Schemata und RDF-Beschreibungen. Die Semantik der Sprachmittel ist jedoch unzureichend genau spezifiziert. Sollen unabhängige Sprachinterpreter zu den gleichen logischen Schlüssen kommen, ist eine genauere Spezifikation unerläßlich. Die in Abschnitt 4.3 gezeigte Formalisierung (RDF-FOL-Semantik) erfaßt die Semantik dieser Sprachmittel in Prädikatenlogik erster Ordnung. Die Formalisierung ist jedoch nur eine von mehreren möglichen Interpretation der intendierten RDF-Semantik. Die RDF-Spezifikationen lassen Interpretationsspielraum und in der RDF-Forschungsgemeinschaft gibt es bisher keine Einigkeit über die genauen Anforderungen an RDF bzw. an die Semantik von RDF. Die in diesem Kapitel gezeigte Formalisierung wurde der RDF-Forschungsgemeinschaft bereits zur Begutachtung bereitgestellt (vgl. [40]), als wertvolle Analyse der RDF-Semantik bewertet (vgl. [13]), diskutiert (vgl. [94]), als Ausgangspunkt für weitere Arbeiten verwendet (vgl. z.B. [128]) und zur Erläuterung der RDF-Technologie eingesetzt (vgl. z.B. [148]).

Das W3C sieht ebenfalls die Notwendigkeit, die RDF-Semantik formal zu spezifizieren. Pat Hayes stellt mit der RDF-Modell-Theorie eine weitere Interpretation der RDF-Semantik zur Verfügung. Die Eindeutigkeit und Klarheit der im Abschnitt 4.3 gezeigten Formalisierung veranlaßte Tim Berners-Lee zur Artikulation von Änderungsbedarf bzgl. der RDF-Semantik (speziell zur Semantik des Range-Constraints). Entsprechende Änderungen wurden diskutiert und haben bereits Eingang in die RDF-MT-Semantik gefunden. Die RDF-MT wird durch mengenalgebraische Ausdrücke definiert. Um einen Vergleich der wesentlichen Unterschiede zwischen RDF-FOL-Semantik und RDF-MT-Semantik zu ermöglichen, zeigt Abschnitt 4.4 eine Formulierung der entsprechenden RDF-MT-Semantik in Prädikatenlogik erster Ordnung. Außerdem wird gezeigt, daß die RDF-FOL-Semantik an die RDF-MT-Semantik angepaßt werden kann. Mögliche Auswirkungen durch die Änderung der RDF-Semantik auf schon bestehende RDF-Applikationen werden ebenfalls erläutert.

XML ermöglicht syntaktischen Interoperabilität zwischen Anwendungen. Ziel von RDF ist es, semantische Interoperabilität zu ermöglichen. Dies kann jedoch nur gewährleistet werden, wenn ein RDF-Dokument für alle beteiligten Aktoren (Anwendungen/Interpreter/Menschen) die gleiche Bedeutung hat. Ist dies nicht der Fall, ist die semantische Interpretation eines RDF-Dokuments (sei es ein RDF-Schema oder eine RDF-Beschreibung) mehrdeutig. Die RDF-

Semantik sollte eindeutig definiert werden, da sonst eine babylonisches Sprachverwirrung entsteht und sich die beteiligten Aktoren nicht “verstehen”. Eine grundlegende und allgemeinverständliche bzw. allgemeingültige Semantik ist notwendig. Die Formalisierung der RDF-Semantik in Prädikatenlogik erster Ordnung, einem eindeutigen, wohluntersuchten und allgemein bekannten Formalismus, kann hierzu beitragen.

5 Implementierung der RDF-Semantik mittels Prolog

Die semantische Annotation von Daten ist notwendig, wenn komplexe Interaktionen, in die eine Vielzahl von Aktoren involviert sind, verteiltes und gemeinsames Verständnis der ausgetauschten Informationen erfordern. Semantische Annotation ermöglicht beispielsweise intelligentes Suchen statt Schlüsselwort-Matching, Query-Answering-Systeme statt Information-Retrieval [62][1] oder Definition und Austausches von Wissensbasen statt rein datenformat-basiertem Austausch von Informationen. Die Semantic-Web-Arbeitsgruppe [182] des W3C unterstreicht die Relevanz des Austausches von semantisch interpretierbarer Syntax für die weitere Entwicklung des Webs. RDF soll semantische Interoperabilität ermöglichen und gilt als die Grundlage des Semantic-Web. RDF ist als Standard für die Beschreibung der Semantik von Informationen durch Metadaten-Beschreibungen gedacht (vgl. [62]).

Im Vordergrund dieser Arbeit steht die Betrachtung der Eignung von RDF zum Web-basierten Austausch von semantisch interpretierbaren Daten. Zur Realisierung des Semantic-Web (das auf RDF basiert) müssen unabhängige und heterogene Aktoren (Benutzer, Agenten, Werkzeuge) in der Lage sein Metadaten auszutauschen und auf der Grundlage von gemeinsamen semantischen Interpretationen zu verarbeiten. Es kann jedoch in Frage gestellt werden, ob RDF hierzu geeignet ist, da die meisten Aspekte der RDF-M+S-und RDFS-Spezifikation informal beschrieben sind. Diese Feststellung wird bereits ausführlich in Kapitel 4 diskutiert. Außerdem wird dort eine Formalisierung der RDF-Semantik vorgeschlagen. Zur Formalisierung wird mathematische Logik (Prädikatenlogik erster Ordnung) eingesetzt. Ein Vorteil der Verwendung von Prädikatenlogik erster Ordnung ist, daß hierfür gut untersuchte Ausdrucksmechanismen mit allgemein anerkannter Interpretation vorliegen. Außerdem steht damit eine im-

[1] Fensel bietet in *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce* [62] einen aufschlußreichen Überblick über und Begründungen für (ontologiegetriebene) Semantik in verschiedenen organisatorischen Kontexten.

plementierungsunabhängige Formalisierung der RDF-Semantik zur Verfügung. Ausgehend hiervon können verschiedene Sprachen, wie z.B. Prolog, zur Implementierung der im Kapitel 4 gezeigten RDF-Semantik eingesetzt werden. In diesem Kapitel wird eine entsprechende Prolog-Implementierung vorgestellt.[2] Eine Sprache zur Abfrage von RDF-Modellen muß in diesem Fall nicht entwickelt werden, da durch die Implementierung der RDF-Semantik in Prolog bereits eine logikbasierte Abfragesprache zur Verfügung steht. Diese kann zur Validierung und Abfrage von RDF-Schemata und -Beschreibungen eingesetzt werden. Der im Rahmen dieser Arbeit entwickelte RDF-Schema-Explorer integriert die Prolog-Implementierung der RDF-Semantik und den durch Jan Wielemaker entwickelten RDF-Parser [175]. Zur komfortableren Validierung und Abfrage von RDF-Modellen stellt die Web-basierte Schnittstelle des RDF-Schema-Explorers[3] vordefinierte Abfragen zur Verfügung.

In den folgenden Abschnitten 5.1-5.3 wird beschrieben, wie der *RDF-Schema-Explorer* funktioniert und welche grundlegenden Prolog-Prädikate zur Verfügung stehen, mit denen eine RDF-Beschreibung befragt werden kann. Im Abschnitt 5.4 wird vorgeschlagen, wie Prolog-Code in RDF-Dokumente integriert werden kann, der zur Abfrage von RDF-Beschreibungen dient. Der Abschnitt 5.5 schließt das Kapitel mit einer Diskussion des hier vorgestellten Ansatzes ab.

5.1 RDF-Schema-Explorer

Der RDF-Schema-Explorer [38] ermöglicht es, RDF-Beschreibungen zu befragen. Eine Abfrage kann sich sowohl auf die Statements einer RDF-Beschreibung als auch auf Wissen beziehen, das mittels der Fakten und Regeln, welche die Semantik der Konzepte und Constraints von RDF/RDFS erfassen, geschlossen (inferiert) werden kann. Diesem Zweck dienen vordefinierte Prolog-Prädikate. Weitere Prolog-Prädikate können zur Validierung einer RDF-Beschreibung genutzt werden. Abbildung 5.1 verdeutlicht die Zusammenhänge: Der RDF-Parser liest die, durch ein RDF-Dokument repräsentierten, Statements ein und fügt sie der Wissensbasis hinzu. Die Wissenbasis kann der Benutzer mittels der Abfrageschnittstelle durch Eingabe/Auswahl von Prolog-Prädikaten befragen/validieren.

[2] Eine Datalog-Implementierung der RDF-Semantik wurde ebenfalls im Rahmen dieser Arbeit entwickelt (siehe Anhang 9.7).
[3] Der RDF-Schema-Explorer ist unter [38] online verfügbar.

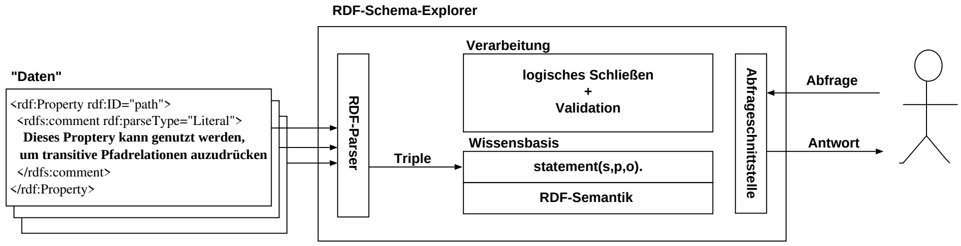

Abbildung 5.1: RDF-Schema-Explorer

Die Funktionsweise des RDF-Schema-Explorers im Detail: Der SWI-Prolog-basierte RDF-Parser[4] liest ein RDF-Dokument ein und erzeugt ein Prolog-Prädikat, das die Statements/Triple einer RDF-Beschreibung repräsentiert (ein Triple [S,P,O] wird zur Relation `statement(S,P,O)`).[5] Innerhalb der Relation `statement(S,P,O)` repräsentieren URIs Resourcen. Ein URI kann an Subjekt-, Prädikat- oder Objekt-Position vorkommen. `XML-Namensraum:Resourcen-Name` und/oder `URI#Resourcen-Name` sind syntaktische Formate, in denen ein URI angegeben wird. Damit festgestellt werden kann, daß z.B. `ns-prefix:yyy` und `URI#yyy` die gleiche Resource repräsentieren, ist eine Normalisierung (Transformation aller URIs in ein einheitliches Format) der URIs notwendig. Die im RDF-Schema-Explorer integrierte Version des SWI-Prolog-RDF-Parsers normiert URIs zu qnamen, wenn dies möglich ist.

Zur Abfrage der, durch ein RDF-Dokument repräsentierten, *Fakten/Statements* kann die Relation `statement(S,P,O)` genutzt werden. Hierzu bietet der RDF-Schema-Explorer ein Feld (siehe Abbildung 5.2), in das Prolog-Abfragen eingeben werden können; beispielsweise: `statement(S,rdf:'type',O)` zur Abfrage der type-Relation oder `setof(R,statement(R,rdf:'type',rdfs:'Class'),Rs)` zur Abfrage aller Resourcen von Typ `rdfs:Class`.

[4]Der RDF-Parser wurde von Jan Wielemaker entwickelt. Er ist Bestandteil des SWI-Prolog [175], einem frei verfügbaren und im offenen Quellcode vorliegenden Prolog.

[5]Anmerkung zum Triple: Es sollte nicht *per se* davon ausgegangen werden, daß jedes Triple einer Instanz einer binären Relation entspricht, wie dies z.B. in der Arbeit *Query + Metadata + Logic = Metalog* [114] vorausgesetzt wird. Im Kapitel 7 wird z.B. diskutiert und gezeigt, daß ein Triple, die Reifikation dieses Triples und ein negiertes Wahrheitsprädikat genutzt werden können, um Aussagen zu negieren – z.B.: bei gegebenem Triple [S,P,O] und gegebener Reifikation R, die [S,P,O] repräsentiert, kann mit dem Triple [R hasTruthValue FALSE] ausgedrückt werden, daß es bekannt ist, daß Aussage [S,P,O] nicht wahr ist. Dies wäre in der Form nicht möglich, wenn Triple durch binäre Relationen repräsentiert werden. Außerdem sind bei Verwendung einer dreistelligen Relation `statement(S,P,O)` auch Abfragen auf den Prädikatnamen P möglich, ohne daß Prädikatenlogik höherer Ordnung notwendig ist. Dies ist bei einer binären Relation, deren Relationsname der Prädikatname P ist, nicht möglich.

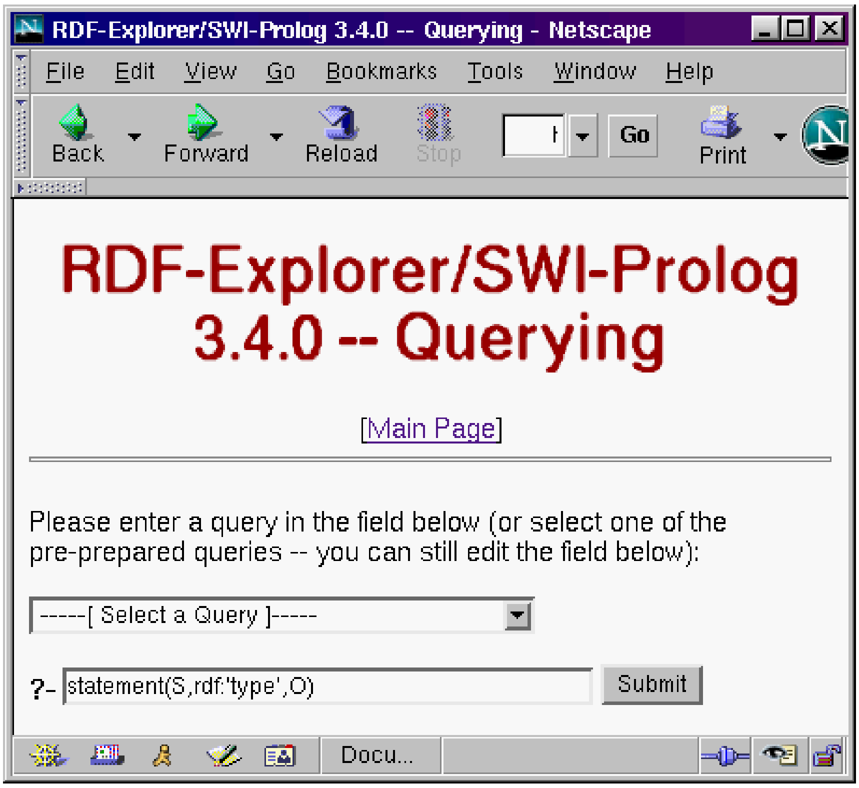

Abbildung 5.2: RDF-Schema-Explorer

Darüber hinaus implementiert der RDF-Schema-Explorer Fakten und Regeln zur Erfassung der *RDF-Semantik*[6] (siehe Abbildung 5.1), die Bestandteil jeder durch den RDF-Schema-Explorer verarbeiteten Wissensbasis sind. Die Fakten und Regeln werden durch Prolog-Prädikate repräsentiert. Abschnitt 5.2 zeigt die entsprechenden Prolog-Prädikate und gibt jeweils eine kurze Erläuterung. Die Prolog-Prädikate können zur Abfrage und Validierung (bzw. zum Finden von Constraint-Verletzungen[7]) einer RDF-Beschreibung genutzt werden.

Zur komfortablen Bedienung des RDF-Schema-Explorers bietet dieser vordefinierte Abfragen, die aus einer Liste (siehe Abbildung 5.3) ausgewählt werden können. Abschnitt 5.3 erläutert die entsprechenden Prolog-Prädikate. Bei Verwendung einer vordefinierten Abfrage ist die Kenntnis von Prolog für die Bedienung des RDF-Schema-Explorers zwar hilfreich, aber nicht notwendig. Ein Beispiel ist das Prolog-Prädikat `show_subClassOf`. Diese Auswahl führt zur Abfrage aller subClassOf-Relationen.

[6]Fakten und Regeln zur Erfassung der RDF-Semantik sind in Kapitel 4 erläutert.

[7]Hierzu dient das Prolog-Prädikat `violation`, mit dem inferiert werden kann, welche Fakten einer RDF-Beschreibung RDF-Constraints verletzten.

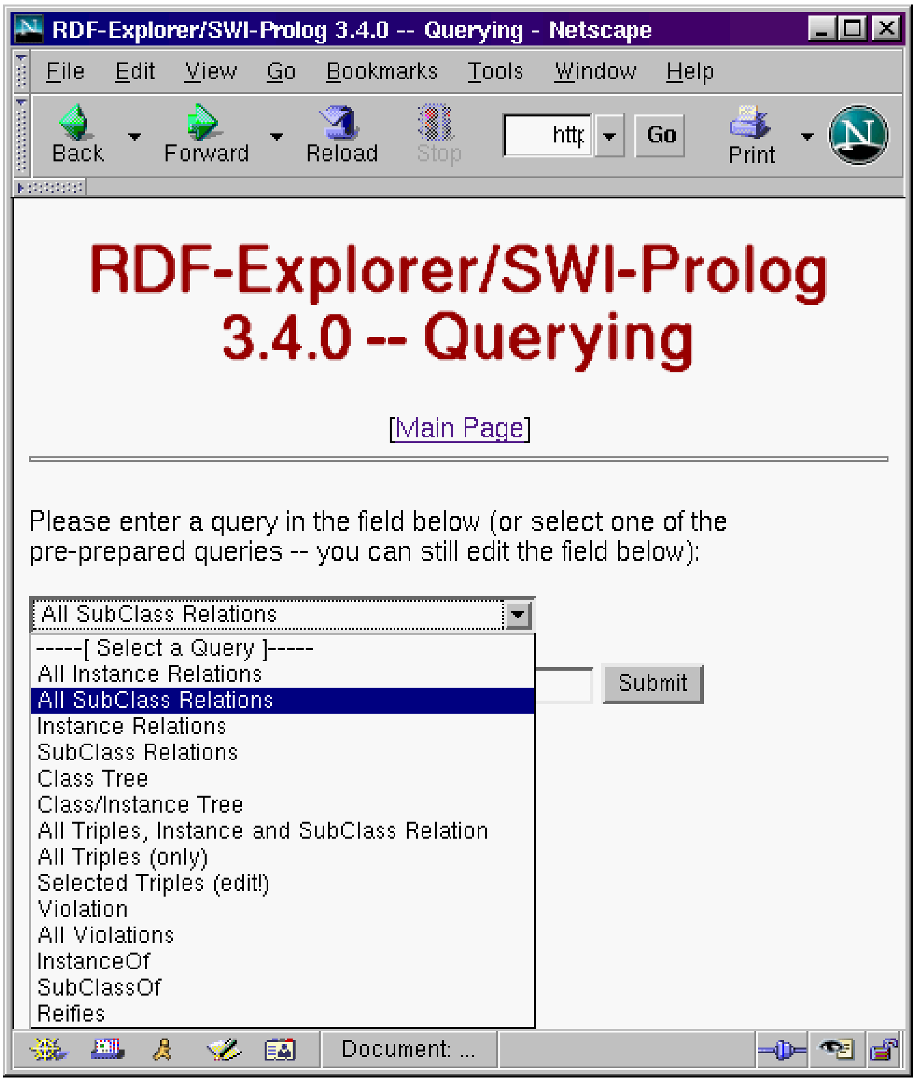

Abbildung 5.3: RDF-Schema-Explorer

Hinweise zur Syntax von Prolog: Soll in Abfragen ein Resourcen-Name verwendet werden, der mit Großbuchstaben beginnt, ist der Resourcen-Name in Anführungszeichen zu setzen - z.B. `'Resource'`. Das gilt auch, wenn Sonderzeichen im Resourcen-Name verwendet werden (dies ist z.B. bei Verwendung von Namensräumen der Fall - `rdfs:'Resource'`). Ist einem Benutzer nicht bekannt oder präsent, welche Namen bereits vergeben worden sind, kann das Prolog-Prädikat `show_instanceOf` zur Anzeige aller Instanz-Relationen verwendet werden. Der RDF-Parser des SWI-Prolog speichert jedes Literal `L` im Term `literal(L)`. Soll ein Literal abgefragt werden, ist dieser Term in der Anfrage zu benutzten. Auch hier gilt: Fängt das Literal mit Großbuchstaben an, ist es in Anführungszeichen zu setzten - z.B. `literal('Test')`.

5.2 Prolog-Prädikate zur Erfassung der RDF-Semantik

Die folgende Tabelle zeigt die Umsetzung der Formalisierung aus Kapitel 4 in Prolog. Die gezeigten Prolog-Prädikate können zur Abfrage und Validierung von RDF-Beschreibungen eingesetzt werden. Für Implementierungsdetails zu den einzelnen Prolog-Prädikaten sei auf Anhang 9.6 verwiesen.

Prädikat	Intendierte Semantik
`statement(S,P,O)`	Repräsentiert die Fakten der Wissensbasis.
`res(R)`	Repräsentiert Resourcen.
`lit(O)`	Repräsentiert Literale.
`reifies(R,S,P,O)`	R reifiziert ein (nicht notwendigerweise vorhandenes) Triple `[S P O]`.
`reifyingStatement(R)`	R erfüllt `reifies/4` für ein S,P,O.
`reifies_fact(R)`	R erfüllt `reifies/4` für eine Belegung S,P,O und das Triple `[S P O]` ist Fakt der Wissensbasis.
`subClassOf(C,D)`	Transitives Prädikat, welches die Relation erfaßt, die durch das Property `rdfs:subClassOf` ausgedrückt wird.
`instanceOf(R,C)`	Transitives Prädikat, welches die Relation erfaßt, die durch die Properties `rdf:type` und `rdfs:subClassOf` ausgedrückt wird. Jedes Literal, jede Resource und jedes Property wird explizit in die Wissensbasis eingefügt und zwar entsprechend als Instanz von `rdfs:'Literal'`, `rdfs:'Resource'` oder `rdfs:'Property'`. In einer korrekten RDF-Beschreibung sollten diese Relationen explizit durch das Property `rdf:type` beschrieben werden, da die RDF-Spezifikationen nicht klar definieren, ob diese Relationen automatisch in die Wissenbasis eingetragen werden müssen, wenn eine statement-Relation eingetragen wird.
`violation('subClassCycle',C)`	Prädikat zur Validierung. Das Prädikat ist erfüllt ("wahr"), wenn aus der Wissensbasis `subClassOf(C,C)` geschlossen werden kann. Der aktuelle RDF-MT Draft [77] erlaubt, im Gegensatz zur RDFS-Spezifikation, solche zirkulären Relationen in Klassenhierarchien. Durch zirkuläre Relationen einer Klassenhierarchie soll die Gleichheit von Klassen ausgedrückt werden können. Diese Constraint-Verletzung wird also voraussichtlich in zukünftigen RDFS-Versionen erlaubt sein. Dies gilt im übrigen, wie bereits erwähnt, auch für alle anderen Verletzungen. Eine Validierung von RDF-Beschreibungsmodellen gegen diese Constraints wird dann nicht mehr möglich sein.

Prädikat	Intendierte Semantik
`subPropertyOf(X,Y)`	Transitives Prädikat, welches die Relation erfaßt, die durch das Property `subPropertyOf` ausgedrückt wird. Der Inferenzmechanismus von Prolog erfordert eine besondere Behandlung der Regeln für die Properties `statement` und `subPropertyOf`, da diese voneinander abhängig sind (für Erläuterungen siehe Kapitel 4).
`violation('subPropertyCycle',P)`	Prädikat zur Validierung. Das Prädikat ist wahr, wenn aus der Wissenbasis `subPropertyOf(P,P)` geschlossen werden kann.
`domain_constrained_property(P)`	Es existiert mindestens ein Statement, welches einen Domain-Constraint für das Property `P` spezifiziert.
`domain(X,P)`	`X` ist eine Instanz einer der Klassen, die im Domain von `P` liegen.
`violation('domain',S,P,O)`	Prädikat zur Validierung. Es ist erfüllt, wenn ein Triple `[S P O]` in der Wissensbais ist, für `P` ein Domain-Constraint existiert und `S` nicht im Domain von `P` ist. Wenn kein Domain- bzw. Range-Constraint vorhanden ist, dann ist die Verwendung der Property bzw. die Zuweisung eines Werts durch die Property laut RDFS-Spezifikation nicht eingeschränkt. Diese Bedingung ist natürlich in dynamischen Kontexten, in denen neue Einschränkungen nach und nach bekannt werden, problematisch, denn die Bedingung wird strenger, wenn ein erster Domain-Constraint bekannt wird, während sie im Falle des Bekanntwerdens eines zweiten, dritten usw. Domain-Constraints schwächer wird. Im ersten Fall kann dies dazu führen, dass bereits erlaubte Verwendungen nun zu Constraint-Verletzungen führen (nicht-monoton), während im zweiten Fall nun bisherige Verletzungen erlaubt sein können (monoton).
`is_range(C,P)`	`C` ist (eine der) Range-Restriktion(en) für `P`.
`violation('rangeCardinality',P)`	Prädikat zur Validierung. Es ist erfüllt, wenn zwei oder mehr verschiedene Range-Restriktionen für `P` existieren.
`has_range(P)`	Für `P` existiert ein Range-Constraint.
`range(X,P)`	`X` ist Instanz einer (der) Klasse(n) für die ein Range-Constraint spezifiziert ist.
`violation('range',S,P,O)`	Prädikat zur Validierung. Es ist erfüllt, wenn für `P` ein Range-Constraint existiert, das Triple `[S P O]` in der Wissensbasis ist und O nicht im Range von `P` liegt.
`violation(T,S,P,O)`	Zusatzprädikat, das alle Constraint-Verletzungen sammelt. `T` zeigt den Typ der Verletzung und `[S P O]` ist das constraint-verletzende Triple.

5.3 Prolog-Prädikate zur komfortableren Abfrage und Validierung

Die folgende Tabelle zeigt Prolog-Prädikate, die einem Benutzer die Abfrage/Validierung von RDF-Beschreibungen erleichtern sollen.

Prädikat	Intendierte Semantik
`show_instanceOf(X)`	Zeigt Typ von X.
`show_instanceOf/0`	Zeigt alle Typ-Relationen.
`show_subClassOf(X)`	Zeigt Oberklasse von X.
`show_subClassOf/0`	Zeigt alle subClassOf-Relationen.
`show_classes/0`	Zeigt die Klassenhierarchie.
`show_tree/0`	Zeigt Klassen- und Typ-Hierarchie.
`show_statements(S,P,O)`	Zeigt Statements mit bestimmter Variablenbelegung. Statt der Variablen S, P, O können Werte angegeben werden, um ausgewählte Statements anzuzeigen. `show_statements(S,P,rdfs:'Class')` zeigt beispielsweise alle Triple mit `rdfs:Class` an der Objektposition.
`show_statements/0`	Zeigt alle Triple.
`show_all/0`	Zeigt alle subClassOf- und alle Typ-Relationen, sowie alle Triple.
`show_violations/0`	Zeigt alle Constraint-Verletzungen.
`get_rdf_schemata/0`	Liest alle von einem RDF-Dokument per Namensraum-URI referenzierten RDF-Dokumente ein, sofern diese nicht schon gelesen worden sind.[8]
`normalize_nons/0`	Vereinheitlichung der Namensdarstellung aller Ressourcen (Namensraumpräfix:Resourcen-Name).

5.4 Integration von Prolog-Code in RDF-Dokumente

RDF bietet derzeit keine Möglichkeit, Abfragen in RDF-Syntax zu definieren. Es gibt hierfür allerdings eine Reihe von Vorschlägen [112, 72, 134, 145]. Es ist jedoch noch nicht abzusehen, welcher Vorschlag sich durchsetzen wird, weshalb in diesem Abschnitt ein sehr einfacher, auf Prolog basierender Ansatz, vorgeschlagen wird: Statt einer Abfrage in RDF-Syntax wird Prolog-Syntax in RDF-Dokumente integriert. Der Prolog-Code wird als Wert eines Literals angegeben. Bei diesem Ansatz ist gewährleistet, daß ein RDF-Dokument, das

[8] Derzeit wird das Einlesen von RDF-Dokumenten aus Dateien durch das Prolog-Prädikat `get_rdf_schemata/0` unterstützt. Das Einlesen von RDF-Dokumenten, die mittels des Protokolls HTTP transferiert werden, ist jedoch konzeptuell vorgesehen.

Prolog-Code integriert, ein standardkonformes RDF-Dokument bleibt. Hierzu wird das Property `xwmf:select` eingeführt. Abbildung 5.4 zeigt ein Beispiel einer Abfrage, die bewirkt, daß eine Liste von Resourcen vom Typ `Car` zurückgeben wird.

```
<xwmf:select rdf:parseType="Literal">
  xwmf:select(listOfCars,List) :-
  setof(X, statement(X,rdf:type,vehicle:'Car'), List).
</xwmf:select>
```

Prolog-Code

Abbildung 5.4: Integration von Prolog-Code in RDF-Syntax

Der RDF-Schema-Explorer führt bei jedem Auftreten des Properties `xwmf:select`[9] ein `assert` des entsprechenden Prolog-Codes aus, so daß dieser Bestandteil der Wissenbasis wird. Die Wissensbasis enthält so:

1. die durch ein RDF-Dokument repräsentierten Statements/Fakten,
2. die Prolog-Prädikate zur Erfassung der RDF-Semantik,
3. die Prolog-Prädikate zur erleichterten Abfrage und Validierung,
4. und den über das `xwmf:select`-Property integrierten Prolog-Code.

In den Arbeiten[41, 43] wird dieser Ansatz weitergehender diskutiert. Es wird dort vorgeschlagen, diesen Ansatz zur eindeutigen Definition der Semantik von RDF-Schemata und zur Erhöhung der Ausdrucksmächtigkeit derartiger Schemata zu nutzen.

5.5 Diskussion

In diesem Kapitel wird gezeigt, wie die RDF-Semantik in Prolog implementiert werden kann. Grundlage hierfür ist die im Kapitel 4 vorgeschlagene Formalisierung der RDF-Semantik. Abschnitt 5.2 zeigt die Prolog-Prädikate zur Erfassung der RDF-Semantik. Diese Prolog-Prädikate sind Grundlage vordefinierter Abfragen (siehe Abschnitt 5.3), die der Validierung und komfortableren Abfrage von RDF-Beschreibungen dienen. Hervorzuheben ist, daß bei Verwendung von Prolog eine spezielle RDF-Abfragesprache nicht entwickelt/gelernt werden muß,

[9] Auf die gleiche Art kann auch Prolog-Code integriert werden, der nicht ausschließlich der Abfrage dient. Hierzu kann Property `isDefinedAs` genutzt werden. Zur detailierten Erläuterung siehe die Arbeiten [41, 43, 44].

da Prolog (bzw. die Prolog-Implementierung der RDF-Semantik) als Abfragesprache genutzt werden kann. Einen Prolog-Interpreter zur Abfrage einzusetzen, hat im Gegensatz zum Einsatz eines speziellen RDF-Interpreters[10] den Vorteil, daß auf eine Reihe von Implementierungen von Prolog-Interpretern zurückgegriffen werden kann, die schon lange im produktiven Einsatz sind und eine hohe Qualität haben.

Im Abschnitt 5.4 wird vorgeschlagen Prolog-Code in RDF-Dokumente zu integrieren, um es einem Autor eines RDF-Dokumentes zu ermöglichen, vordefinierte Abfragen auf das RDF-Dokument bereitzustellen. Dies soll nur ein pragmatischer Ansatz sein, der solange genutzt werden kann, bis eine standardisierte Abfragesprache für RDF vorliegt.[11]

Der RDF-Schema-Explorer integriert einen RDF-Parser und die gezeigte Prolog-Implementierung der RDF-Semantik. Er ermöglicht die Anwendung der RDF-Konzepte und -Constraints auf selbst entworfene RDF-Dokumente bzw. RDF-Schemata und unterstützt so die Entwicklung und den Test neuer RDF-Anwendungen.

[10] Interpreterimplementierungen für spezielle RDF-basierte Sprachen wie z.B. OIL[29] oder DAML[74] müssen sich dagegen erst noch bewähren.

[11] Einen verwandten Ansatz, der es ermöglicht Axiome explizit in RDF-Schema-Dokumente einzubetten, stellen Staab et al. vor [164].

6 XWMF – ein erweiterbares Web-Modellierungs-Framework

Das Semantic-Web soll es ermöglichen, Metadaten bzw. Daten derart auszutauschen, daß Maschinen/Automaten logische Schlüsse aus diesen Daten ziehen können. Dazu sind die Daten durch ein geeignetes (Wissens-)Repräsentationsformat darzustellen. Die RDF-Spezifikationen definieren für diesen Zweck eine XML-basierte Syntax (RDF-Syntax). Sollen Daten einer Web-Anwendung, zusätzlich (d.h. über die Verarbeitung durch herkömmliche Web-Clients hinaus) durch Semantik-Web-Automaten verarbeitet werden können, sind diese in RDF-Syntax zu repräsentieren. Beispielsweise kann der Preis eines Produkts einem Benutzer durch einen Web-Browser angezeigt werden und zusätzlich lassen sich diese Informationen durch einen Semantic-Web-Automaten verarbeiten, wenn diese Information in RDF-Syntax vorliegt. Werkzeuge zu Erstellung entsprechender (Semantic-)Web-Anwendungen sollten deshalb ebenfalls RDF-Syntax erzeugen können.

Durch Analyse, Entwurf, Implementierung und Wartung von Web-Anwendungen fallen eine Vielzahl von Daten und Metadaten an. Diese Daten und Metadaten können ebenfalls durch RDF-Syntax repräsentiert werden. Damit steht ein interoperables und einheitliches Daten- und Austauschformat zur Verfügung, das zur Repräsentation aller Daten und Metadaten, die im Lebenszyklus einer Web-Anwendung erzeugt werden, eingesetzt werden kann. Eine derartige RDF-Beschreibung kann von Semantic-Web-Automaten (bereits) verarbeiten, so daß im Idealfall keine weitere Aufbereitung für Semantic-Web-Anwendungen mehr notwendig ist. Die entsprechende Web-Anwendung kann automatisch aus der RDF-Beschreibung (Web-Anwendungsbeschreibung) erzeugt werden.

Im folgenden wird das eXtensible Web Modeling Framework (XWMF) vorgestellt, welches RDF auf das Web-Engineering anwendet. XWMF beinhaltet eine erweiterbare Menge von RDF-Schemata, die Vokabular zur Beschreibung von Web-Anwendungen bereitstellen. Ziel war die Entwicklung eines Beschreibungsmodells, das die in Abschnitt 1.1 aufgestellen Anforderungen erfüllt.

In XWMF werden RDF-Metadaten zur Beschreibung der Eigenschaften von und der Beziehungen zwischen Web-Resourcen genutzt. Das XWMF-Beschreibungsmodell kann genutzt werden, um den *Inhalt* (engl. content) einer Web-Anwendung zu definieren und um *Aussagen über den Inhalt* (wie z.B. Struktur, Verwaltungsinformation, Modellierungsinformation) zu definieren.

Die Daten und Metadaten einer Web-Anwendung werden durch ein einheitliches, graphbasiertes Datenmodell (einen RDF-Graph) beschrieben. Ein RDF-Graph kann, wie in Kapitel 4 erläutert, als semantisches Netzwerk aufgefaßt werden, welches durch Prädikatenlogik erster Ordnung formal beschrieben werden kann. Kapitel 4 zeigt ebenfalls, daß die RDF-Semantik durch Prädikatenlogik erster Ordnung formal beschrieben werden kann. Diese formale Grundlage wird durch XWMF genutzt, so daß eine Web-Anwendung und deren Inhalt formal durch RDF-Beschreibungen spezifiziert werden kann. Damit steht einem Benutzer bzw. Entwickler ein Formalismus zur Verfügung, der es ermöglicht die Validierung der Semantik der Daten und Metadaten einer Web-Anwendung vorzunehmen und Abfragen durchzuführen. Sowohl Daten als auch Metadaten einer Web-Anwendung können automatisch semantisch erschlossen (von Maschinen "verstanden") werden.

Als Strukturierungsprimitiv wird im XWMF-Beschreibungsmodell nicht die Web-Datei verwendet, wie dies bei einigen Web-Entwicklungswerkzeugen der Fall ist, sondern Klassen und Objekte. Das Beschreibungsmodell von XWMF ist erweiterbar, so daß ein Web-Anwendungsentwickler weiteres Vokabular hinzufügen kann, um etwa unternehmensspezifische Anforderungen oder werkzeugspezifische Informationen in einer Web-Anwendungsbeschreibung berücksichtigen zu können.

Die im Rahmen dieser Arbeit entwickelten Werkzeuge können genutzt werden, um aus einer Web-Anwendungsbeschreibung die entsprechende Web-Anwendung automatisch zu generieren. Zudem implementieren die Werkzeuge ein Abfragesystem, das Abfrage und Validierung sowohl der Daten als auch der Metadaten einer Web-Anwendung ermöglicht.

Das Kapitel gliedert sich wie folgt. Abschnitt 6.1 diskutiert verwandte Web-Engineering-Ansätze und Werkzeuge. Abschnitt 6.2 beschreibt das Zusammenspiel der an XWMF beteiligten Technologien. Abschnitt 6.3 erläutert das Web-ObjectComposition-Modell, daß Grundlage jedes XWMF-Beschreibungsmodells ist. Die XWMF-Werkzeuge werden in Abschnitt 6.4 vorgestellt. Abschnitt 6.5 zeigt die Anwendung von XWMF an einem Beispiel und Abschnitt 6.6 beendet das Kapitel mit der Überprüfung, ob der gezeigte Ansatz die in Kapitel 1.1 aufgestellten Anforderungen erfüllt.

6.1 Verwandte Arbeiten

XWMF unterstützt die konzeptuelle Strukturierung der Inhalte von Web-Anwendungen. Es gibt eine Reihe von weiteren (Web-)Design-Ansätzen dies ebenfalls unterstützen. Als Beispiele seien die *Object-Oriented Hypermedia Desing Methodology* (OOHDM) [154] und die *Relationship Management Methodology* (RMM) [92] genannt. Diese Ansätze unterstützen im wesentlichen die Entwurfsphase (des Lebenszykluses) einer Web-Anwendung. XWMF bietet darüber hinaus ein standardisiertes und erweiterbares Beschreibungs- und Austauschformat, das als Grundlage der Unterstützung des gesamten Lebenszykluses einer Web-Anwendung dient.

Das *Web Object Oriented Model* (WOOM) [37] definiert ein generatives Modell zur Beschreibung von Web-Anwendungen, deren Struktur durch Objekte, die in einem gerichteten azyklischen Graphen (engl. directed acyclic graph (DAG)) angeordnet werden, festgelegt wird. Für jedes Objekt kann eine sogenannte Transformer-Methode definiert werden, die daß Objekt in die entsprechende Web-Resource konvertiert. Das WebObjectComposition-Modell von XWMF erfaßt Struktur ebenfalls durch Objekte, die in Form eines DAGs angeordnet werden. Im Gegensatz zu WOOM definieren in XWMF-Beschreibungsmodellen deskriptive Eigenschaften (die Klassen zugeordnet werden) Informationen, die zur Konvertierung eines Objekts in die entsprechende Web-Implementierung benötigt werden. Dies vermeidet, daß Modellierungs bzw. Implementierungsinformation in den Transformer-Methoden von Objekten "versteckt" wird und damit nicht mehr Bestandteil der Web-Anwendungsbeschreibung ist.

Wie die Objekte in WOOM verkapseln Objekte in WebComposition [70] Implementierungsinformation in einer dedizierten Methode. Ein WebComposition-Anwendungsmodell wird durch XML-Syntax erfaßt und ist deshalb in einem standardisierten und programmiersprachenunabhängigen Austauschformat repräsentiert. In XWMF wird dagegen RDF zur Repräsentation von Beschreibungsmodellen eingesetzt, womit zusätzlich logikbasierte Abfragen von Daten und Metadaten einer Web-Anwendung möglich werden.

Es existieren eine Reihe kommerzieller Werkzeuge, z.B. zum Web-Authoring, zur Web-Entwicklung und für das Web-Site-Management. Viele dieser Werkzeuge sind jedoch in sich geschlossen und daher schwer in andere Werkzeuge, die zur Erstellung/Steuerung/Verarbeitung von Web-Anwendungen eingesetzt werden, zu integrieren. XWMF, das im folgenden Abschnitt vorgestellt wird, benutzt ein standardisiertes Austauschformat, das die Integration von generischen Web-Engineering-Werkzeugen unterstützen kann.

6.2 XWMF-Architektur

XWMF (siehe Abbildung 6.1) besteht aus einer erweiterbaren Menge von RDF-Schemata, die Vokabular zur Beschreibung einer Web-Anwendung definieren. RDF wird als technologische Grundlage verwendet, die Syntax und Vokabular zur Erstellung von Schemata und Beschreibungen definiert.

Die Web-Engineering-Schemata (welche in der Mitte der Abbildung 6.1 dargestellt werden) sind RDF-Schemata, die Vokabular definieren, das zum Modellieren einer Web-Anwendung eingesetzt werden kann.

Das WebObjectComposition-Schema (WOCS) definiert Vokabular, durch das die Struktur und der Inhalt einer Web-Anwendung durch Objekte beschrieben werden können. Eine derartige Beschreibung ist Kern jeder Web-Anwendungsbeschreibung. Das WOCS-Vokabular und das Strukturierungsmodell werden detailliert im folgenden Abschnitt 6.3 erläutert.

XWMF ist erweiterbar, da weitere Web-Engineering-Schemata hinzugefügt werden können, beispielsweise Schemata, die Vokabular für das Content-Management, die Zugriffskontrolle und/oder die Navigationssteuerung bereitstellen.

Das durch die Web-Engineering-Schemata und RDF definierte Vokabular wird zur Erstellung von Web-Anwendungsschemata verwendet (siehe linken mittleren Bereich der Abbildung 6.1). Ein Web-Anwendungschema beschreibt anwendungsspezifische Konzepte/Klassen zur Modellierung einer Web-Anwendung. Beispielsweise können Klassen zur abstrakten Kategorisierung/Strukturierung des Inhaltes einer Web-Anwendung, wie z.B. Produktkassen, definiert werden. Diese Modellierunginformation kann für mehrere Instanzen (Web-Anwendungen) eines bestimmten Anwendungsbereichs (z.B. Web-basierte Produktdatenbanken) wiederverwendet werden.

Das durch RDF, die Web-Engineering-Schemata und die Web-Anwendungsschemata definierte Vokabular dient der Erstellung von Web-Anwedungsbeschreibungen. Eine Web-Anwendungsbeschreibung definiert die Objekte einer Web-Anwendung, die Beziehungen von Konzepten/Objekten und sonstige Metainformationen der Konzepte/Objekte. Eine Web-Anwendungsbeschreibung ist eine Menge von RDF-Beschreibungen, aus der eine Web-Anwendung automatisch generiert werden kann. Die RDF-Beschreibungen können von Semantic-Web-Automaten verarbeitet werden, so daß neben der Web-Anwendung eine Semantic-Web-Anwendung bereitgestellt werden kann (beispielsweise können die RDF-Beschreibungen von Produkte von Semantic-Web-Automaten verarbeitet werden).

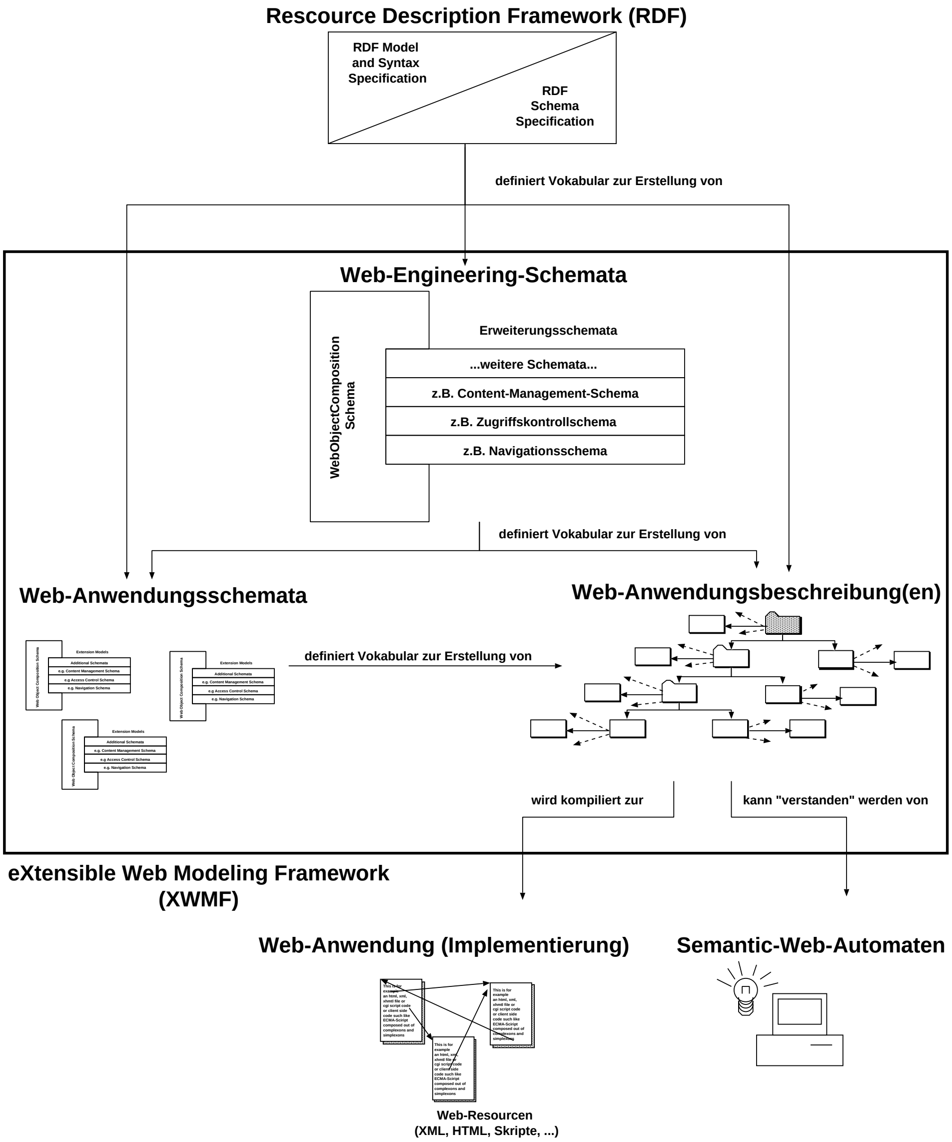

Abbildung 6.1: eXtensible Web Modeling Framework

6.3 WebObjectComposition-Modell und Methode

Das WebObjectComposition-Modell (WOCM) ist ein objektbasiertes, formales (Meta-)Datenmodell für den Entwurf der Struktur und des Inhalts einer Web-Anwendung. WOCM definiert die Konstrukte *Simplexon* und *Complexon*, die in einem gerichteten azyklischen Graph (engl. directed acyclic graph (DAG)) angeordnet werden, wobei die Simplexons die Blätter eines DAG bilden. Ein DAG definiert die Struktur einer Web-Anwendung, wobei Complexons die strukturgebenden Modellierungsprimitive sind. Durch Simplexons wird der Inhalt und die Inhaltsdarstellung definiert. *Komponenten* sind spezielle Complexons, die eine physische Resource (z.B. eine HTML-Seite) repräsentieren. Abbildung 6.2 zeigt ein Beispiel für einen DAG. Ein DAG erlaubt die Darstellung von Baumstrukturen, die den Möglichkeiten von Markup-Sprachen (z.B. HTML, XML, WML) zur syntaktischen Strukturierung der Inhalte von Web-Anwendungen entsprechen. Ein DAG kann mehrere Bäume enthalten und ein Ast/Blatt kann parallel mehreren Bäumen zugeordnet sein. Im folgenden werden die Modellierungskonstrukte, die durch WOCM definiert werden, im Detail erläutert.

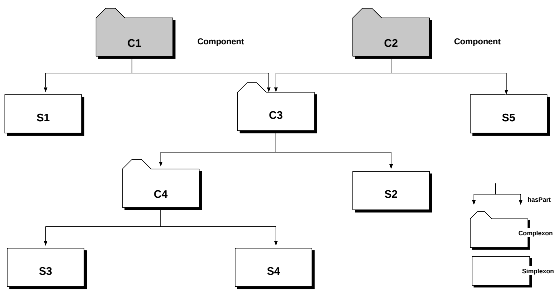

Abbildung 6.2: DAG, der zwei Komponenten strukturiert

6.3.1 Complexons

Complexons sind Container für Simplexons und Complexons. Complexons dürfen sich nicht selbst enthalten. Dies gilt auch für (indirektes) Enthaltensein, das durch zirkuläre Ausdrücke definiert ist. Complexons unterstützen die Isolierung verschiedener Modellierungsgegenstände (sog. separation of concerns) insofern, als es ermöglicht wird, die Struktur einer Web-Anwendung unabhängig von der Organisation der Dateien (die z.B. HTML-Code enthalten) zu definieren. Die Enthaltensein-Beziehung wird durch das Property *hasPart* ausgedrückt, welches im WebObjectComposition-Schema definiert ist. Abbildung 6.3 zeigt die Definition einer Complexon-Klasse `Employee`. Die Complexon-

Instanz `i1Employee` von Typ `Employee` definiert mit dem Property *hasPart* eine Enthaltensein-Beziehung für eine Sequenz von Simplexon-Instanzen. Abbildung 6.4 zeigt den entsprechenden RDF-Code. Anhang 9.9 zeigt dieses und alle nachfolgenden RDF-Code-Beispiele im Zusammenhang.

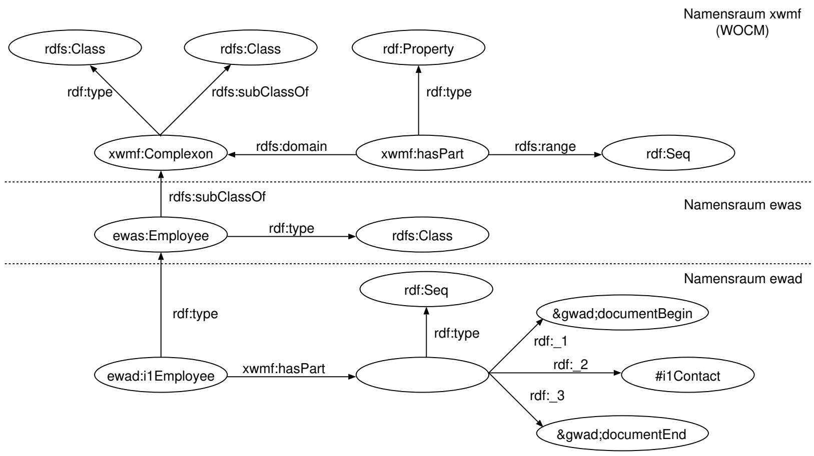

Abbildung 6.3: Definition einer Complexon-Klasse und -Instanz[1]

```
<rdfs:Class rdf:ID="Employee">
 <rdfs:subClassOf rdf:resource="&xwmf;Complexon"/>
</rdfs:Class>

<ewas:Employee rdf:ID="i1Employee">
 <xwmf:hasPart>
  <Seq>
   <li resource="&gwad;documentBegin"/>
   <li resource="#i1Contact"/>
   <li resource="&gwad;documentEnd"/>
  </Seq>
 </xwmf:hasPart>
</ewas:Employee>
```

Abbildung 6.4: Code-Beispiel[2]einer Complexon-Klasse und -Instanz

[1]Zur Verbesserung der Lesbarkeit, werden in dieser und den folgenden Abbildungen nur die für die Erläuterung relevanten Statements gezeigt. Beispielsweise fehlt im Graph das Statement `rdf:Propterty, rdf:type, rdfs:Class`, da dieses Statement bereits durch die RDF-Spezifikationen hinreichend erläutert ist.

[2]`ewas` ist Namensraumpräfix für das RDF-Schema, indem die Klasse `Employee` definiert ist. `&gwad;` ist eine Entity-Referenz für einen URI, der das RDF-Schema referenziert, indem `documentBegin` und `documentEnd` definiert ist. Zur Namenskonvention: `was` steht für *web application schema* und `wad` für *web application description*.

6.3.2 Simplexons

Die Metaklasse Simplexon wird zur Definition einer (Darstellungs-)Sicht (Realisierungscode für Objekte bestimmten Typs) genutzt. Dazu ist für eine Instanz[3] der Klasse Simplexon ein Code-Template zu definieren. In das Code-Template werden bei Generierung einer (Teil-)Web-Resource die Property-Werte eines Objekts der Simplexon-Instanz eingefügt. Die Isolierung von Modellierungsgeständen, in diesem Fall die Trennung von Inhalt und Realisierungscode, wird unterstützt, da der Inhalt eines Objektes und entsprechende Code-Templates separat definiert/modifiziert werden können. Aus der Anwendung des propertyzentrierten Ansatzes von RDF (d.h. im Gegensatz zu objektorientierten Systemen werden die Property-Definitionen von den Klassen-Definitionen getrennt) ergibt sich, daß Mehrfachverwendung von Property-Definitionen unterstützt wird. Abbildung 6.5 zeigt die Definition des Properties `name`, das zur Annotation von Instanzen der Klasse `Contact` verwendet werden kann.

```
<rdfs:Class rdf:ID="Contact">
 <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<Property ID="name">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>
```

Abbildung 6.5: Von der Klassen-Definition getrennte Property-Definition

Durch subClassOf-Beziehungen können Klassenhierarchien modelliert und Domain/Range-Constraints entlang der subClassOf-Beziehungen vererbt werden. Abbildung 6.6 zeigt ein Beispiel einer Klassenhierarchie, in der die Simplexons `HtmlContact` und `WmlContact` die Domain/Range-Constraints für das Property `name` von der Klasse `Contact` erben. Abbildung 6.7 zeigt den entsprechenden RDF-Code, durch den die subClassOf-Beziehung der Klasse `Contact` und `HtmlContact` durch das Property *subClassOf* beschrieben wird.

Für eine Simplexon-Klasse kann eine Code-Schablone definiert werden, die zur automatischen Generierung von Web-Anwendungscode genutzt wird. Das Property *template* dient zur Definition von Code-Schablonen.[4] Als Code können Markup und beliebige (XML-)Strings verwendet werden. Beispielsweise lassen sich Schablonen für Web-Seiten zur Darstellung in HTML, XML oder WML definieren.

[3] Eine Instanz der Klasse Simplexon ist selbst eine Klasse.

[4] Eine Code-Schablone ist als CDATA-Section zu definieren. Für einen RDF-Parser ist eine Code-Schablone ein Literal.

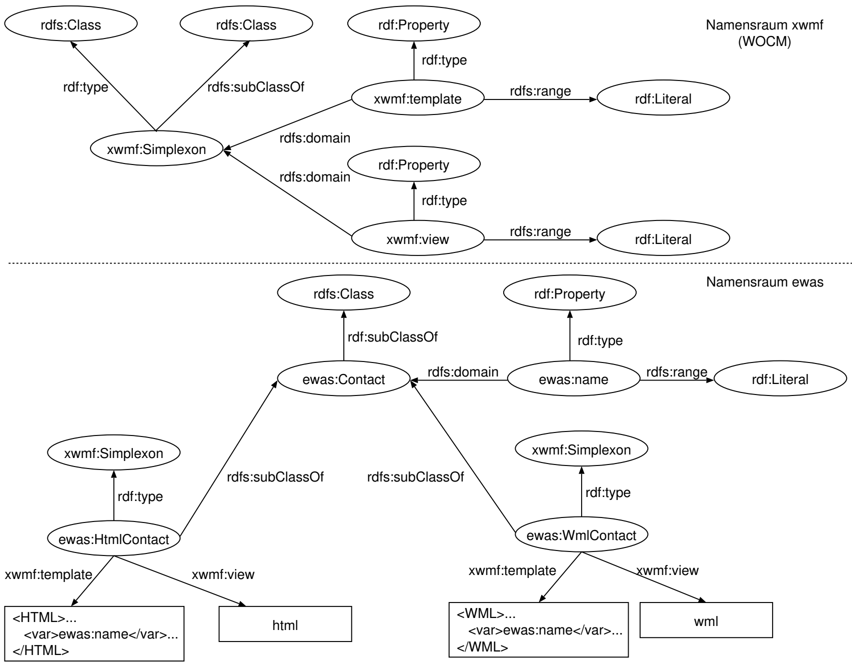

Abbildung 6.6: Vererbung der Domain/Range-Constraints

```
<xwmf:Simplexon rdf:ID="HtmlContact">
 <rdfs:subClassOf rdf:resource="#Contact"/>
 <xwmf:view>html</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
  <![CDATA[
   <CENTER>
    <TABLE bgcolor="#e8e8e8" ><TR>
     <TD valign="bottom">
      <var>ewas:name</var><br>
      Phone: <var>ewas:phone</var><br>
      Fax: <var>ewas:fax</var><br>
      <A HREF="mailto:<var>ewas:email</var>">
      <var>ewas:email</var></A><BR>
      </TD>
     </TABLE>
     </CENTER>
     ]]>
 </xwmf:template>
</xwmf:Simplexon>
```

Abbildung 6.7: Definition einer Simplexon-Klasse

Schablonen beliebiger Granularität ist erlaubt. Damit sind sowohl Ad-hoc-Designs (z.B. durch die Definition einer einzigen Schablone für eine umfangreiche Web- Seite) als auch detaillierte Designs (z.B. durch die Definition eines Complexons, das Simplexons zu einer Web-Seite zusammensetzt) möglich. Innerhalb einer Code-Schablone können durch das XML-Element *var* Variablen/Platzhalter definiert werden die (in der automatischen Code-Generierung) durch die entsprechende Werte einer Simplexon-Instanz ersetzt werden.[5] Abbildung 6.7 zeigt beispielsweise den Platzhalter `<var>ewas:name</var>`, der durch den Wert des Properties `name`, der für eine Instanz der Klasse `HtmlContact` angegeben ist, ersetzt wird. Abbildung 6.8 zeigt das Objekt `i1Contact`, das eine Instanz der Klasse `HtmlContact` ist. Abbildung 6.9 zeigt den Realisierungscode (in diesem Fall HTML), der aus den Property-Werten des Objekts `i1Contact` und der Code-Schablone der Simplexon-Klasse `HtmlContact` (siehe Abbildung 6.7) generiert wird.

```
<ewas:HtmlContact rdf:ID="i1Contact">
 <ewas:name>Reinhold Klapsing</ewas:name>
 <ewas:phone>+49 (201) 183 4078</ewas:phone>
 <ewas:fax>+49 (201) 183 4073</ewas:fax>
 <ewas:email>
  Reinhold.Klapsing@uni-essen.de
 </ewas:email>
</ewas:HtmlContact>
```

Abbildung 6.8: Definition einer Simplexon-Instanz

```
<CENTER>
 <TABLE bgcolor="#e8e8e8" ><TR>
  <TD valign="bottom">
   Reinhold Klapsing<br>
   Phone: +49 (201) 183 4078<br>
   Fax: +49 (201) 183 4073<br>
   <A HREF="mailto:Reinhold.Klapsing@uni-essen.de">
    Reinhold.Klapsing@uni-essen.de</A><BR>
  </TD>
 </TABLE>
</CENTER>
```

Abbildung 6.9: Realisierungscode für die Simplexon-Instanz aus Abbildung 6.8

Durch diesen Ansatz wird der Realisierungscode vom Inhalt einer Web-Resource getrennt. Dies unterstützt die Wiederverwendung, da nur eine Code-Schablone für potentiell viele Simplexon-Instanzen genutzt wird. Außerdem wird dadurch die Wartung erleichtert bzw. die Flexibilität erhöht, da zur Änderung von potentiell vielen Web-Resourcen nur die Code-Schablone geändert werden muß.

[5] Im Abschnitt 6.4 wird erläutert, wann die Substitution stattfindet und welche Werkzeuge hierzu dienen.

Ein Objekt kann Instanz von mehr als einer Simplexon-Klasse sein. Dies, in Verbindung mit dem Property *view* ermöglicht die Modellierung verschiedener Sichten (Anwendung verschiedener Code-Templates) auf ein Objekt[6] abhängig von dem Kontext (siehe Erläuterung zum Property *isView* in Abschnitt 6.3.3), in dem ein Objekt verwendet wird. Außerdem wird hierdurch die Wiederverwendung von Instanzen unterstützt, da nur eine Instanz für mehrere Sichten definiert werden muß.

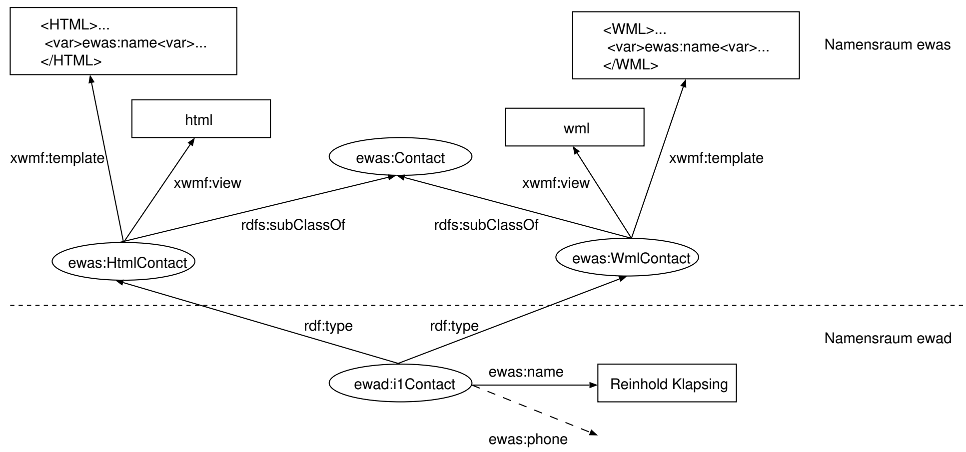

Abbildung 6.10: Mehrfachtypisiertes Objekt

```
<rdf:Description ID="i1Contact">
 <rdf:type resource="this#HtmlContact"/>
 <rdf:type resource="this#WmlContact"/>
 <ewas:name>Reinhold Klapsing</ewas:name>
 <ewas:phone>+49 (201) 183 4078</ewas:phone>
 <ewas:fax>+49 (201) 183 4073</ewas:fax>
 <ewas:email>Reinhold.Klapsing@uni-essen.de</ewas:email>
</rdf:Description>
```

Abbildung 6.11: RDF-Code eines mehrfachtypisierten Objekts

Abbildung 6.10 zeigt das Objekt `i1Contact`, das Instanz der Simplexon-Klassen `HtmlContact` und `WmlContact` ist, und Abbildung 6.11 zeigt den entsprechenden RDF-Code. Die Simplexon-Klasse `HtmlContact` definiert für das Property *view*, den Wert `html` (siehe Abbildung 6.7). Dies ermöglicht es, für jede Instanz der Klasse `HtmlContact` die Sicht `html` zu realisieren. Beispielsweise kann für das Objekt `i1Contact` die Sicht `html` realisiert werden. Außerdem ist das Objekt `i1Contact` Instanz der Simplexon-Klasse `WmlContact`, welche die Sicht `wml` definiert, so daß `i1Contact` auch im Kontext `wml` verwendet werden kann.

[6]Für jede Variable (<var>...</var>) im Code-Template, müssen die entsprechenden Property-Werte für das Objekt angegeben sein.

6.3.3 Komponenten

Eine Web-Anwendung wird aus Komponenten[7] gebildet. Eine Komponente (engl. component) ist ein spezielles Complexon. Die entsprechenden *subClassOf*-Beziehungen zeigt Abbildung 6.12. Eine Komponente repräsentiert eine physische Resource (z.B. ein HTML-Dokument) der Web-Anwendung. Eine physische Resource kann etwa ein Web-Dokument sein, das in einer Datei gespeichert ist/wird. Eine Komponente wird aus Complexons und Simplexons mittels der *hasPart*-Beziehung zusammengesetzt. Mittels des Properties *isComponent* ist der URI zu spezifizieren, mit dem eine Komponente abgerufen werden kann. Durch den Wert des Properties *isView* wird spezifiziert, in welchem Kontext die Simplexons einer Komponente zu instanziieren sind. Das Property *isView* muß für eine Komponente angegeben werden, wenn diese Simplexon-Instanzen enthält, die Instanzen von mehr als einer Klasse sind.

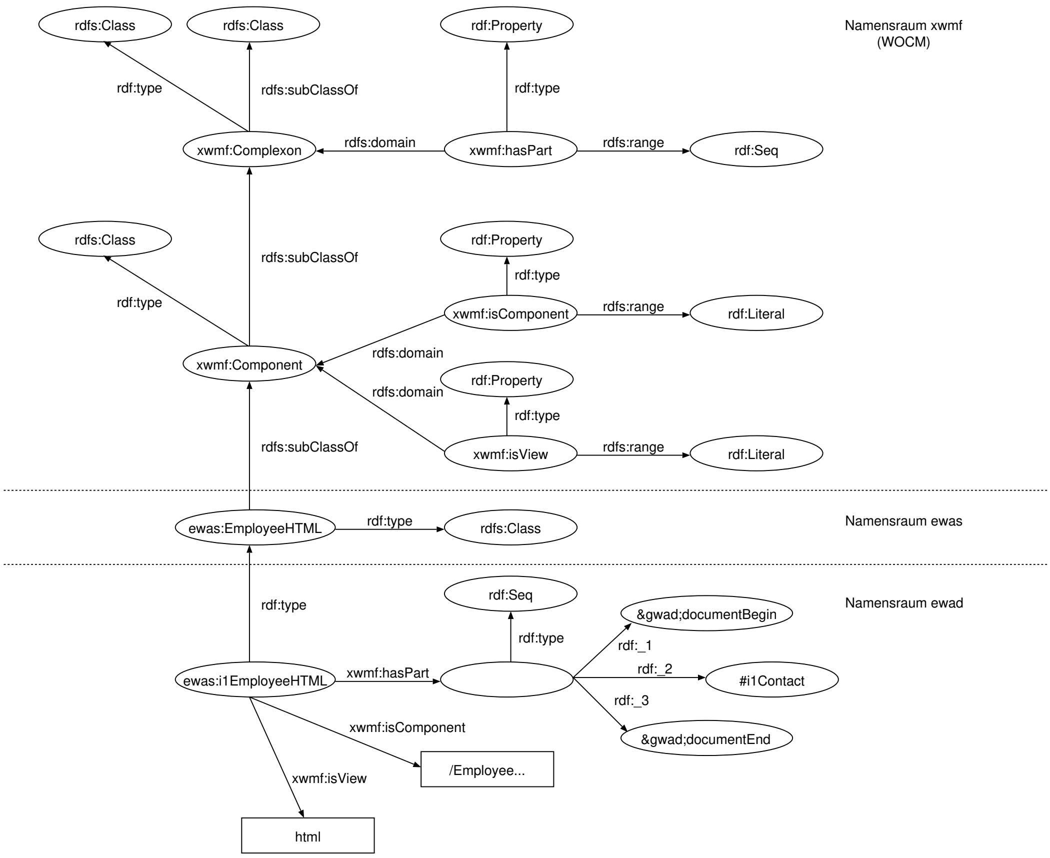

Abbildung 6.12: Komponenten sind Complexons

[7] Zum Begriff Komponente: Ein physische Einheit eines Informationssystems (z.B. ein Web-Dokument) wird als Komponente bezeichnet (vgl. z.B. [22] S. 443ff).

```
<rdfs:Class rdf:ID="EmployeeHTML">
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
</rdfs:Class>

<ewas:EmployeeHTML rdf:ID="i1EmployeeHtml">
 <xwmf:isComponent>/Employee_Klapsing.html</xwmf:isComponent>
 <xwmf:isView>html</xwmf:isView>
  <xwmf:hasPart>
  <Seq>
   <li resource="&gwad;documentBegin"/>
   <li resource="#i1Contact"/>
   <li resource="&gwad;documentEnd"/>
  </Seq>
</xwmf:hasPart>
</ewas:EmployeeHTML>
```

Abbildung 6.13: Definition einer Komponente

Abbildung 6.12 zeigt die Definition der Komponente `i1EmployeeHtml` und Abbildung 6.13 den entsprechenden RDF-Code. Nach der Erzeugung des Realisierungscodes für `i1EmployeeHtml` kann dieser mittels GET-Request (und URI `/Employee_Klapsing.html`[8]) angefragt werden. Der Wert `html` des Properties *isView* definiert den Kontext, in welchem die Simplexons, die in (transitiver) *hasPart*-Beziehung zur der Komponente stehen, zu instanziieren sind. Der Kontext `html` definiert, daß für das Objekt `i1Contact` das Code-Template der Simplexon-Klasse `HtmlContact` (und nicht das Code-Template der Simplexon-Klasse `WmlContact`) beim Erzeugungsprozeß verwendet werden muß.

6.3.4 Indexlisten

Für die Komponenten einer Web-Anwendung können konditionale Indexlisten erstellt werden. Welche Komponenten Bestandteil einer Indexliste sein sollen, ist durch die Formulierung einer Abfrage (eng. query) festzulegen. Beispielsweise kann definiert werden, daß alle Komponenten eines bestimmten Typs Bestandteil einer Indexliste sein sollen. Für jede Komponente der Indexliste wird ein Indexeintrag erzeugt. Ein Indexeintrag kann beispielsweise ein Hyperlink zu der entsprechenden Komponente sein. Sollen Instanzen einer Komponenten-Klasse in einer Indexliste verwendet werden können, ist ein Code-Template für Listeneinträge und eine Sicht zu definieren. Das Code-Template wird bei Erzeugung der Listeneinträge für diese Komponenten angewendet. Eine derartige

[8]Der URI-Teil für die Dokumentenwurzel (engl. document root) ist dem URI /Employee_Klapsing.html noch voranzustellen.

Komponenten-Klasse ist deshalb auch eine Simplexon-Instanz. Abbildung 6.14 verdeutlicht den Zusammenhang am Beispiel. Abbildung 6.15 zeigt die RDF-Beschreibung der Komponenten-Klasse `EmployeeHTML`, für die ein Code-Template definiert ist, das bei Erzeugung einer Indexliste im Kontext `html` verwendet wird. Abbildung 6.16 zeigt eine Instanz der Klasse `EmployeeHTML`. Der Hyperlink zu einer entsprechenden Instanz wird in diesem Fall aus den durch die Properties `template`, `isCommponent` und `linklabel` angegebenen Werte erzeugt. Abbildung 6.17 zeigt den Realisierungscode für einen Listeneintrag.

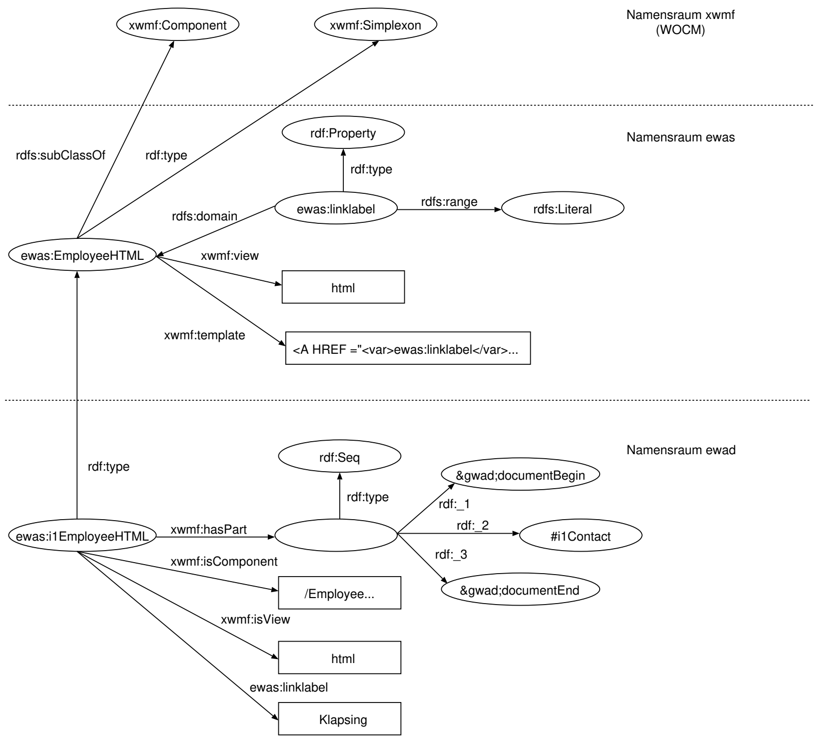

Abbildung 6.14: Definition einer Komponenten-Klasse deren Instanzen in Indexlisten verwendet werden können

```
<rdfs:Class rdf:ID="EmployeeHTML">
 <rdf:type rdf:resource="&xwmf;Simplexon"/>
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
 <xwmf:view>html</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
  <![CDATA[<A HREF="<var>xwmf:isComponent</var>">
                  <var>ewas:linklabel</var></A><P>
  ]]>
 </xwmf:template>
</rdfs:Class>

<Property ID="linklabel">
 <rdfs:domain rdf:resource="#EmployeeHTML"/>
 <rdfs:domain rdf:resource="#EmployeeWML"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>
```

Abbildung 6.15: RDF-Beschreibung einer Komponenten-Klasse, deren Instanzen in einer Indexliste verwendet werden können

```
<ewas:EmployeeHTML rdf:ID="i1EmployeeHtml">
 <xwmf:isComponent>/Employee_Klapsing.html</xwmf:isComponent>
 <xwmf:isView>html</xwmf:isView>
 <ewas:linklabel>Klapsing</ewas:linklabel>
 <xwmf:hasPart>
   <Seq>
    <li resource="&gwad;documentBegin"/>
    <li resource="#i1Contact"/>
    <li resource="&gwad;documentEnd"/>
   </Seq>
 </xwmf:hasPart>
</ewas:EmployeeHTML>
```

Abbildung 6.16: RDF-Beschreibung einer Komponente, die in einer konditionalen Liste verwendet werden kann

```
<A HREF="/Employee_Klapsing.html">Klapsing</A><P>
```

Abbildung 6.17: Realisierungscode für einen Listeneintrag

Zur Definition konditionaler Indexlisten ist eine Instanz der Klasse `Query` (siehe Abbildung 6.18) zu bilden, die Repräsentant für eine Liste von Simplexons ist. Mittels des Property `select` kann eine konditionale Abfrage (von Simplexons) in Prolog-Syntax definiert werden. Zum Einlesen der Prolog-Abfrage (Literal-Wert des Properties `select`) wird der im Kapitel 5 vorgestellte Mechanismus zur Integration von Prolog-Code auf das Property `select` angewendet (d.h. der Wert des Properties `select` wird durch einen Prolog-Interpreter ausgewertet). Abbildung 6.18 zeigt die Definition der konditionalen Indexliste `listOfEmployeeHTML` und Abbildung 6.19 die entsprechende RDF-Beschreibung.

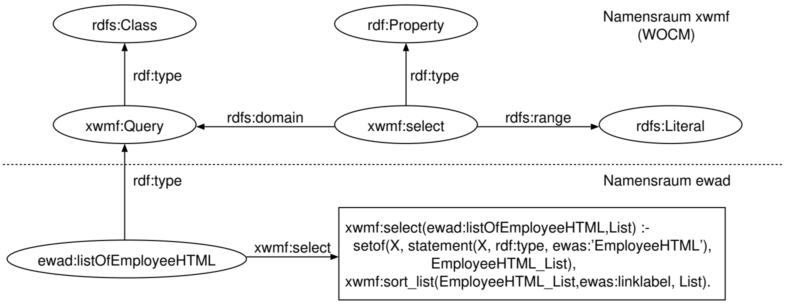

Abbildung 6.18: Definition einer konditionalen Indexliste

Die select-Abfrage der Query-Instanz `listOfEmployeeHTML` gibt eine Liste von Instanz-Namen/Simplexons vom Typ `EmployeeHTML` zurück, deren Ordnung durch die alphabetische Sortierung der Werte des Properties `linklabel` bestimmt ist. Das Property *sort_list* ist Bestandteil des WebObjectComposition-Modell (siehe Anhang 9.8). Die Semantik des Properties *sort_list* ist in Prolog definiert[9]

```
<xwmf:Query rdf:ID="listOfEmployeeHTML">
 <xwmf:select rdf:parseType="Literal">
  xwmf:select(ewad:listOfEmployeeHTML,List) :-
  setof(X, statement(X, rdf:type, ewas:'EmployeeHTML'),
                    EmployeeHTML_List),
  xwmf:sort_list(EmployeeHTML_List,ewas:linklabel, List).
 </xwmf:select>
</xwmf:Query>
```

Prolog-Code

Abbildung 6.19: Definition einer Indexliste

[9]Verwendung des Prologprädikats : `sort_list(+Input_List, +Property, -List)`.

```
<ewas:IndexOfEmployeeHTML rdf:ID="indexEmployeeHtml">
 <xwmf:isComponent>/index_of_employees.html</xwmf:isComponent>
 <xwmf:isView>html</xwmf:isView>
 <xwmf:hasPart>
  <Seq>
   <li resource="&gwad;documentBegin"/>
   <li resource="#listOfEmployeeHTML"/>
   <li resource="&gwad;documentEnd"/>
  </Seq>
 </xwmf:hasPart>
</ewas:IndexOfEmployeeHTML>
```

Abbildung 6.20: Verwendung einer Indexliste

Abbildung 6.20 zeigt die Verwendung von `listOfEmployeeHTML` als Bestandteil einer Komponente.

6.3.5 Integration von Erweiterungsschemata

Die Integration von Erweiterungsschemata in XWMF wird durch die RDF-konforme Nutzung von XML-Namensräumen [24] ermöglicht. Ein Namensraum referenziert ein RDF-Schema, das Vokabular eines Erweiterungsschemata (Erweiterungsvokabular) definiert. Durch ein Erweiterungsvokabular definierte Properties können Bestandteile (Klassen, Objekte, Properties, Complexons, Simplexons und Komponenten) einer Web-Anwendungsbeschreibung annotieren und so die Beschreibung erweitern.

Abbildung 6.21 zeigt beispielsweise die RDF-Beschreibung der Simplexon-Instanz `i1Contact` inklusive einer annotierten Erweiterungsinformation (die durch den Wert des Properties `expires` ausgedrückt wird). Das Property `expires` definiert das Verfallsdatum von Information. Diese Information ist beispielsweise für das Content-Management verwendbar. Eine Content-Managerin könnte eine Software nutzen, die eine Web-Anwendungsbeschreibung bzgl. des Properties `expires` auswertet (abfragt), um zu ermitteln, welche Information veraltet ist und erneuert/modifiziert werden muß. Das Property `expires` wird durch das RDF-Schema `http://.../.../cm.s` definiert, das durch den XML-Namensraumpräfix `cm` referenziert wird.

```
<rdf:RDF
 xmlns:rdf="http://.../.../22-rdf-syntax-ns#"
 xmlns:ewas="http://.../.../employee.was#"
 xmlns:cm="http://.../.../cm.s#">
...
<ewas:HtmlContact rdf:ID="i1Contact">
 <ewas:name>Reinhold Klapsing</ewas:name>
 <ewas:phone>+49 (201) 183 4078</ewas:phone>
 <ewas:fax>+49 (201) 183 4073</ewas:fax>
 <ewas:email>Reinhold.Klapsing@uni-essen.de</ewas:email>
 <cm:expires>31 Dec 2000</cm:expires>
</ewas:HtmlContact>
...
</rdf:RDF>
```

Abbildung 6.21: Integration von Erweiterungsvokabular

Die RDF-Syntax ermöglicht es, Statements getrennt von der Objektdefinition zu speichern (d.h. eine Aussage s über Objekt $o1$ ist in RDF-Dokument $r1$ gespeichert und die Definition des Objekts $o1$ ist in RDF-Dokument $r2$ gespeichert). Dadurch ist die Definition von Erweiterungsinformationen in separaten RDF-Dokumenten möglich, die zudem auf einem entfernten Rechner gespeichert ist. Dies kann als Grundlage für verteiltes Web-Anwendungsmanagement genutzt werden. Bezogen auf das Beispiel Content-Management bedeutet dies, daß z.B. die Verfallszeiten in einem separaten RDF-Dokument auf einem entfernten Rechner gespeichert werden können. Nur dieses RDF-Dokument bräuchte zunächst durch ein Content-Management-Werkzeug verarbeitet zu werden.

6.4 Werkzeuge

Im folgenden wird die Implementierung des WebObjectComposers erläutert, die genutzt werden kann, um aus einer Web-Anwendungsbeschreibung den entsprechenden Realisierungscode (z.B. HTML- oder WML-Dokumente) einer Web-Anwendung zu erzeugen. Als mögliche Zeitpunkte der Erzeugung des Realisierungscodes können unterschieden werden: (1) Realisierungszeitpunkt wird durch einen Administrator festgelegt oder (2) zur Laufzeit durch Abfrage einer Komponente durch einen Benutzer. Im Fall (1) veranlaßt ein Administrator die (teilweise) Realisierung einer Web-Anwendung *nach* der Erstellung bzw. *nach* jeder Modifikation der entsprechenden Web-Anwendungsbeschreibung. Im Fall (2) fordert ein Benutzer/Agent mittels GET-Request eine Komponente an, die daraufhin (zur Laufzeit) realisiert und ausgeliefert wird. Für die aktuelle Version des WebObjectComposers ist das Laufzeitmodell implementiert worden, da so *dynamisch* sämtliche Änderungen sowohl der Daten als auch der Metadaten einer Web-Anwendung berücksichtigt werden.

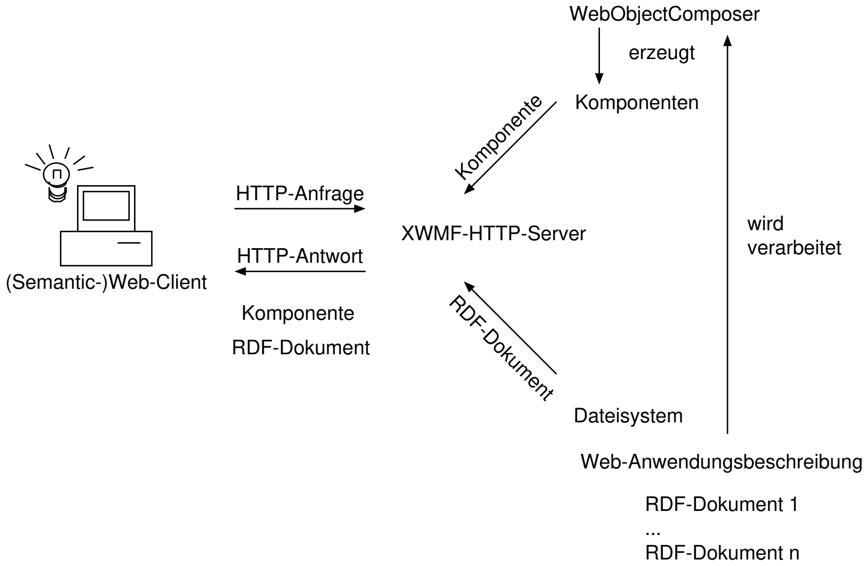

Abbildung 6.22: XWMF-HTTP-Server

Abbildung 6.22 zeigt den in einen HTTP-Server integrierten WebObject-Composer. Ein (Semantic-)Web-Client stellt eine Anfrage an den Server, der entweder eine angeforderte Komponente (z.B. ein HTML-Dokument) oder ein RDF-Dokument ausliefert. Der WebObjectComposer verarbeitet die Web-Anwendungsbeschreibung, erzeugt dynamisch (nach der Anfrage) die angeforderte Komponenten und übergibt sie dem HTTP-Server, der die Komponente ausliefert. Eine Demonstration der Implementierung und ein Beispiel einer Web-Anwendungsbeschreibung sind online verfügbar [193].

Der XWMF-HTTP-Server und der WebObjectComposer[10] sind in Prolog programmiert. Beim XWMF-HTTP-Server handelt es sich um eine (leicht) modifizierte Version des PrologHttp-Servers [143]. Die Modifikationen (siehe httpd.pl) betreffen im wesentlichen Änderungen zur Einbindung des WebObjectComposers: Erhält der HTTP-Server einen Request, prüft dieser anhand des URI-Path und der Komponenten-Werte für das Property *isComponent*, ob die Web-Anwendungsbeschreibung eine entsprechende Komponente definiert. Ist dies der Fall, erzeugt der WebObjectComposer den Realisierungscode der Komponente, den der HTTP-Server ausliefert. Ansonsten verhält sich der XWMF-HTTP-Server wie der PrologHttp-Server.

[10] Eine frühe Version des WebObjectComposers liegt in XOTcl vor. Weitere Erläuterungen siehe Abschnitt 1.2.4.

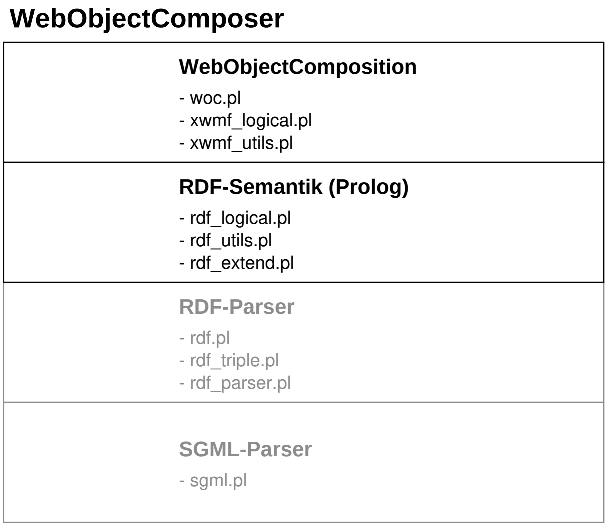

Abbildung 6.23: WebObjectComposer

Abbildung 6.23 zeigt den Aufbau des WebObjectComposers (siehe Programmdatei *woc.pl*). Der WebObjectComposer nutzt den SGML-Parser und den RDF-Parser des SWI-Prolog zum Einlesen von RDF-Dokumenten.[11] Desweiteren wird die in Prolog repräsentierte RDF-Semantik (Programmdatei *rdf_logical.pl*) eingebunden. Siehe Abschnitt 5.2 für eine Erläuterung der für diesen Zweck definierten Prolog-Prädikate. Prolog-Prädikate zur komfortableren Abfrage und Validierung von RDF-Beschreibungen definiert die Programmdatei *rdf_utils.pl*.[12] Für Erläuterungen der entsprechenden Prolog-Prädikate siehe Abschnitt 5.3. Die Datei *rdf_extend*: implementiert den im Abschnitt 5.4 beschriebenen Meachnismus zur Integration von Prolog-Code in RDF-Dokumente.

Die Programmdateien *xwmf_logical.pl* und *xwmf_utils.pl* implementieren Prolog-Prädikate zur Verarbeitung einer Web-Anwendungsbeschreibung. In Tabelle 6.1 werden die für Benutzer des WebObjectComposer wesentlichen Prolog-Prädikate erläutert.

[11] Für Erläuterungen zur Benutzung des SGML- und RDF-Parsers siehe die SWI-Prolog-Dokumentation.

[12] Die Prolog-Prädikate zur komfortableren Abfrage und Validierung von RDF-Beschreibungen sind ebenfalls Bestandteil des RDF-Schema-Explorers (siehe online [38]).

Prädikat	Intendierte Semantik
`load_rdf_file(File)`	Liest das angegebene XML-Dokument *File* ein und separiert die darin enthaltene RDF-Syntax. Die durch die RDF-Syntax ausgedrückten Triple werden erzeugt und der Wissensbasis hinzugefügt, nachdem die Namensdarstellung aller Resourcen vereinheitlicht (normalisiert) wurde. Alle via Namespace referenzierten Dokumente werden eingelesen (sofern dies nicht schon erfolgt ist).
`simplexon(X)`	Erfüllt, wenn X eine Instanz einer Simplexon-Klasse ist.
`show_simplexons/0`	Zeigt alle Simplexons.
`complexon(X)`	Erfüllt, wenn X eine Instanz einer Complexon-Klasse ist.
`show_complexons/0`	Zeigt alle Complexons.
`component(X)`	Erfüllt, wenn X eine Instanz einer Component-Klasse ist.
`show_components/0`	Zeigt alle Komponenten.
`isComponent_instance_URI(P,CN)`	Gibt für einen gegebenen URI-Path P den Komponentennamen CN zurück.
`hasPart(Y,X)`	Gibt für ein gegebenes Complexon Y den durch das `hasPart`-Property spezifizierten Sequenz-Container X zurück.
`class_for_isView(V,SI,C)`	Gibt, für eine gegebene Sicht V und eine gegebene Simplexon-Instanz SI, die Simplexon-Klasse C zurück, die die Sicht V implementiert.
`isView(CN,V)`	Gibt die Sicht V zurück, die für eine gegebene Komponente CN gefordert wird.
`show_namespaces/0`	Zeigt alle Namensraumpräfixe und dazugehörige URIs.
`show_xmwf_violations/0`	Zeigt Namensraumverletzungen. Eine Namensraumverletzung ist gegeben, wenn ein Namensraumpräfix für unterschiedliche URIs (die verschiedene Schemata referenzieren) vergeben wird.
`show_web_resource(P, RC)`	Zeigt den Realisierungscode RC für eine Komponente deren `isComponent`-Property den Wert P hat.[13]

Tabelle 6.1: Prolog-Prädikate zur Verarbeitung und Abfrage von Web-Anwendungsbeschreibungen (*woc.pl*, *xwmf_logical.pl* und *xwmf_utils.pl*)

Der folgende Abschnitt erläutert ein Beispiel einer Web-Anwendungsbeschreibung. Der Realisierungscode der Web-Anwendung wird durch den WebObject-Composer generiert und kann vom XWMF-HTTP-Server (online [193]) abgerufen werden.

[13] Wird von dem XWMF-HTTP-Server (siehe `httpd.pl`) aufgerufen.

6.5 Beispiel einer Web-Anwendungsbeschreibung

Die im folgenden gezeigte Web-Anwendungsbeschreibung integriert die in den vorangegangenen Abschnitten erläuterten Konzepte und Beispiele, so daß der Zusammenhang der XWMF-Konzepte weiter verdeutlicht wird.

Inhaltlich handelt es sich bei der Beispielanwendung um Mitarbeiterbeschreibungen und Publikationsbeschreibungen. Die Informationen über Mitarbeiter und Publikationen werden im HTML- und WML-Format zur Verfügung gestellt. Ein Benutzer kann auf die Informationen mittels konditionaler Indexlisten zugreifen.

Die Web-Anwendung wird durch eine Menge von RDF-Dokumenten beschrieben. Abbildung 6.24 zeigt eine Darstellung der RDF-Dokumente zur Beschreibung der Beispielanwendung. Die Darstellung ist in Schichten unterteilt. Ein RDF-Dokument einer unteren Schicht basiert auf Vokabular das durch RDF-Dokumente darüberliegender Schichten definiert wird. Im Anhang 9.9 sind die in Abbildung 6.24 gezeigten RDF-Dokumente enthalten. Im folgenden wird auf die wesentlichen Zusammenhänge eingegangen.

Abbildung 6.24: RDF-Dokumente der Beispielanwendung

Das RDF-Dokument `swt.wad` definiert die Einstiegsseite `/index.html` (Komponente `i1MenueHtml`), die aus den Simplexons `&gwad;documentBegin`,[14] `#i1text` und `&gwad;documentEnd` zusammengesetzt ist. Als Sicht ist `html` definiert, dies bedeutet, daß als Realisierungcode ein HTML-Dokumente erstellt wird (siehe Abbildung 6.25). Äquivalent ist die Einstiegsseite `/index.wml` (Komponente `i1MenueWml`) definiert, wobei hierfür als Realisierungscode WML-Dokumente erstellt wird.

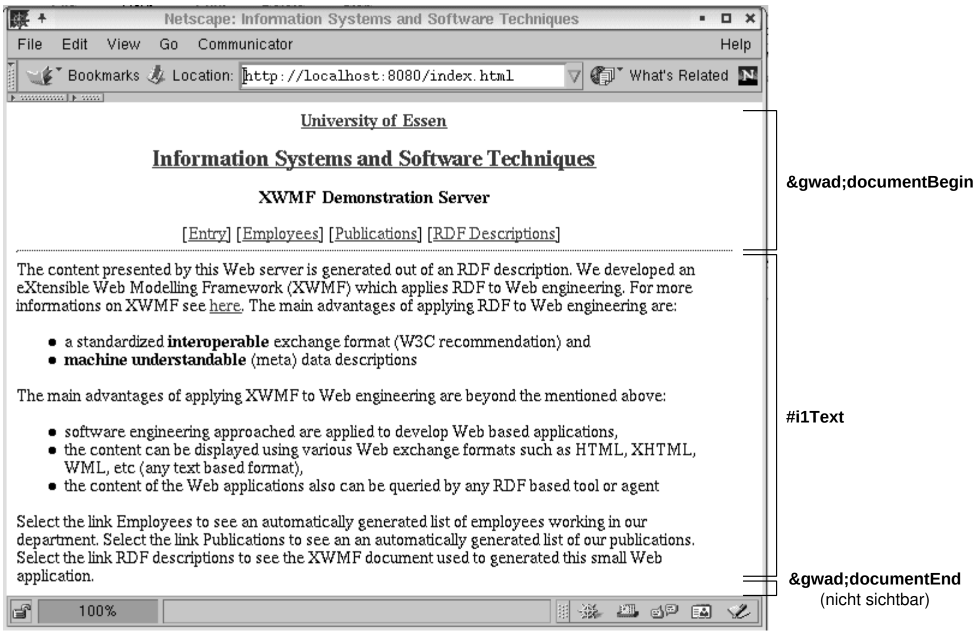

Abbildung 6.25: Realisierung der Komponente i1MenueHtml (/index.html)

Das RDF-Dokument `generic.was` definiert generische Simplexon-Klassen, die wiederverwendet werden können. Das RDF-Dokument `generic.wad` definiert z.B. das Simplexon `documentBegin` (eine Instanz der Simplexon-Klassen `HtmlDocumentBegin` und `WmlDocumentBegin`), das Bestandteil jeder Komponente der Beispiel-Web-Anwendung ist. Durch die Definition von generischen Simplexon-Klassen lassen sich komplexe Bibliotheken aufbauen. Dies kann den Entwicklungsaufwand mindern und ein einheitliches Web-Design sicherstellen.

Das RDF-Dokument `employee.wad` definiert für jeden Mitarbeiter eine Komponente vom Typ `EmployeeHTML` zur Erzeugung eines HTML-Dokuments und

[14] &gwad; ist eine Entity-Referenz, die durch den Namensraum-URI des RDF-Schemas ersetzt wird, in dem documentBegin definiert ist. In diesem Fall ist dies der URI zum Dokument generic.wad.

eine Komponente vom Type `EmployeeWML` zur Erzeugung eines WML-Dokuments. Für jeden Mitarbeiter ist eine Instanz der Klasse `Contact` definiert (mit Properties wie z.B. Telefon-Nr., Email). Abbildung 6.26 zeigt das HTML-Dokument `/Employee_Klapsing.html` (Komponente `i1EmployeeHtml`), das aus den Simplexons `&gwad;documentBegin`, `#i1Contact` und `&gwad;documentEnd` zusammengesetzt ist. `i1Contact` ist Instanz der Simplexon-Klassen `&ewas;HtmlContact` und `&ewas;WmlContact` (die durch das RDF-Dokument `employee.was` definiert sind) und über die subClassOf-Beziehung auch der Klasse `Contact` (für weitere Erläuterungen siehe auch Abschnitt 6.3.2). Das RDF-Dokument `cm.s` definiert das Property `expires`, das jeder Instanz der Klasse `Contact` zugeordnet ist und ein Verfallsdatum für den Content (in diesem Fall Kontaktinformationen) definiert. Dies ermöglicht eine Abfrage von Content, der verfallen/veraltet ist und einer Überarbeitung bedarf.

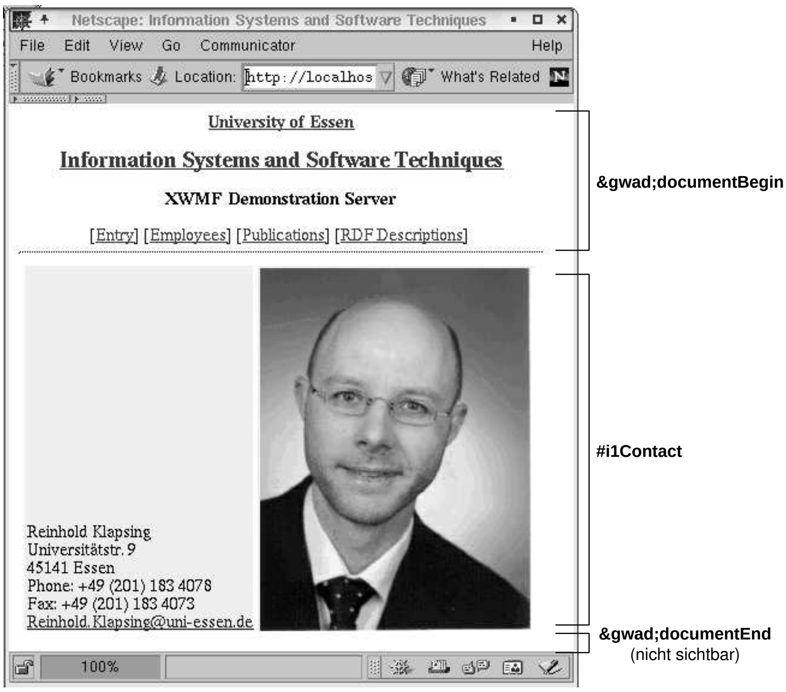

Abbildung 6.26: Realisierung der Komponente i1EmployeeHtml (/Employee_Klapsing.html)

Das RDF-Dokument `employee.wad` definiert außerdem die Komponente `indexEmployeeHtml`. Abbildung 6.27 zeigt das HTML-Dokument `/index_of_employees.html` das aus den Simplexons `&gwad;documentBegin` und `&gwad;documentEnd` sowie der Query `#listOfEmployeeHTML` zusammengesetzt ist. Die

Query `listOfEmployeeHTML` bewirkt eine Abfrage aller Komponenten vom Typ `ewas:EmployeeHTML`. Die Komponenten-Klasse `ewas:EmployeeHTML` ist gleichfalls eine Simplexon-Klasse, die ein Template definiert, das zur Erzeugung des Indexes (in diesem Fall eine Liste von Hyperlinks zu den einzelnen Komponenten der Mitarbeiter) im Realisierungscode HTML verwendet wird (für weitere Erläuterungen siehe Abschnitt 6.3.4). Eine äquivalente Query (`listOfEmployeeWml`) zur Erzeugung eines Indexes im Realisierungscode WML ist ebenfalls gegeben.

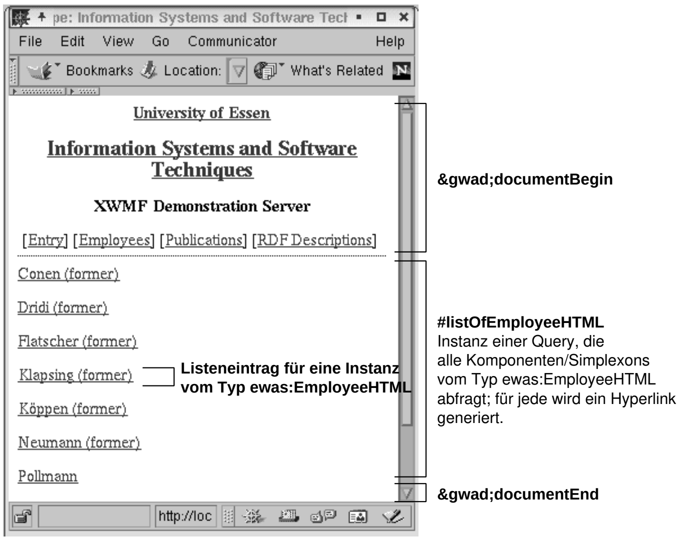

Abbildung 6.27: Realisierung der Komponente indexEmployeeHtml (/index_of_employees.html)

Die RDF-Dokumente `publication.was` und `publication.wad` definieren Beschreibungen für Publikationseinträge, die in einer Indexliste (`#listOfPublicationHtml`) angeordnet werden (siehe Abbildung 6.28). Die Indexliste besteht jedoch nicht, wie im vorangegangenen Beispiel, aus Hyperlinks (auf Komponenten), sondern aus den Publikationseinträgen selbst. Ein Publikationseintrag wird durch je eine Instanz der Simplexon-Klassen `&pwas;PublicationHTML` und `&pwas;PublicationWML` (die über die subClassOf-Beziehung auch Instanz der Klasse `Publication` ist) beschrieben. Zur Beschreibung werden Properties eingesetzt, die durch das RDF-Schema `dc.s` definiert sind. Das RDF-Schema `dc.s` enthält Vokabular, daß gemäß des *dublin core elemen set* verwendet werden soll.[15]

[15]Das *dublin core elemen set* wird oft als Beispiel für ein RDF-Vokabular angeführt. Allerdings gibt es hierfür kein offizielles RDF-Schema. Das im Anhang gezeigte RDF-Schema `dc.s` ist daher ein Vorschlag zur Umsetzung des *dublin core elemen set* in ein RDF-Schema.

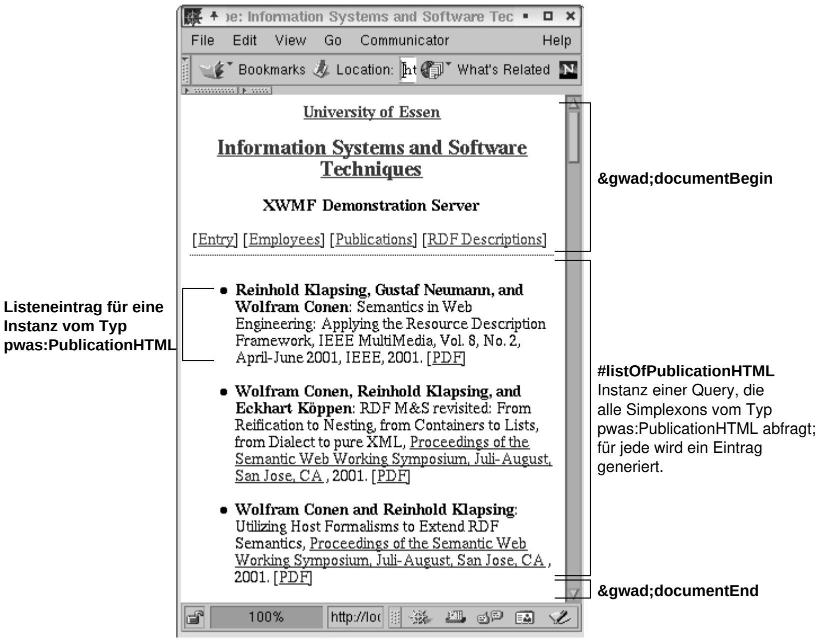

Abbildung 6.28: Realisierung der Komponente indexPublicationHtml (/index_of_publications.html)

6.6 Diskussion

In den vorangegangenen Abschnitten wurde XWMF vorgestellt, eine Anwendung der RDF-Technologie auf das Web-Engineering. Durch RDF-Dokumente werden Vokabulare definiert, die zur Beschreibung einer Web-Anwendung einsetzbar sind. Aus einer Web-Anwendungsbeschreibung wird automatisiert der Realisierungscode der Web-Anwendung generiert. Der vorgestellte WebObject-Composer ist ein Beispiel eines Werkzeugs, das diese Funktion erfüllt.

Im folgenden wird entlang der in Abschnitt 1.1 aufgestellten (und hier nocheinmal angeführten) Anforderungen überprüft, ob diese durch den Einsatz des XWMF-Ansatzes erfüllt werden.

Formal überprüfbare Anforderungen

1. Die Inhalte einer Web-Anwendung sollen, über die Erfüllung der Forderung nach Darstellbarkeit durch Web-Clienten (z.B. mittels HTML [144] und WML [189]) hinaus, auch von Maschinen in ihrer Bedeutung interpretiert werden können.

   Bedingt erfüllt: Die RDF-Spezifikationen lassen zuviel Interpretationsspielraum, so daß es nicht unwahrscheinlich ist, daß verschiedene Implementierungen von RDF-Interpretern eine RDF-Beschreibung auch verschieden (logisch) interpretieren. Um dies zu vermeiden, ist die RDF-Semantik eindeutig festzulegen. Aus diesem Grund zeigt Kapitel 4 einen Vorschlag einer Formalisierung der RDF-Semantik in Prädikatenlogik erster Ordnung. Implementieren zwei RDF-Interpreter diese formal und eindeutig dargestellte (RDF-)Semantik, kommen diese zu den gleichen logischen Schlüssen, womit die Anforderung bedingt erfüllt ist.

2. Die Inhalte der Web-Anwendung sollen in verschiedenen textuellen Repräsentationen (z.B. HTML und WML [189]) zur Verfügung gestellt werden können. Es sollen also nicht nur HTML-basierte Web-Anwendungen entwickelt werden können, sondern auch XML-basierte Anwendungen, die z.B. die Interaktion mit mobilen Web-Clienten ermöglichen.

   Erfüllt: Simplexons definieren durch Code-Templates Schablonen des Realisierungscodes, die durch den Content, den Content-Objekte repräsentieren, ausgefüllt werden. Ein Code-Template kann ein beliebiger String und damit auch z.B. HTML und WML sein.

3. Es soll ein Ansatz zur Erzeugung von Web-Anwendungen gewählt werden, dessen Anwendbarkeit nicht auf ein bestimmtes Gebiet von Web-Anwendungen festgelegt ist (anwendungsbereichsunabhängiges Beschreibungsmodell).

   Erfüllt: Die RDF-Vokabulare und das WebObjectComposition-Vokabular sind anwendungsbereichsneutral, d.h. nicht auf eine bestimmte Art von Web-Anwendungen wie z.B. Produktkataloge, Portale, Bestellsysteme begrenzt. Ein Anwendungsentwickler kann ergänzend anwendungsbereichsabhängiges Vokabular durch Web-Anwendungschemata definieren und zur Modellierung einsetzen.

4. Die Modellierungsdaten sollen zur Web-Anwendung auch im Laufe der Evolution der Web-Anwendung konsistent bleiben.

   Erfüllt: Modellierungsdaten sind Metadaten. Sowohl die Daten, als auch die Metadaten einer Web-Anwendung sind Bestandteil der Web-Anwendungsbeschreibung. Aus dieser wird automatisch und damit konsistent die Web-Anwendung generiert.

5. Das Beschreibungsmodell einer Web-Anwendung soll *durchgängig* sein, d.h. aus den Modellierungsdaten und den (Meta-)Daten einer Web-Anwendung soll die Web-Anwendung automatisch generiert werden können.

Erfüllt: Alle Daten und Metadaten einer Web-Anwendung sind Bestandteil der Web-Anwendungsbeschreibung. Dies, und die Integration von Code-Templates, ermöglichen die durchgängige Beschreibung einer Web-Anwendung, aus der automatisiert der Realisierungcode der Web-Anwendung generierbar ist.

6. Das Beschreibungsmodell soll die Modellierungsdaten und die Inhalte einer Web-Anwendung in einem einheitlichen Datenformat erfassen können.

   Erfüllt: Eine Web-Anwendungsbeschreibung umfaßt sowohl die Daten als auch die Metadaten einer Web-Anwendung. Als einheitliches und standardisiertes Datenformat wird RDF-Syntax genutzt.

7. Das Beschreibungsmodell soll in allen Phasen des Lebenszykluses einer Web-Anwendung eingesetzt werden können, in denen Modellierungsdaten und sonstige Metadaten der Web-Anwendung erfaßt bzw. verarbeitet werden.

   Erfüllt: Grundlegendes Beschreibungsmodell ist der RDF-Graph, der als semantisches Netz kategorisierbar ist. Graphen bzw. semantische Netze sind mächtige Beschreibungsmittel, die weit verbreitet und in allen Phasen des Lebenszykluses einer Web-Anwendung zur Beschreibung von Modellierungs- und sonstigen Metadaten eingesetztbar sind. Hinzu kommt, daß das Vokabular zur Beschreibung einer Web-Anwendung erweiterbar ist. Zusätzliche Modellierungsprimitive bzw. sonstiges Vokabular für Beschreibungsbereiche, die bisher noch nicht berücksichtigt sind, können hinzugefügt werden. Weiter bietet die Verwendung eines standardisierten Austauschformats (RDF) eine Schnittstelle zur Integration von Werkzeugen, die in unterschiedlichen Phasen des Lebenszykluses einer Web-Anwendung eingesetzt werden können.

8. Verwaltungsdaten (z.B. die Festlegung, wer für bestimmte Informationen der Web-Anwendung verantwortlich ist oder wann Informationen veralten) sollen durch das Beschreibungsmodell erfaßbar sein.

   Erfüllt: Verwaltungsdaten sind Metadaten. Objekte, die Inhalte der Web-Anwendung repräsentieren (Content-Objekte), können mit Verwaltungsdaten annotiert werden (dies auch separat und verteilt vom RDF-Dokument, welches das Content-Objekt definiert). Abschnitt 6.3.5 zeigt dies exemplarisch.

9. Das Beschreibungsmodell soll erweiterbar sein, d.h. es sollen neue Beschreibungsmittel hinzugefügt werden können, ohne daß die grundlegen-

den Datenstrukturen des Beschreibungsmodells geändert werden müssen.

Erfüllt: Ein RDF-Graph ist durch das Hinzufügen von Knoten und Kanten erweiterbar, ohne daß an dem grundlegenden Konstrukt: {*Subjekt-Knoten, gerichtete beschriftete Kante, Objekt-Knoten*} etwas geändert werden muß. Neue Beschreibungsmittel können hinzugefügt werden: Einerseits können neue Vokabulare definiert und in das Beschreibungsmodell integriert werden (siehe Abschnitt 6.3.5) und andererseit ist bestehendes Vokabular (z.B. durch subClassOf- oder subPropertyOf-Beziehungen) erweiterbar (dies ist wegen der Verwendung von RDF-Syntax auch Web-basiert und separat (verteilt) vom Ursprungsvokabular möglich).

10. Das Beschreibungsmodell soll die Verwendung verschiedener Abstraktionsebenen zulassen. Die Web-Anwendung soll einerseits sehr abstrakt beschrieben werden können (z.B. in Bezug auf die generellen strukturellen Aspekte der Web-Anwendung). Andererseits soll eine sehr detaillierte Beschreibung möglich sein, wenn es um die Beschreibung von feinstrukturiertem Inhalt geht.[16]

    Erfüllt: XWMF ermöglicht sowohl die Inhaltsstrukturierung durch abstrakte Content-Klassen (die nicht direkt instanziiert werden, sondern indirekt über subClassOf-Beziehungen) verschiedener Abstraktionsebenen, als auch die Definition von Content-Klassen die feinstrukturierten Inhalt beschreiben.

11. Das Beschreibungsmodell soll die Granularität der zu entwerfenden Informationsstruktur nicht vorschreiben.

    Erfüllt: Die Granularität der Informationsstruktur ist abhängig von der Größe der Code-Templates und der Content-Objekte, die nicht vorgeschrieben wird, sondern durch den Entwickler einer Web-Anwendung bestimmt wird.

12. Das Beschreibungsmodell soll die Trennung verschiedener Modellierungsaspekte[17] unterstützen:

    a) Die Definition der Inhalte und die Definition der Darstellung der Inhalte sollen getrennt modelliert werden können.

       Erfüllt: Inhalte werden durch Content-Objekte definiert. Separat davon werden Code-Templates definiert, in die der Inhalt eingefügt

[16] Garzotto bezeichnet dies als *authoring-in-the-large* und *authoring-in-the-small* (vgl. [68, 69]).

[17] Coda et. al. bezeichnen dies als *seperation of concerns* (vgl. [37])

wird. Die Zusammenführung von Code-Template und Inhalt resultiert im Realisierungscode (z.B. ein HTML-Dokument), der den Inhalt darstellt.

b) Die Definition der Inhalte und die Definition der strukturellen Anordnung der Inhalte sollen getrennt modelliert werden können.

Erfüllt: Inhalte werden durch Content-Objekte definiert und separat davon die strukturelle Anordnung der Content-Objekte (in einem gerichteten azyklischen Graphen) durch die hasPart-Beziehung. Content-Objekte "leben" unabhängig von ihrer strukturellen Anordnung. Ein und dasselbe Content-Objekt kann Bestandteil verschiedener Web-Anwendungen sein.

13. Wiederverwendung von (a) Modellierungskonzepten und (b) Web-Anwendungskonzepten soll unterstützt werden.

    Erfüllt: Ein RDF-Schemata, das anwendungsbereichsunabhängiges Vokabular definiert, kann zur Beschreibung von Web-Anwendungen unterschiedlicher Anwendungsbereiche eingesetzt werden (die RDF-Schemata und das WebObjectComposition-Schema sind Beispiele dafür). Vorhandenes Vokabular kann durch die subClassOf- und subPropertyOf-Beziehungen erweitert (dies auch separat und verteilt vom Ursprungsvokabular) und somit wiederverwendet werden. Zudem ist ein Web-Anwendungschema zur Beschreibung von verschiedenen Web-Anwendungen eines bestimmten Anwendungsbereichs einsetzbar (z.B. durch die Wiederverwendung von Simplexon-Klassen). Darüberhinaus kann ein Content-Objekt mehrfach typisiert werden (zusätzliche Typisierung kann auch nachträglich und separat von der ursprünglichen Definition des Content-Objekts erfolgen) und so Inhalte für verschiedene Code-Templates liefern (sei es z.B. HTML oder WML). Die Definition verschiedener Darstellungen eines Content-Objekts ist durch die Definition von Sichten mittels der Properties view und isView möglich.

14. Die Modellierungsdaten und sonstige Metadaten einer Web-Anwendung sollen in einem standardisierten Format[18] verfaßt werden.

    Erfüllt: RDF ist ein vom W3C verabschiedeter Standard.

15. Es soll eine graphische Repräsentation des Beschreibungsmodells verfügbar sein.

[18]Unter einem standardisierten Format werden hier Datenformate verstanden, die von Standardisierungsgremien, wie z.B. dem W3C oder der IETF veröffentlicht werden.

Erfüllt: Eine Web-Anwendungsbeschreibung kann durch einen RDF-Graphen repräsentiert werden.

16. Es sollen frei verfügbare Werkzeuge,[19] deren Programm-Code öffentlich verfügbar ist, zur Verarbeitung des Beschreibungsmodells genutzt werden können.

    Erfüllt: Zur Implementierung des WebObjectComposers und des RDF-Schema-Explorers wurde das frei erhältliche SWI-Prolog eingesetzt. Eine frühe Version des WebObjectComposers ist ebenfalls in XOTcl verfügbar. Darüberhinaus existieren frei verfügbare Implementierungen von RDF-Editoren, die mittels XPCE (graphische Bibliotheken des SWI-Prolog) und $W_a f^e$ erstellt wurden (siehe auch Abschnitt 1.2.4). Der Programm-Code der Werkzeuge ist auf der XWMF-Home-Page verfügbar bzw. kann vom Autor angefordert werden.

Soziotechnische Anforderungen

1. Es soll eine Syntax gewählt werden, für die potentiell viele Werkzeuge entwickelt werden.[20]

   Erfüllt: Als Repräsentationsformat für Web-Anwendungsbeschreibungen wird RDF-Syntax verwendet. RDF ist ein offener Standard des W3C, der als Grundlage des Semantic-Web gilt. Die finanzielle Förderung der Europäischen Union und der USA von Projekten zur Erforschung des Semantic-Web hat zur Entwicklung einer Vielzahl von Werkzeugen geführt, die RDF-Dokumente verarbeiten können. RDF-standardkonforme Werkzeuge können zum Erstellen/Modifizieren/Auswerten von Web-Anwendungsbeschreibungen eingesetzt werden.

2. Die Beschreibungsmittel sollen einfach, verständlich und intuitiv nutzbar sein.

   Bedingt erfüllt: Die RDF-Syntax wurde mit dem Ziel entwickelt, auch von Menschen (leicht) lesbar zu sein. Dieses Ziel wurde verfehlt (im Dokument [174] empfehlen Mitglieder des W3C die Entwicklung einer vereinfachten RDF-Syntax: *A new simplified XML transfer syntax for RDF ... should be produced.* Das W3C hat die RDFCore Working Group zur

[19] Unter frei verfügbaren Werzeugen werden Softwareprogramme verstanden, die z.B. unter der GNU Public License erhältlich sind.

[20] Es kann davon ausgegangen werden, daß für ein standardisiertes Format eine Reihe von Werkzeugen entwickelt werden. Insofern hängt diese Anforderung mit der Anforderung der Verwendung eines standardisierten Formates zusammen.

Entwicklung einer verbesserten RDF-Syntax eingesetzt. Im Abschnitt 7 dieser Arbeit wird eine sehr einfache XML-Syntax (XRDF-Syntax) vorgeschlagen. Bestehende standardkonforme RDF-Dokumente können automatisiert in XRDF-Syntax transformiert werden.

Die XML-Serialisierung von RDF ist jedoch nur eines von mehreren Repräsentationsformaten. Desweiteren kann eine Beschreibung durch Triple oder durch den RDF-Graph repräsentiert werden. Ein RDF-Graph kann als semantisches Netz kategorisiert werden. Semantische Netze wurden mit dem Ziel entwickelt, kognitive Gedächtnismodelle abzubilden, weshalb sie als einfach, verständlich und intuitiv nutzbar gelten. Insofern ist die Anforderung bedingt erfüllt.

3. Die graphische Repräsentation soll den Entwicklungsprozeß unterstützen.

   Erfüllt: Web-Anwendungsbeschreibungen können durch semantische Netze (RDF-Graphen) repräsentiert werden, die zur Kommunikationsgrundlage für Gespräche zwischen allen am Entwicklungsprozeß Beteiligten nutzbar sind.

4. Ein arbeitsteiliger Entwicklungsprozeß soll unterstützt werden.

   Erfüllt: Die RDF-Syntax wurde mit dem Ziel entwickelt, Aussagen über im Web verteilte Resourcen tätigen zu können. Die Repräsentation der Aussagen kann separat (und im Web verteilt) von der zu beschreibenden Resource gespeichert werden. Dies kann einen arbeitsteiligen Entwicklungs- und Verwaltungsprozeß unterstützen, da verschiedene Personen entwicklungs- oder verwaltungsbezogene Information in separaten RDF-Dokumenten speichern, die bei Bedarf automatisiert über das Web angefordert werden.

Zusammenfassend läßt sich festhalten, daß der XWMF-Ansatz die in Abschnitt 1.1 aufgestellten Anforderungen erfüllt.

7 XRDF – ein alternativer Vorschlag zu RDF

Die an RDF bereits in den vorrangegangenen Kapiteln vorgenommene Kritik, sowie die Darstellung der Ungenauigkeiten und Unklarheiten führt zu der Frage, ob Verbesserungen an RDF vorgenommen werden sollen und wie dies geschehen kann. Die Frage ob Verbesserungen vorgenommen werden sollen beantwortet bereits das W3C-Dokument *The Cambridge Communiqué* [174]: "A new simplified XML transfer syntax for RDF ... should be produced".

In diesem Abschnitt werden (unter dem Namen XRDF) ein alternatives Datenmodell und dazugehörige syntaktische Repräsentationen vorgeschlagen. XRDF ist zu RDF insofern kompatibel, als daß sich eine XRDF-Beschreibung in eine RDF-Beschreibung transformieren läßt und vice versa. Der wesentliche Unterschied bezüglich des Datenmodells ist, daß XRDF *verschachtelte* Triple *und Listen* als strukturelle Primitive einsetzt. Der strukturelle Kontext, in dem Primitive verwendet werden, bleibt in allen XRDF-Repräsentationen erhalten. Auf diese strukturelle Ebene können semantische Ebenen aufbauen (Beispiele zur Behandlung von formal definierter Semantik werden gezeigt). Bezüglich der Syntaxebene ist der wesentliche Unterschied, daß eine sehr einfache und, im Gegensatz zur RDF-Syntax, validierbare XML-Syntax zur Repräsentation von Beschreibungsmodellen angegeben wird. Eine XRDF-Beschreibung läßt sich formal durch syntaktische Transformation sowohl in eine äquivalente graphische Repräsentation überführen, als auch in das XRDF-Datenmodell. Die graphische Repräsentation ist gegenüber einem RDF-Graphen übersichtlicher[1] und ermöglicht zudem die Komposition und Dekomposition graphischer Repräsentationen, was die Übersicht von RDF-Beschreibungen (vor allem bei werkzeugunterstütztem Entwurf von Beschreibungsmodellen) noch einmal erhöhen kann. Der hier vorgestellte Ansatz kann zudem helfen, einige Unklarheiten im Zusam-

[1] Ohne genaue Definition des Begriffs Übersichtlichkeit ist dies natürlich weitgehend eine subjektives Kriterium. Der Begriff meint, in dem hier verwendeten Zusammenhang, die Möglichkeit der Zusammenfassung/Sammlung/Komposition von Gruppen/Listen von Primitiven. Eine Komposition lässt sich durch einen Repräsentanten darstellen und erhört so durch Informationsweglassung die Übersicht.

menhang mit der Interpretation der Reifikation und der Container des RDF-Datenmodells zu beseitigen.

Im folgenden wird kurz der grundlegende Ansatz von XRDF vorgestellt: Zur Modellierung einer Beziehung (bzw. Relation) zwischen einem Statement und einer Resource oder zwischen zwei Statements wird im RDF-Datenmodell die Reifikation benutzt. Zur Modellierung einer Beziehung zwischen einer Entität (dies kann sein eine Resource, ein Statement oder eine Liste von Entitäten) und einer Liste von Entitäten werden Container (im einzelnen rdf:Bag, rdf:Seq oder rdf:Alt) genutzt. Beide Konstrukte (Reifikation und Container) sind notwendig, um verschachtelte Ausdrücke bzw. Listen durch flache Triple repräsentieren zu können. Die Bedeutung dieser Konstrukte ist jedoch durch das RDF-Datenmodell nicht eindeutig festgelegt. Beispielsweise ist die Bedeutung der Zuordnung eines Prädikats zu einem Reifikanten[2] im RDF-Datenmodell nicht festgelegt. Zur Erläuterung: angenommen r reifiziert $[s, p, o]$ und $[r, p_2, o_2]$ sei gegeben – steht nun $[s, p, o]$ *in* p_2*-Relation zu* o_2 oder ist das Triple buchstäblich zu betrachten, d.h. r steht für sich selbst, so das gilt r *steht in* p_2*-Relation zu* o_2). Diese Ungenauigkeit klärt XRDF zugunsten der ersten Möglichkeit.

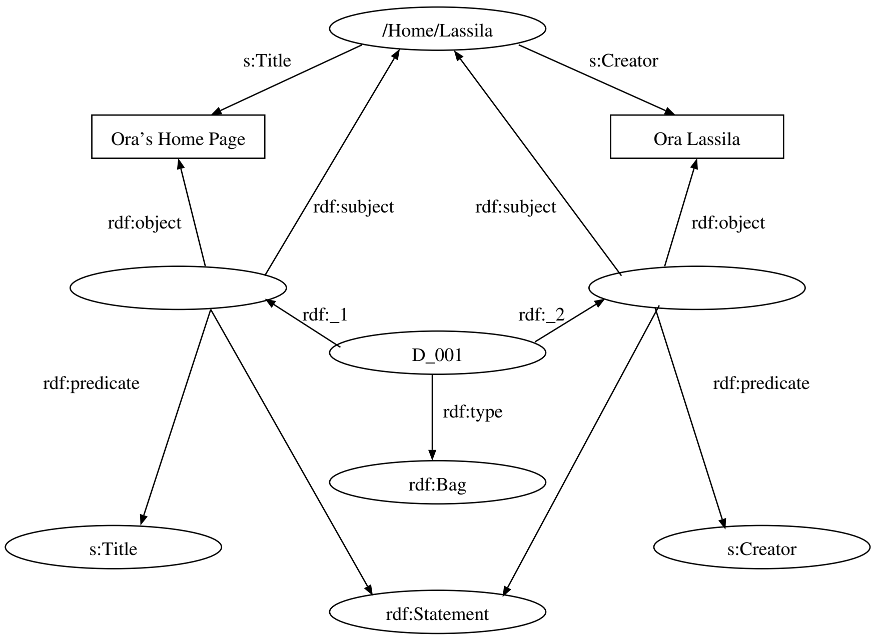

Abbildung 7.1: Repräsentation des Konzepts rdf:Bags und der Reifikation (s.a. Abbildung 9 in [101])

Den Unterschied in der Verwendung der grundlegenden Primitive von RDF und XRDF verdeutlichen die Abbildungen 7.1 und 7.2, die graphische Repräsenta-

[2]Diese Terminologie wird im nachfolgenden Text erläutert.

tionen der gleichen Beschreibung sind. Abbildung 7.1 zeigt einen RDF-Graphen (der durch eine Menge flachter Triple beschrieben werden kann) und Abbildung 7.2 den intentionsäquivalenten XRDR-Graphen (der durch eine Menge verschachtelter Triple beschrieben werden kann).

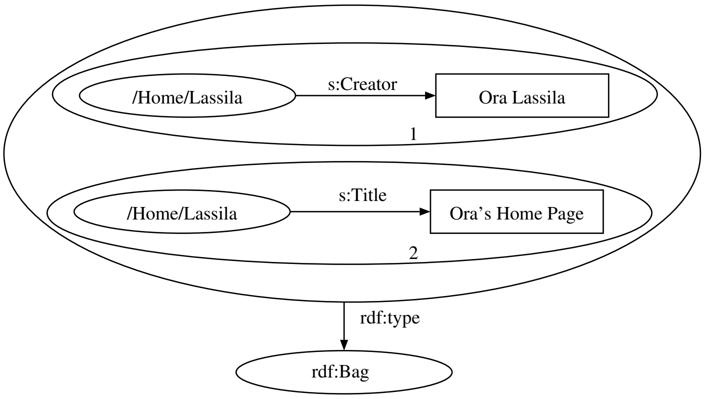

Abbildung 7.2: Eine intentionsäquivalente verschachtelte Repräsentation

Für das XRDF-Datenmodell wird angenommen, daß eine Resource, die eine *Reifikation* repräsentiert, Platzhalter für das entsprechenden Triple ist und das jede Resource, die einen Container repräsentiert, Platzhalter für eine Liste von Entitäten[3] ist. Semantische Äquivalenz sei gegeben, wenn jeder Repräsentant einer Reifikation durch das entsprechende Triple ersetzt wird, bzw. wenn jeder Repräsentant eines Containers durch die entsprechende Liste von Entitäten ersetzt wird. Triple, die in der flachen Triple-Notation von RDF zur Konstruktion einer Reifikation oder eines Containers technisch notwendig sind (und die sonst keine Semantik ausdrücken), werden eliminiert.

Es wird gezeigt, daß ein XRDF-Datenmodell die Essenz (Definition siehe Abschnitt 7.3) eines RDF-Datenmodells erfaßt. Im folgenden werden die beiden, dem XRDF-Datenmodell zugrundeliegenden, Annahmen im Detail erläutert.

7.1 Annahmen

Sei S eine Menge flacher Triplen.

Annahme 1: Sei r der *Reifikant*[4] des Triples $[s,p,o]$, d.h. die Menge S enthält die Triple $[r, \text{rdf:subject}, s]$, $[r, \text{rdf:predicate}, p]$, $[r, \text{rdf:object}, o]$ und

[3] Eine Entität kann eine Resource, ein Literal, ein Statement oder eine Liste von Entitäten sein.

[4] In der RDF-M+S-Spezifikation wird der Reifikant *reified statement* genannt. Diese Wortwahl ist jedoch nicht treffend. Das was mit *reified statement* bezeichnet wird, ist eine Resource

$[r, \text{rdf:type}, \text{rdf:Statement}]$[5]. Weiter sei r in einem Statement, beispielsweise $[s_1, p_1, r]$, angegeben. *Die Interpretation dieser Untermenge von Statements ist folgende:* p_1 *setzt* s_1 *in Relation zu* $[s, p, o]$*. Der intendierte Ausdruck sei dann durch eine verschachtelte Triple-Notation folgendermaßen erfaßt:* $[s_1, p_1, [s, p, o]]$.

Annahme 2: Sei c ein Container, beispielweise vom Typ rdf:Seq, so daß gilt $[c, \text{rdf:type}, \text{rdf:Seq}] \in S$. Weiter sei $R_C = (r_1, \ldots, r_n)$ eine n-elementige Sequenz von Resourcen, die die Elemente von c enthält, so daß gilt $[c, \text{rdf:\_}i, r_i] \in S$ für jedes $i \in \mathbb{N}, 1 \leq i \leq n$. Der Container c sei in einem anderem Element von S, beispielsweise $[s, p, c]$, verwendet. *Die Interpretation dieser Untermenge von Statements ist folgende:* p *setzt* s *in Beziehung zu* $(r_1, \ldots, r_n)$*. Der intendierte Ausdruck sei dann durch verschachtelte Triple-Notation folgendermaßen erfaßt:* $[s, p, (r_1, \ldots, r_n)]$.[6]

Auf Grundlage dieser beiden Annahmen, können das Container und Reifikation genutzt werden, um Strukturen beliebiger Komplexität auszudrücken und zwar durch ein sehr einfaches Modell, das auf flachen Triplen basiert.

Das im folgendem vorgestellte XRDF-Datenmodell nutzt zur Repräsentation der Ausdrucksintention, die den RDF-Konstrukten *Reifikation* und *Container* zugrundeliegt, verschachtelte Triple und Listen. Im Abschnitt 7.3 wird der Beweis geführt, daß jedes RDF-Datenmodell durch ein intentionsäquivalentes XRDF-Datenmodell ausgedrückt werden kann *und* vice versa. Ein XRDF-Datenmodell kann deshalb (nach Transformation) von allen RDF-konformen Werkzeugen weiterverarbeitet werden. In Abschnitt 7.4 werden zwei Repräsentation für das XRDF-Datenmodell vorgeschlagen: (1) eine graphische Notation (XRDF-Graph) und (2) eine XML-Notation (XRDF-Syntax). Für die XRDF-Syntax (kann und) wird eine DTD angegeben, weshalb die Notation gegenüber der RDF-Syntax zumindest in diesem Punkt vorteilhaft ist (da zur RDF-Syntax keine DTD angegeben werden kann). In Abschnitt 7.5 wird durch prozedurale Beschreibungen der syntaktischen Transformationen gezeigt, wie die verschiedenen XRDF-Repräsentationen ineinander überführbar sind. Abschnitt 7.6 untersucht die Beziehung zwischen strukturellen und semantischen Aspekten

und kein Statement, wie die Bezeichnung suggeriert. Diese Resource z.B. r ist Repräsentant für ein Statement und ist Bestandteil einer Reifizierung. D.h. r ist **nicht** Element der Menge *Statements* (vgl. Abschnitt 3.2). r hat den rdf:type rdf:Statement, was im RDF-Datenmodell bedeutet, daß es sich um einen String handelt, der eine Resource repräsentiert. r repräsentiert ein Statement, ist aber keins.

[5] Die gezeigten Triple werden in der Ordnung $[subject, predicate, object]$ angegeben.

[6] Anmerkung: Zusammengenommen können die beiden Annahmen genutzt werden, um Listen von Statements zu erfassen.

und es werden Vorschläge gemacht, wie zusätzliche Semantik für das XRDF-Datenmodell entwickelt werden kann. Abschließend diskutiert Abschnitt 7.8 den XRDF-Ansatz.

7.2 XRDF-Datenmodell

Sei A ein geeignet gewähltes Alphabet.[7] Sei A^* die Menge der Strings, die über dem Alphabet A definiert ist. Die folgende Grammatik definiert Ausdrücke der Form R über A^* rekursiv:

$$R \quad ::= \quad r \quad | \quad '(' \, R \, ',' \, R \, ')' \quad | \quad '[' \, R \, ',' \, R \, ',' \, R \, ']'$$

Wobei r Elemente von A^* bezeichnet. Ein (Sub-)Ausdruck dieser Form wird *Atom* genannt. Ein (Sub-)Ausdruck der Form R wird *Resource* genannt. Ein (Sub-)Ausdruck der Form R, welcher mit dem Muster $[R, R, R]$ übereinstimmt, wird *Statement* genannt. Ein (Sub-)Ausdruck der Form R, welcher mit dem Muster (R, R) übereinstimmt, wird *Liste* genannt. Es werden nur endliche Mengen von endlichen Ausdrücken betrachtet.

Adressierung von Resourcen

Jeder Ausdruck kann als XRDF-Beschreibung m interpretiert werden. Jede XRDF-Beschreibung m besteht aus Sub-Ausdrücken von R. Die maximale Menge von Sub-Ausdrücken von m wird $R(m)$ bezeichnet. Für jeden Sub-Ausdruck von $R(m)$ kann eine eindeutige Adresse angegeben werden.

Gegeben sei ein Ausdruck m. Sei e_r der (Sub-)Ausdruck der r direkt (d.h. auf erster Verschachtelungsebene) enthält.

1. Atom: $p(r) = p(e_r[r]) = 1$ für jedes e_r.

2. In Liste: Gegeben sei $e_r = R_1, \ldots, R_N$, $r = R_i$, $1 \leq i \leq n$, dann gilt $p(r) = p(e_r[r]) = i$

3. In Triple: Gegeben sei $e_r = [R_1, R_2, R_3]$, $r = R_i$, $1 \leq i \leq 3$, dann gilt $p(r) = p(e_r[r]) = i$.

[7] Unter einem geeigneten Alphabet wird hier ein Alphabet verstanden, daß es ermöglicht URIs, Literale und XML-Dokumente auszudrücken: z.B. das im XML-Standard – Abschnitt *Character Classes* gezeigte Alphabet.

Für jeden gegebenen Sub-Ausdruck S von m kann ein Pfad zu jedem Bestandteil s von S wie folgt angegeben werden:

$p(s) \;=\; p(e_s) \;+\; / \;+\; p(e_s(s))$

wobei + eine Konkatenation beschreibt und e_s der (Sub-)Ausdruck ist, der s direkt enthält.

Dies ermöglicht die eindeutige Adressierung jedes Sub-Ausdrucks bei gegebenem Ausdruck m. Das folgende Beispiel verdeutlicht den Zusammenhang:

Beispiel:

```
[(Wolfram, Reinhold, Eckhart), sagen,
         ( [RDF,  benutzt, Triple],
           [XRDF, benutzt, (verschachtelte Triple, Listen)] )
]
```

Für den Sub-Ausdruck `benutzt` können alle Vorkommen folgendermaßen adressiert werden: 3/1/2 und 3/2/2. Abschnitt 7.4 zeigt eine XML-Syntax zur Repräsentation von XRDF-Beschreibungen. Der hier gezeigte Adressierungsmechanismus passt zu dem XPointer-Adressierungsmeachnismus für Child Sequences (sie Abschnitt 4.2.3 der XPointer-Spezifikation), weshalb XPointer-Adressen bei Repräsentation von XRDF-Beschreibungen in XML zur Adressierung von Sub-Ausdrücken genutzt werden können.

Hinweis zur Verwendung der Notation: an den Stellen, an denen keine Fehlinterpretation möglich ist, wird im folgenden auch $(r_1, r_2, \ldots, r_{n-1}, r_n)$ statt der schwerfälligeren Notation $(r_1, (r_2, (\ldots, (r_{n-1}, r_n) \ldots)$ verwendet. Zur Verbesserung der Lesbarkeit der Beispiele wird das Komma an einigen Stellen nicht angegeben. Für jedes Atom r wird angenommen, daß Zeichen der Grammatik im Atom (ohne Angabe von Escape-Zeichen) verwendet werden können. Bei programmtechnischer Implementierung ist für eine Unterscheidung durch einen geeigneten Mechanismus zu sorgen.

7.3 Beziehungen zwischen XRDF- und RDF-Datenmodell

Für die folgenden Betrachtungen wird davon ausgegangen, daß die Menge der Statements einer RDF-Beschreibung in dem Sinne wohlgeformt ist, daß weder ein Reifikant noch ein Container-Repräsentant Teil einer (möglicherweise komplexen) Struktur ist, die durch den Reifikanten bzw. Container-Repräsentanten dargestellt wird.[8]

Weiter unten folgt die Vorstellung eines Algorithmus, der Resourcen in Schichten anordnet, so daß jede Resource in einer Schicht eine Struktur repräsentiert, die aus Resourcen einer tieferen Schicht besteht. Der Algorithmus kann darüber hinaus dazu genutzt werden, nicht-wohlgeformte (Unter-)Mengen von Statements einer RDF-Beschreibung zu finden. Weiter wird im folgenden die gegenseitige Abbildbarkeit von XRDF-Beschreibungen und RDF-Beschreibungen formal gezeigt.

Seien $s, p, o, s_1, o_1, r_1, \ldots, r_n$ Entitäten, d.h. entweder ein Atom, ein Statement oder eine Liste von Entitäten, wenn das XRDF-Datenmodell betrachtet wird, bzw. Resourcen oder Literale (die nur an Objekt-Position erlaubt sind), wenn das RDF-Datenmodell betrachtet wird.

Definition 1 (Reifikation).
Gegeben sei Resource r und die folgende Menge von Statements, T^r:

$$\begin{aligned} T^r = \{ \quad & [r, \mathit{rdf{:}subject}, s],\ [r, \mathit{rdf{:}predicate}, p], \\ & [r, \mathit{rdf{:}object}, o], [r, \mathit{rdf{:}type}, \mathit{rdf{:}Statement}] \quad \} \end{aligned}$$

Dann wird r Reifikant *von $[s, p, o]$ genannt und T^r wird* Reifikation *von $[s, p, o]$ genannt.*

Definition 2 (Reifikation: Ableitung, Konsequenz).
Sei u ein geschachteltes Statement des XRDF-Datenmodells der Form $[[s, p, o], p_1, o_1]$. Sei r ein Reifikant von $[s, p, o]$ und T^r die entsprechende Reifikation. Die Menge $D = [r, p_1, o_1] \cup T^r$ wird Ableitung von u genannt. u wird Konsequenz *von D genannt (analog für $u = [s_1, [s, p, o], o_1]$ und $[s_1, p_1, [s, p, o]]$). Hinsichtlich der Menge $\mathcal{C}$ von Statements, gilt dann die Aussage: u ist* ableitbar *in $\mathcal{C}$ wenn gilt $D \subseteq \mathcal{C}$.*

Definition 3 (Container).
Gegeben sei die Resource c und die folgende Menge von Statements (mit $X \in \{\mathit{rdf{:}Seq},\ \mathit{rdf{:}Alt}, \mathit{rdf{:}Bag}\ \}$): $T^c = \{\ [c,\ \mathit{rdf{:}type},\ X]\} \cup \{[c,\ \mathit{rdf{:}\_i},\ r_i] \mid i \in \mathbb{N}, 1 \leq i \leq n\}$. Dann wird c Container *genannt, T^c wird n-stellige* Container-Repräsentation *und die Elemente der n-stelligen Sequenz $R_c = (r_1, \ldots, r_n)$, die Entitäten sind, werden* Elemente *von c genannt.*

[8] Im RDF-Datenmodell ist es beispielsweise möglich, ein Statement zu reifizieren, welches den das Statement repräsentierenden Reifikanten enthält. Dies wird als Fehler des RDF-Datenmodells aufgefaßt, d.h. es wird angenommen, daß eine solche Struktur nicht erlaubt sein sollte. Das gleiche gilt für Container, die sich selbst enthalten.

Anmerkung: die Definitionen 1 und 3 gelten für beide Modelle.

Definition 4 (Container: Ableitung, Konsequenz).
Sei u ein geschachteltes Statement des XRDF-Datenmodells der Form $[(r_1,\ldots,r_n),p,o]$. Sei c ein Container vom Typ rdf:Seq für die Elemente $(r_1,\ldots,r_n)$ und sei T^c die entsprechende Container-Repräsentation. Die Menge $D = [c,p,o] \cup T^c$ wird dann Ableitung *von u genannt. u wiederum wird* Konsequenz *von D genannt (analog für $u = [s,(r_1,r_2,\ldots,r_n),o]$ und $[s,p,(r_1,r_2,\ldots,r_n)]$). Hinsichtlich der Menge $\mathcal{C}$ von Statements gilt dann die Aussage: u ist* ableitbar *in $\mathcal{C}$ wenn $D \subseteq \mathcal{C}$.*

Anmerkung: Eine *Konsequenz* kann kein Statement des flachen RDF-Datenmodells sein. Die *Ableitung* eines Statements, für das es nur eine Verschachtelungsebene gibt, kann jedoch vollständig aus Statements des flachen Triple-Datenmodells bestehen. Um eine Art Gleichheit zwischen Mengen von Statements des XRDF-Datenmodells und des RDF-Datenmodells definieren zu können, muß notwendigerweise festgelegt werden, wie verschachtelte Statements aus einer Menge flacher Statements rekursiv abgeleitet werden können.

Definition 5 (Wurzel, Verwurzelt, Hülle).
Sei $\mathcal{O}$ eine Menge flacher Statements des RDF-Datenmodells. Sei $\mathcal{N}$ eine Menge von Statements des XRDF-Datenmodells. Sei u ein Statement des XRDF-Datenmodells. Dann gilt die Aussage: u ist in $\mathcal{O}$ verwurzelt, *wenn entweder*
(1) $u \in \mathcal{O}$ oder
(2) eine Ableitung D von u angegeben werden kann, so daß jedes Statement $t \in D$ in $\mathcal{O}$ verwurzelt *ist.*

Es gilt weiter: $\mathcal{N}$ ist verwurzelt *in $\mathcal{O}$, wenn jedes Statement $n \in \mathcal{N}$ in $\mathcal{O}$* verwurzelt *ist. $\mathcal{O}$ wird* Wurzel *von u genannt, wenn u in $\mathcal{O}$ verwurzelt ist. $\mathcal{O}$ wird* minimale Wurzel *von u genannt, wenn $\mathcal{O}$ eine Wurzel von u ist und für jedes Statement $t \in \mathcal{O}$ gilt: u ist nicht in $\mathcal{O}\backslash\{t\}$ verwurzelt.*

$\mathcal{O}$ wird Wurzel *von $\mathcal{N}$ genannt, wenn jedes Statement $n \in \mathcal{N}$ in $\mathcal{O}$ verwurzelt ist. $\mathcal{O}$ wird* minimale Wurzel *von $\mathcal{N}$ genannt, wenn $\mathcal{O}$ eine Wurzel von $\mathcal{N}$ ist und für jedes Statement $t \in \mathcal{O}$ ein Statement $n \in \mathcal{N}$ gefunden werden kann, so daß n nicht in $\mathcal{O}\backslash\{t\}$ verwurzelt ist. Es gilt dann: $\mathcal{N}$ ist eine* Hülle *von $\mathcal{O}$, wenn $\mathcal{O}$ eine minimale Wurzel von $\mathcal{N}$ ist – alternativ kann gesagt werden, daß $\mathcal{N}$ und $\mathcal{O}$* intentionskonsistent *(oder einfach* konsistent*) sind.*

Abbildung 7.3 zeigt die Relation zwischen Mengen verschachtelter und flacher Statements. Wenn M leer ist, ist $\mathcal{O}$ eine *Wurzel* von $\mathcal{N}$. Wenn M und P leer sind, dann ist $\mathcal{O}$ eine *minimal Wurzel* von $\mathcal{N}$.

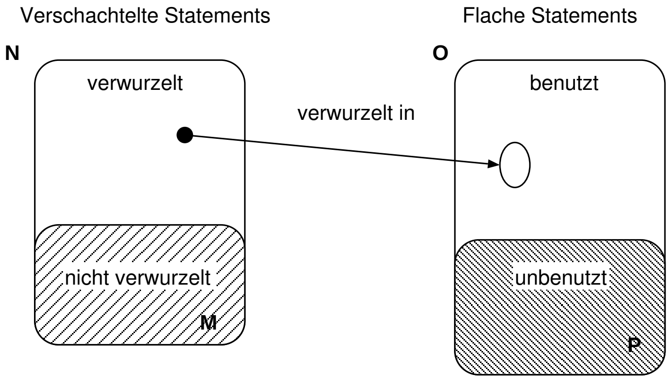

Abbildung 7.3: Relation zwischen Mengen verschachtelter und flacher Triple.

Beispiel Die folgende Menge $\mathcal{O}$ flacher Triple ist eine *minimale Wurzel* (d.h., kein Statement kann aus $\mathcal{O}$ eliminiert werden) für das verschachtelte Triple [Jemand sagt [XRDF benutzt (Triple Listen)]]:

{ [Jemand sagt r_1],
[r_1 rdf:type rdf:Statement],
[r_1 rdf:subject XRDF],
[r_1 rdf:predicate benutzt],
[r_1 rdf:object l_1],
[l_1 rdf:type rdf:Seq],
[l_1 rdf:_1 Triple],
[l_1 rdf:_2 Listen] }

Die Menge
{ [Jemand sagt [XRDF benutzt (Triple Listen)]] }
und die Menge
{ [Jemand sagt [XRDF benutzt (Triple Listen)]],
[Jemand sagt r_1], [r_1 rdf:predicate benutzt]}
sind (neben endlich vielen anderen[9]), *Hüllen* von $\mathcal{O}$.

Gesucht werden jedoch nur Hüllen, die die minimal notwendige Anzahl von Statements enthalten, um die Intention (im obigen Sinne) der zugrundeliegenden Mengen flacher Statements zu erfassen.

Definition 6 (Essenz).
Sei $\mathcal{O}$ eine Menge flacher Statements des RDF-Datenmodells. Sei $N^{\mathcal{O}} = \{\mathcal{N} | \mathcal{N}$ ist eine Menge von Statements des XRDF-Datenmodells $\wedge$ $\mathcal{N}$ ist eine Hülle

[9] Im Gegensatz dazu gibt es, wegen der möglichen Namensvariationen für Container und Reifikanten, für eine Menge verschachtelter Statements im allgemeinen eine unendliche Menge von möglichen minimalen Wurzeln.

von $\mathcal{O}\}$ *die* Menge der möglichen Hüllen *von* $\mathcal{O}$*. Die Untermenge* $N^{\mathcal{O}}_{\min} = \{\mathcal{N} \in N^{\mathcal{O}} \mid \nexists \mathcal{M} \in N^{\mathcal{O}}$ *mit* $|\mathcal{M}| < |\mathcal{N}|\}$ *ist die* Menge der minimalen Hüllen*. Ein Element von* $N^{\mathcal{O}}_{\min}$ *wird* minimale Hülle *oder* Essenz *von* $\mathcal{O}$ *genannt – alternativ kann gesagt werden, das* $\mathcal{N}$ *und* $\mathcal{O}$ intentionsäquivalent *(oder, kurz gesprochen,* äquivalent*) sind.*

Hinweis: Aufgrund der Definition der Ableitung ist die minimale Hülle einer gegebenen Menge von Statements des RDF-Datenmodells eindeutig.

Beispiel Die minimale Hülle der Menge $\mathcal{O}$ des obigen Beispiels ist:
{ [Jemand sagt [XRDF benutzt (Triple Listen)]] }.

Auf Grundlage der obigen Definitionen können die beiden folgenden Theoreme bewiesen werden. Das erste Theorem sagt im wesentlichen aus, daß jede Menge verschachtelter Statements durch eine intentionskonsistente Menge flacher Statements ausgedrückt werden kann. Umgekehrt sagt das zweite Theorem aus, daß jede Menge flacher Statements durch eine intentionsäquivalente Menge verschachtelter Statements ausgedrückt werden kann.

Theorem 1: *Für jede Menge* $\mathcal{N}$ *von Statements des XRDF-Datenmodells kann eine Menge* $\mathcal{O}$ *von Statements des RDF-Datenmodells gefunden werden, so daß* $\mathcal{O}$ *eine minimale Wurzel von* $\mathcal{N}$ *ist.*

Beweis (konstruktiv): Jedes geschachtelte Statement kann schrittweise in eine Menge flacher (Zwischen-)Statements transformiert werden, solange bis keine weitere Transformationen mehr möglich sind. Die Einführung von Namen (Symbole) für Container oder Reifikanten ist dabei durch eine bijektive Funktion sicherzustellen. Zur Erläuterung folgendes Beispiel:

Initialer Ausdruck:	[Jemand sagt [XRDF benutzt (Triple Listen)]]
Schritt 1:	[Jemand sagt [XRDF benutzt l_1]],
(füge Ableitung der	[l_1 rdf:type rdf:Seq],
Liste hinzu)	[l_1 rdf:_1 Triple], [l_1 rdf:_2 Liste],
Schritt 2:	[Jemand sagt r_1],
(füge Ableitung des	[r_1 rdf:type rdf:Statement], [r_1 rdf:subject XRDF]
verschachtelten	[r_1 rdf:predicate benutzt], [r_1 rdf:object l_1],
Statements hinzu)	[l_1 rdf:type rdf:Seq],
	[l_1 rdf:_1 Triple], [l_1 rdf:_2 Listen]

Dies kann folgendermaßen formalisiert werden. Sei C^E eine Menge von Statements[10] des XRDF-Datenmodells. Folgender Algorithmus bestimmt eine inten-

[10] Beliebige Mengen von Ausdrücken bzw. Resourcen könnten ebenfalls erlaubt werden. Das würde nur eine einfache, aber für diese Darstellung unnötige, Erweiterung erfordern.

tionskonsistente Menge C^R von Statements des RDF-Datenmodells.

Algorithm *Flaten(***In***: C^E)*
$C^R = \emptyset$. **Foreach** $t \in C^E$ **do**
Expand(t,0,C^R)

Function *Expand* (**In**: Expression t, **In**: Int l, **InOut**: Set of Statements E) **returns** *Symbol*
If $t \in A^*$ **then return** t
If $Match(t, [s, p, o])$[11] **then** /* Statement */
s_r = Expand(s,l+1,E);
s_p = Expand(p,l+1,E);
s_o = Expand(o,l+1,E);
r =Symbol[12](t);
If $(l = 0)$[13] **then** $E = E \cup \{ [s_r, s_p, s_o]\}$; **return** *EmptySymbol*
else
$E = E \cup \{$ $[r, \text{rdf:type}, \text{rdf:Statement}]$,
$[r, \text{rdf:subject}, s_r]$,
$[r, \text{rdf:predicate}, s_p]$,
$[r, \text{rdf:object}, s_o]$ $\}$;
return r
If Match(t,($r_1, \ldots, r_n$)) **then** /* List */
r =Symbol(t);
$E = E \cup \{ [r, \text{rdf:type}, \text{rdf:Seq}]\}$
For $1 \leq i \leq n$ **do**
s_i =Expand(r_i,l+1,E);
$E = E \cup \{ [r, \text{rdf:\_}i, s_i] \}$
return r

[11]Die Funktion *Match* ermöglicht den Zugriff auf die Elemente der Struktur von t (auf erster Verschachtelungsebene), indem die einzelnen Elemente dem zweiten Argument zugewiesen werden, wenn eine Übereinstimmung gefunden wird. Bei Übereinstimmung gibt die Funktion den Wert TRUE zurück, anderennfalls FALSE (ähnlich wie der Operator =∼ der Scriptsprache Perl.)

[12]Die Funktion *Symbol* gibt ein eindeutiges Symbol für jeden (Sub-)Ausdruck t zurück, der noch nicht in Form eines flachen Triples vorkommt, oder das schon bekannte Symbol bei bereits bekanntem (Sub-)Ausdruck.

[13]Der Ausdruck auf oberster Ebene ist immer ein Statement. Es ist nicht nötig, dieses Statement zu reifizieren, da der Reifikant (in dieser speziellen Expansion) nicht benutzt wird. Ein solcher Reifikant könnte jedoch als ein Repräsentant für eine vollständige RDF-Beschreibung dienen.

Nachweis der Korrektheit des Algorithmus: (1) Der Algorithmus terminiert. Dazu folgende Betrachtungen: Die Funktion *Expand* läuft rekursiv absteigend durch die Struktur der Eingabeausdrücke. Die Rekursion terminiert in jedem Zweig der Struktur, sobald ein Element von A^* gefunden wird (was ultimativ der Fall sein wird, da per Annahme gilt, daß alle Ausdrücke endlich sind). Außerdem wird jeder Subausdruck genau einmal betrachtet. (2) C^E und die berechnete Menge C^R sind intentionskonsistenz. Dazu folgende Betrachtungen: Alle Statements, die C^R hinzugefügt werden, sind flach. Der Algorithmus erzeugt eine Ableitung für jedes verschachtelte Statement, indem rekursiv für jeden vorkommenden Ausdruck eine Ableitung erzeugt wird. Deshalb ist jedes Statement von C^E in der erzeugten Menge C^R verwurzelt. C^R ist eine minimale Wurzel, da nur Elemente von Ableitungen zu C^R hinzugefügt werden. $\boxminus$

Anmerkungen zur Funktion *Symbol*. Diese Funktion berechnet einen *eindeutigen* Namen für syntaktisch identische Subausdrücke, d.h. angenommen, das Statement [Ecki mag RDF] tritt zweimal auf, dann wird hierfür nur eine Reifizierung vorgenommen (obwohl – um den Algorithmus einfach zu halten – das Statement zweimal verflacht wird, was jedoch in einer identischen Menge flacher Statements resultiert, da bei mengentheoretischer Betrachtungsweise doppelte Elemente als ein Element betrachtet werden). Das macht es einfach, syntaktisch äquivaltente Ausdrücke in der Menge flacher Triplen zu finden (gleiche Ausdrücke haben den gleichen Namen). Allerdings kann dies die Behandlung unterschiedlicher Bedeutungen erschweren, die sich aus dem mehrfachen Auftreten von syntaktisch identischen Ausdrücken an verschiedenen Stellen ergeben. Dazu wäre die Betrachtung von strukturellem Kontext notwendig. Die Bedeutung von Kontext in Zusammenhang mit RDF-Beschreibungen bedarf noch der weiteren Diskussion in der RDF-Community. Die Behandlung von Kontext sollte jedoch aufbauend auf dem XRDF-Datenmodell möglich sein, da gerade durch dieses Modell strukturelle Aspekte besonders berücksichtigt werden. Es ist natürlich leicht möglich, für jedes Auftreten von syntaktisch identischen Ausdrücken (an verschiedenen Stellen in der Beschreibung) ein neues Symbol zu erzeugen.

Theorem 2: *Für jede Menge $\mathcal{O}$ von Statements des RDF-Datenmodells kann eine Menge $\mathcal{N}$ von Statements des XRDF-Datenmodells gefunden werden, so daß $\mathcal{N}$ eine minimale Hülle von $\mathcal{O}$ ist.*

Beweis: (konstruktiv) Die in einer RDF-Beschreibung verwendeten Resourcen können in Schichten angeordnet werden, wenn die Annahme gilt, daß keine zirkulären Referenzen in Reifikationen bzw. Containern existieren (für Einzelheiten siehe weiter unten). Der folgende Algorithmus bestimmt entweder eine

Schichtung (engl. stratification) oder identifiziert zirkuläre Referenzen.

Algorithm *Stratification(***In***: C^E)*
Initial seien keine Resourcen und Literale markiert.
[Berechne Schicht 0]
Markiere jedes Literal als Element der Schicht 0. Markiere alle Resourcen, die weder Reifikant noch Container sind, als Elemente der Schicht 0.
[Berechne weitere Schichten]
while eine Resource r existiert, die *nicht markiert* ist *und* jede Resource oder jedes Literal das durch r repräsentiert wird (diese Menge von Entitäten sei E genannt)[14]markiert ist **do**
Bestimme den höchsten Rang j bezüglich der Resourcen in E.
Markiere r als Element der Schicht $j+1$.
[Prüfe Validität]
if eine nicht-markierte Resource existiert
then return "ERROR: es sind gegenseitig referenzierende Strukturen vorhanden"
else return "OK - eine Schichtung konnte bestimmt werden".

Bei gegebener, endlicher Eingabemenge, terminiert der Algorithmus. In jeder Iteration wird eine unmarkierte Resource markiert. Kommt es zu der Ausgabe "OK", liegt die folgende Bedingung zugrunde: jede Resource r, die eine Struktur repräsentiert, ist Mitglied einer höheren Schicht als die Resourcen/Literale, die Teil der repräsentierten Struktur sind. Gibt der Algorithmus "ERROR" aus, repräsentiert mindestens eine Resource eine Struktur, die entweder eine Resource oder eine Struktur enthält, die sich selbst enthält.[15]

Bei Annahme der Endlichkeit der Eingabe und deren Struktur (es gibt keine zirkulären Ausdrücke) folgt, daß eine oberste nicht-leere Schicht k existiert. Außerdem folgt aus der Konstruktion, daß die Schichten, die durch die Markierung der Resourcen und Literale erzeugt werden, fortlaufend numeriert sind.

Beispiel zur Transformation einer Menge von geschichteten, flachen Statements in ein verschachteltes Statement. Die Eingabemenge sei

[14]Wenn r ein Reifikant ist und $[s,p,o]$ ein Statement, das r reifiziert, dann sind s,p und $o \in E$. Wenn c ein Container ist, dann sind die Elemente von R_c (Definition siehe oben) $\in E$. Bemerkung: unter Annahme der obigen Definition für eine Container-Repräsentation gilt: eine Menge flacher Statements, die einen n-stelligen Container repräsentiert, ist gleichzeitig eine Repräsentation von $(n-1)$, $(n-2)$...-stelligen Containern. Es sei jedoch angenommen, daß E alle Entitäten enthält, die Elemente des Containers mit der höchsten Stelligkeit sind.

[15]Dies kann ebenfalls eine Kette von Reifikationen und Containern sein. D.h., hier wird es als Fehler betrachtet, wenn ein Container einen Reifikanten enthält, der ein Statement reifiziert, das den Container enthält, etc.

{ [Jemand sagt r_1],
[r_1 rdf:subject XRDF],
[r_1 rdf:predicate benutzt],
[r_1 rdf:object l_1],
[r_1 rdf:type rdf:Statement],
[l_1 rdf:type rdf:Seq],
[l_1 rdf:_1 Triple],
[l_1 rdf:_2 Listen] }

Schicht 0:	{ Jemand, Triple, ... rdf:Seq, rdf:_1, rdf:_2 }
Schicht 1:	$\{l_1\}$
Schicht 2:	$\{r_1\}$

Hierauf kann die Transformation entlang der Schichten angewandt werden. Zunächst wird

{ [l_1 rdf:type rdf:Seq],
[l_1 rdf:_1 Triple],
[l_1 rdf:_2 Liste] }

transformiert zu

(Triple, Liste),

die Statements werden aus der initialen Menge entfernt und jedes Vorkommen von l_1 wird ersetzt durch (Triple, Liste). Dies führt zur nächsten Menge von (jetzt schon verschachtelter) Statements:

{ [Jemand sagt r_1],
[r_1 rdf:subject XRDF],
[r_1 rdf:predicate benutzt],
[r_1 rdf:object (Triple, Listen)],
[r_1 rdf:type rdf:Statement],
[(Triple, Listen) rdf:type rdf:Seq] }.

Weiter wird die Menge

{ [r_1 rdf:subject XRDF],
[r_1 rdf:predicate benutzt],
[r_1 rdf:object (Triple, Listen)],
[r_1 rdf:type rdf:Statement] }

transformiert zu

[XRDF benutzt (Triple, Listen)].

Weiter wird in

[Jemand sagt r_1]

r_1 ersetzt, so das die minimale Hülle

{[Jemand sagt [XRDF benutzt (Triple, Listen)]]

[(Triple, Listen) rdf:type rdf:Seq] }.
resultiert.

Diese Prozedur wird durch folgenden Algorithmus erfaßt. Der Algorithmus bestimmt ein intentionsäquivalentes XRDF-Datenmodell C^E aus einer Menge C^R flacher Statements und einer Menge von Schichten (engl. stratum) $s[k]$.

Algorithm *Nest* (**In:** C^R, **In:** stratum[], **In:** k)
For $s = 1$ to k **do**
  For all Resourcen r in $stratum[s]$ **do**
    if r ein Reifikant ist **then**
      Entferne aus C^R die vier Statements der Reifikation
        die r als Reifikanten haben.
      Ersetze alle r, die in Ausdrücken von C^R vorkommen, durch das Statement, das r reifiziert.
    if r ein Container ist **then**
      Entferne aus C^R alle Statements der Form $[r, \_i, r_i]$ und
        erstelle eine Liste R_r der Resourcen r_i.
      Ersetze r, die in Ausdrücken von C^R vorkommen, durch die Liste R_r.
Set $C^E = C^R$.
Das Theorem folgt direkt aus der Konstruktion. ⊟

Neben diesem Beweis zeigt der RDF-to-XRDF-Converter [146] am praktischen Beispiel, wie ein RDF-Dokument in ein XRDF-Dokument transformiert werden kann.

Der Zusammenhang von RDF- und XRDF-Datenmodell basiert auf den beiden in Abschnitt 7.1 erläuterten Annahmen. Die hier vorgestellten Theoreme und Beweise erfordern die Akzeptanz der den Annahmen zugrundeliegenden Interpretation des RDF-Datenmodells. Selbst, wenn die Annahmen nicht akzeptiert werden, kann das hier vorgestellte XRDF-Datenmodell als eine Alternative zum RDF-Datenmodell gesehen werden. Das XRDF-Datenmodell ermöglicht es, Ausdrücke durch Verschachtelung und den Einsatz von Listen darzustellen. Dies kann als direktere Art der Darstellung von komplexen Ausdrücken betrachtet werden. Zur Illustration werden im nächsten Abschnitt zwei alternative Repräsentationen, (1) eine graphische und (2) eine XML-Notation vorgeschlagen, die wegen der direkteren Darstellung der Reifikation bzw. von Containern gegenüber den entsprechenden RDF-Repräsentationen als vorteilhaft gesehen werden können.

7.4 Repräsentation von XRDF-Beschreibungen

Im folgenden wird die graphische und die XML-Notation von XRDF vorgestellt. Beide Notationen sind bijektiv in das XRDF-Datenmodell abbildbar.

7.4.1 Graphische Notation

Die graphische Notation zur Repräsentation von XRDF-Beschreibungen besteht aus einem Oval (zur Repräsentation von Resourcen) und gerichteten markierten Kanten (zur Repräsentation von Relationen). Jedes Oval, das eine Resource vom Typ Atom repräsentiert, muß einen String enthalten, der aus dem Alphabet A gebildet werden kann (entspricht PCDATA in wohlgeformten XML). Jedes eingebettete Oval wird mit einer Nummer versehen, die innerhalb des umgebenen Ovals eindeutig ist. Nummern werden ausgelassen, wenn die Numerierung eindeutig durch einen Algorithmus angegeben werden kann (d.h. für Resourcen in Statements, für die die Ordinal-Zahl aus der Richtung der Kante ermittelt werden kann und für Resourcen in Listen, die nur ein Element enthalten.) Eine eindeutige Transformation eines XRDF-Graphen in verschachtelte Triple und vice versa ist gewährleistet (siehe Abschnitt 7.5). Folgende Abbildungen zeigen Beispiele der Verwendung der graphischen Notation.

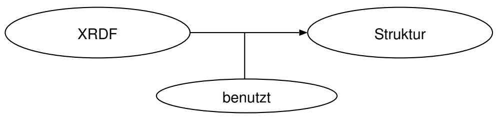

Abbildung 7.4: Repräsentation eines Statements

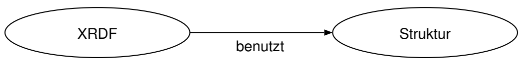

Abbildung 7.5: Das Statement aus Abbildung 7.4 unter Vernachlässigung der Tatsache, daß ein Prädikat ebenfalls eine Resource ist (dies ist die übliche Vereinfachung in RDF).

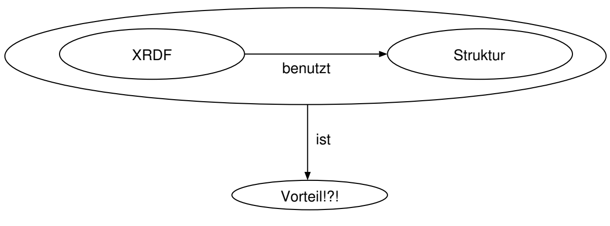

Abbildung 7.6: Repräsentation eines Statements über ein Statement

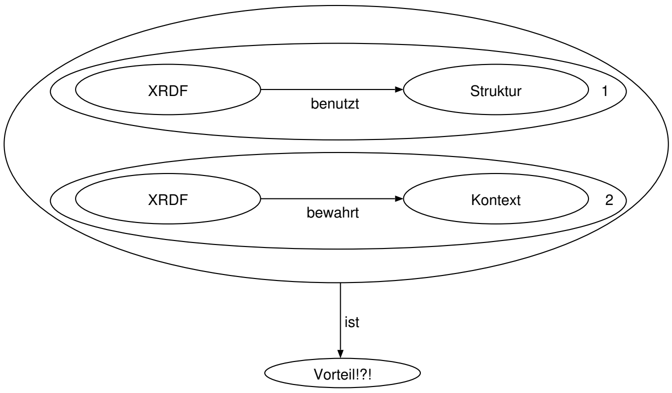

Abbildung 7.7: Repräsentation eines Statements über eine Liste von Statements

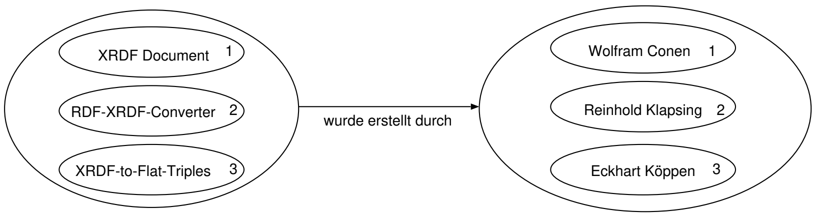

Abbildung 7.8: Repräsentation von Listen an Subjekt- und Objekt-Position

Auf eine Darstellung dieser Beispiele durch RDF-Graphen wird hier verzichtet. Es sollte jedoch deutlich geworden sein, daß bereits die Darstellung von Beschreibungen mittlerer Komplexität durch XRDF-Graphen wesentlich übersichtlicher ist, als die intentionsäquivaltente Darstellung durch RDF-Graphen. Dies gilt auch für die entsprechende Repräsentation durch verschachtelte Triple im Vergleich zu flachen Triplen.

7.4.2 XML-Notation

XRDF-Beschreibungen können in XML-Notation aufgeschrieben werden. Im Gegensatz zum RDF-Datenmodell kann für das XRDF-Datenmodell eine XML-DTD angegeben werden, so daß XRDF-Dokumente validiert werden können und beliebige XML-Werkzeuge zur Verarbeitung von XRDF-Dokumenten verwendet werden können.

```
<!ELEMENT statement (subject, predicate, object)>
<!ELEMENT list      (statement | atom | list)+>
<!ELEMENT atom      (#PCDATA)>
<!ELEMENT subject   (atom|statement|list)>
<!ELEMENT predicate (atom|statement|list)>
<!ELEMENT object    (atom|statement|list)>
```

Ein Algorithmus zur Transformation eines XML-Dokuments (das gegen die oben angegebene DTD validierbar ist) in ein XRDF-Datenmodell läßt sich unmittelbar angeben. Abschnitt 7.5.1 gibt ein entsprechendes XSLT-Stylesheets an. Der in Abschnitt 7.3 gezeigte Algorithmus *Flatten* kann zur Transformation in ein RDF-Datenmodell genutzt werden. Davon ausgehend kann eine Repräsentation in RDF-Syntax (zumindest eine Repräsentation bei der jedes Statement in einer `Description` resultiert) angegeben werden.

Beispiel: Das Statement [([XRDF, benutzt, Struktur], [XRDF, bewahrt, Kontext]), ist, Vorteil!?!], (vergleiche Abbildung 7.7), kann folgendermaßen linearisiert werden:

```
<statement>

  <subject>
    <list>

      <statement>
        <subject><atom>XRDF</atom></subject>
        <predicate><atom>benutzt</atom></predicate>
        <object><atom>Struktur</atom></object>
      </statement>

      <statement>
        <subject><atom>XRDF</atom></subject>
        <predicate><atom>bewahrt</atom></predicate>
        <object><atom>Kontext</atom></object>
      </statement>

    </list>
  </subject>

  <predicate><atom>ist</atom></predicate>

  <object><atom>Vorteil!?!</atom></object>

</statement>
```

Im folgenden Abschnitt wird ein XSLT-Stylesheet gezeigt, das zur Transformation einer derartigen XML-Repräsentation in ein XRDF-Datenmodell (verschachtelte Triple) genutzt werden kann. Eine prozedurale Beschreibung zur Transformation eines XRDF-Datenmodells in XRDF-Syntax wird ebenfalls gezeit. Außerdem wird eine Beschriebung der Transformation eines XRDF-Graphen in XRDF-Syntax angegeben.

7.5 Transformationen

7.5.1 XRDF-Syntax $\rightarrow$ XRDF-Datenmodell

Das folgende XSLT-Stylesheet kann zur Transformation einer XRDF-Beschreibung in eine intentionsäquivalente Menge von verschachtelten Triplen genutzt werden.

```
<!DOCTYPE xsl:stylesheet>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" />
<xsl:strip-space elements="*"/>

<xsl:template match="statement"
             name="statement">[<xsl:apply-templates />]</xsl:template>
<xsl:template match="atom"
             name="atom"><xsl:apply-templates /></xsl:template>
<xsl:template match="subject">[<xsl:apply-templates />], </xsl:template>
<xsl:template match="predicate"><xsl:apply-templates />, </xsl:template>
<xsl:template match="object">[<xsl:apply-templates />]</xsl:template>

<xsl:template match="list">
  <xsl:for-each select="*">
    <xsl:if test="name()='statement'">
       <xsl:call-template name="statement" />
    </xsl:if>
    <xsl:if test="name()='atom'">
       <xsl:call-template name="atom" />
    </xsl:if>
    <xsl:if test="position() &lt; last()">, </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

7.5.2 XRDF-Datenmodell $\rightarrow$ XRDF-Syntax

Ein verschachteltes Triple kann ohne Informationsverlust in XML-Notation transformiert werden. Zur Erinnerung: Jedes XRDF-Datenmodell ist ein Ausdruck über A^*, wobei drei Arten von Subausdrücken auftreten können: Atome, Triple und Listen. Die Transformation T einer XRDF-Beschreibung $\mathcal{R}$ (die in Form von verschachtelten Triplen vorliegt) in die entsprechende XML-Notation ist folgendermaßen definiert.

Sei e ein Element von $\mathcal{R}$. Die Transformation $T(e)$ sei dann definiert als:

- **If** $type(e) = atom$ **then**
  $T(e) =$ `<atom>` e `</atom>`.

- **If** $type(e) = list$ (d.h. $e = R_1, \ldots, R_n$) **then**
  $T(e) =$ `<list>` $T(R_1), \ldots, T(R_n)$ `</list>`.

- **If** $type(e) = statement$ (d.h. $e = [[R_s], r_p, [R_o]]$) **then**

  $T(e) =$
  <statement>
  <subject> $T(R_s)$ </subject>
  <predicate><atom> r_p </atom></predicate>
  <object> $T(R_o)$ </object>
  </statement>.

7.5.3 XRDF-Graph $\leftrightarrow$ XRDF-Syntax

Es folgt eine semiformale, prozedurale Beschreibung zur Transformation eines XRDF-Graphen in XRDF-Syntax.

1. Ein Oval, das zwei Ovale enthält, die durch eine gerichtete Kante verbunden sind, wird ein `statement`-Element.
2. Ein Oval, von dem eine gerichete Kante ausgeht, wird ein `subject`-Element.
3. Eine gerichtete Kante wird ein `predicate`-Element, dessen Inhalt die Markierung der Kante ist.
4. Ein Oval, auf das eine gerichtete Kante zeigt, wird ein `object`-Element.
5. Ein Oval, das nummerierte graphische Konstrukte enthält, wird ein `list`-Element. Die Ordinal-Zahl eingebetteter Konstrukte bestimmt die Position dieser in der Liste.
6. Ein Element von A^* (siehe formales XRDF-Datenmodell) wird ein `atom`-element.

Diese Prozedur ist invertierbar.

7.6 Überlegungen zur Semantik

Syntaktische und strukturelle Einfachheit ist eine Voraussetzung zur Vereinfachung der Notation von (Meta-)Daten. Schichtenansätze zur Modularisierung und zur Kapselung von Funktionen und Daten auf verschiedenen Abstraktionsstufen haben sich bereits bewährt (dies wird besonders deutlich bei Spezifikationen im Zusammenhang mit XML. Hier ist XML Grundlage und Ausgangsbasis weiterer Standards, wie z.B. XML-Namespaces, welche wiederum Grundlage weiterer Standards, z.B. XSLT, sind). Wird ein Schichtenansatz auf das hier vorgeschlagene Datenmodell bezogen, kann die Definition der XRDF-Syntax (die die strukturellen Primitive - atom, list, statement - bereitstellt) auf unterster Ebene eingeordnet werden. Auf diesem grundlegenden, strukturellen Modell können semantische Definitionen und Interpretationen (schichtenweise) aufbauen. Im folgenden wird die Nutzung der hier vorgestellten geschachtelten Strukturen für die semantische Interpretation von Beschreibungen diskutiert. Außerdem werden einige generelle Primitive vorgestellt, die genutzt werden können, um zu definieren, wie Prädikate auf Listenstrukturen angewandt und wie Prädikate in Beziehung zu einem zugrundeliegenden Wissenrepräsentations- und Inferenzmechanismus gesetzt werden können. Die Anzahl der Primitive ist bewußt klein gehalten. Es handelt sich hier nicht um eine normative Präsentation, sondern soll Grundlage für die Diskussion und Weiterentwicklung der semantischen Interpretation von XRDF-Beschreibungen sein.

7.6.1 Semantikpropagation entlang verschachtelter Strukturen

Zur Beschreibung von Semantik unter Verwendung von XRDF gibt es zunächst (in Abhängigkeit von präferierten Design-Entscheidungen und Anforderungen an ein Datenmodell bzw. eine Sprache zum Austausch von Semantik im Web) verschiedene Möglichkeiten der Interpretation von XRDF-Beschreibungen. Im folgenden wird eine Interpretation und dadurch möglich werdende Datenmodellierungsvereinfachungen vorgeschlagen. Der Interpretationsvorschlag basiert auf folgender Annahme: die semantische Interpretation von Ausdrücken hängt von deren Position in der umgebenden Struktur ab, d.h. die Semantik wird ausgehend vom äußersten Teil zum innersten Teil eines verschachtelten Ausdrucks interpretiert. Zur Demonstration zeigt das folgende Beispiel ein einfaches Wahrheitsprädikat.

Beispiel:

[[Himmel hat_Farbe grün] hasTruthValue FALSE]]

kann in flachen Triplen folgendermaßen ausgedrückt werden

[r type Statement]

[r subject Himmel]
[r predicate hat_Farbe]
[r object grün]
[r hasTruthValue FALSE]

Die folgende Transformation und darauf angewandte Zwänge geben den flachen Triplen Bedeutung in einer FOL-Repräsentation:

Transformation: Transformiere jedes Triple [s,p,o] in eine Instanz des Prädikats triple(s,p,o).

Constraints :

(1) reifies(R,S,P,O) ←
triple(R,type,statement) ∧
triple(R,subject,S) ∧
triple(R,predicate,P) ∧
triple(R,object,O).

(2) falsified_resource(R) ←
statement_known_as_true(R,hasTruthValue,FALSE).

(3) statement_known_as_false(S,P,O) ←
triple(S,P,O) ∧
reifies(R,S,P,O) ∧
falsified_resource(R).

(4) statement_known_as_true(S,P,O) ←
triple(S,P,O) ∧
not(statement_known_as_false(S,P,O)).

Auf diesen statement_...-Prädikaten können weitere Inferenzen aufbauen. Werden die Constraints auf das Beispiel angewandt, führt dies zur Falsifikation des Triples [Himmel hat_Farbe grün]. Die Falsifikation von falsifizierten Statements ist ebenfalls möglich. Dazu wird der Wahrheitswert eines verschachtelten Statements vom äußersten Teil eines (Sub-)Ausdrucks zum innersten Teil eines (Sub-)Ausdrucks propagiert. Dieses Prinzip kann ebenfalls genutzt werden, um komplexere Semantik zu definieren. Für Ausdrücke wie z.B.

[Reinhold glaubt
[Wolfram sagt
[[Himmel hat_Farbe grün] hasTruthValue FALSE]]]

ist dies notwendig, um einem solchen Ausdruck eine klar definierte Bedeutung zuzuweisen.

Die Bedeutung von eingebetteten Ausdrücken hängt hierbei vom "semantischen" Kontext ab. Äußere Prädikate haben Einfluß auf die Interpretation von inneren Prädikaten.

```
Geltungsbereich 3: (glaubt Reinhold
Geltungsbereich 2:      (sagt Wolfram
Geltungsbereich 1:            (hasTruthValue FALSE
Geltungsbereich 0:                   (Himmel hat_Farbe grün))))
```

Die Interpretation dieser Information hängt von der formalen Definition der Semantik der Prädikate und deren Interaktion ab. Auf das obige Beispiel bezogen ist die korrekte semantische Interpretation sicherzustellen, d.h. genau festzulegen, was die Bedeutung eines elementaren Statements ist, von dem jemand glaubt, daß jemand anderes sagt, die Negation dieses Statements gelte. Unter einem elementaren Statement sei ein Statement verstanden, welches ein Prädikat verwendet, daß nicht den Wahrheitswert des Subjekts oder Objekts beeinflußt (wie z.B. das Prädikat *hat_Farbe* im obigen Beispiel). Die Interpretation eines Statements, das geglaubt wird, sollte anders behandelt werden (können), als ein Statement das als Fakt (d.h. gesichertes Wissen) gilt.

Aus der Intentionsäquivalenz von XRDF-Datenmodell und RDF-Datenmodell folgt, daß auch für RDF-Beschreibungen die Bedeutung eines Statements von dessen Position (und daraus resultierenden Interpretationen) abhängt. Innerhalb einer RDF-Beschreibung wird die "Position" eines Statement durch dessen Vorkommen in Reifikationen und Listen festgelegt. Beispielsweise erfaßt die folgende Menge flacher Triple den obigen XRDF-Ausdruck:

[Reinhold glaubt r_3]

[r_3 subject Wolfram]
[r_3 predicate sagt]
[r_3 object r_2]
[r_3 type Statement]

[r_2 subject r_1]
[r_2 predicate hasTruthValue]
[r_2 object FALSE]
[r_2 type Statement]

[r_1 subject Himmel]
[r_1 predicate hat_Frabe]
[r_1 object grün]
[r_1 type Statement]

Anmerkung: es ist nicht notwendig (bzw. nutzlos oder sogar sinnverändernd) Sub-Ausdrücke wie z.B. [Wolfram sagt r_2] in die Menge flacher Triple aufzunehmen. Wäre dies der Fall, würde eine dem Beispiel intentionsäquivalente XRDF-Beschreibung zwei Statements enthalten:

[Reinhold glaubt [Wolfram sagt [[Himmel hat_Farbe grün] hasTruthValue FALSE]]]

[Wolfram sagt [[Himmel hat_Farbe grün] hasTruthValue FALSE]]

hat also eine andere Bedeutung, als wenn nur das erste Statement vorhanden ist, da in diesem Fall [Wolfram ...] zu einem faktischen Statement (ein Statement, das in einer Wissenbasis als Fakt gilt) wird. Anhand des Beispiels wird ebenfalls deutlich, daß syntaktisch identische Sub-Ausdrücke im Kontext unterschiedlicher Statements auch unterschiedliche Bedeutung haben können. Das bedeutet, daß die Semantik eines Statements nur im Zusammenhang mit dem Kontext, in dem ein Statement verwendet wird, interpretiert werden sollte. Dadurch wird auch klar, daß es nicht notwendig ist, jedem Vorkommen von syntaktisch identischen Sub-Ausdrücken eine eindeutige Identität zu geben, da die Bedeutung jedes Vorkommens vom Kontext abhängt (wobei der Kontext durch die Position des Vorkommens als Sub-Ausdruck innerhalb verschiedener umgebender Ausdrücke bestimmt wird).

7.6.2 Kurzschreibweisen durch strukturelle Transformation

Strukturelle Transformationen können genutzt werden, um die XRDF-Syntax um Kurzschreibweisen für XRDF-Beschreibungen zu erweitern.

Statement (bzw. Triple) und *List* sind die strukturellen Primitive des XRDF-Datenmodells. Im folgenden wird ein Mechanismus vorgeschlagen, der dazu genutzt werden kann, die semantische Interpretation von Resourcen und deren Beziehungen durch Typen von strukturellen Transformationen, die Prädikaten zugewiesen werden können, zu beeinflussen.

Die Semantik eines Prädikats wird letztlich durch strukturelle Transformtionen festgelegt, d.h. ein Triple wird in Abhängigkeit vom Prädikat in eine andere (Wissens-)Repräsentation transformiert. Semantisch interpretiert bezeichnet der Name eines Prädikats eine Relation zwischen Subjekt und Objekt. Bezogen auf verschachtelte Triple, inklusive Listen die an Subjekt- und Objekt-Position erlaubt sind, seien die folgenden vier Typ-Relationen zu unterscheiden:

1. *(1:1)* – das Prädikat bezeichnet eine Relation zwischen einem Subjekt und einem Objekt.

2. *(n:m)* – das Prädikat bezeichnet eine $n \times m$-Relation. Jedes Element des Subjekts steht in Beziehung zu jedem Element des Objekt.

3. *(1:n)* – das Prädikat bezeichnet $1 \times m$-Relation. Das Subjekt steht mit jedem Element des Objekts in Beziehung.

4. *(n:1)* – das Prädikat bezeichnet $n \times 1$-Relation. Jedes Element des Subjekts steht mit dem Objekt in Beziehung.

Weiter sei durch das Prädikat `representedBy` ausdrückbar, daß ein Objekt ein Subjekt repräsentiert (d.h. das Objekt ist ein Name bzw. Repräsentant für ein Subjekt). Ein solches Objekt kann als Objekt oder Subjekt in Relationen verwendet werden. Für Prädikate, die in einer Relation mit Repräsentanten verwendet werden, kann festgelegt werden, ob über den Repräsentant oder die repräsentierte Resource eine Aussage gemacht wird (Dereferenzierungs-Mechanismus). Um dies zu unterscheiden wird der $(x : y)$-notation[16] ein d (für direkt) oder ein i (für indirekt[17]) hinzugefügt.

Beispiel: Gegeben seien folgende Statements

[(Triple Listen) representedBy Strukturen]
[XRDF benutzt Strukturen]

Eine Interpretation ist, daß `XRDF` die Resource `Strukturen` `benutzt`, d.h. `benutzt` bezeichnet eine $(d1 : d1)$-Relation. D.h.:

[XRDF benutzt Strukturen]

wird transformiert zu

[XRDF benutzt Strukturen]

Eine andere Interpretation ist, daß `XRDF` die Gruppe von Resourcen `benutzt`, die durch `Strukturen` repräsentiert wird, d.h. `benutzt` bezeichnet eine $(d1 : i1)$-Relation (wenn das Objekt ein Repräsentant ist).

[XRDF benutzt Strukturen]

wird transformiert in

[16] Ein i wird wie ein d behandelt, wenn das Subject bzw. Object kein Repräsentant ist. Wenn weder d noch i angegeben sind, wird d angenommen. 1 ist ein Sonderfall von n. Der Default-Typ eines Prädikats ist $1 : 1$.

[17] Die Indirektion ist rekursiv anzuwenden.

[XRDF benutzt (Triple Listen)]

Eine weitere Interpretation ist, daß **XRDF** jede der repräsentierten Resourcen (vollkommen unabhängig voneinander) **benutzt**, d.h. **benutzt** bezeichnet eine $(d1 : im)$-Relation.

[XRDF benutzt Strukturen]

wird transformiert in

[XRDF benutzt Triple]
[XRDF benutzt Listen]

Eine weitere Primitive ist die *nichtdeterministische Auswahl*. Für ein Prädikat kann definiert werden, daß ein Element der Subjekt- und/oder der Objekt-Sequenz nichtdeterministisch ausgewählt wird, um dieses mit einer Resource in Beziehung zu bringen[18]. Die Nichtdeterministische-Auswahl wird durch ein c statt n, m oder 1 gekennzeichnet.

Der Typ eines Prädikats wird durch das Prädikate `ptype` definiert.

Beispiel:

[XRDF benutzt Strukturen]
[benutzt ptype d1:in]

Das Beispiel zeigt, wie für das Prädikat *benutzt* der *Typ d1:in* zur strukturellen Transformations angegeben wird. Erst nach struktureller Transformation wird eine Beschreibung semantisch interpretiert. Der Typ der strukturellen Transformation wird durch Attributierung des Prädikats ptype spezifiziert.

Durch die Attributierung des Prädikats `isDefinedAs` kann die Semantik eines Prädikats durch Angabe von Semantik in der Syntax eines Wirtsformalismus spezifiziert werden (vgl. auch die Arbeiten [43, 41]).

7.6.3 Semantik-Templates

Das Prädikat `isDefinedAs` dient zur Definition der Semantik von Prädikaten mittels Ausdrücken einer zugrundeliegenden Wissensrepräsentation (Wirtsformalismus), in die ein XRDF-Datenmodell transformiert wird. Die gezeigten Beispiele erfolgen unter Verwendung von Prolog, der Ansatz ist jedoch ebenfalls mit anderen Wissensrepräsentationssystemen realisierbar.

[18] Anmerkung: Die nichtdeterministische Auswahl ermöglicht es, ein Äquivalent zur Existenzquantifikation (auf einer relationalen Ebene) auszudrücken. $\forall$-Quantifikation kann durch Expansion ausgedrückt werden.

Das folgende Beispiel zeigt die Definition eines Prädikats, das zur Interpretation von transitiven Relationen genutzt werden kann. Um der Definition eines Prädikates Bedeutung zu geben, ist es notwendig, eine XRDF-Beschreibung in eine Wissensbasis (eines Wirtsformalismus) zu transformieren. Es sei angenommen, daß ein (Logik-)Prädikat *statement* existiert, das alle Statements, die rekursiv aus einer XRDF-Beschreibung abgeleitet werden können, erfaßt.

```
[ [ path ]
  [ isDefinedAs ]
  [ path(X,Y) :- statement(X,path,Y).
    path(X,Z) :- statement(X,path,Y), path(Y,Z). ]
]
```

Das so definierte Prädikat *path* dient der Inferenz von transitiven Beziehungen, die durch Statements, die das Prädikat `path` enthalten, ausgedrückt werden.

Gegeben seien die Fakten:

```
[ A  path B ]
[ B  path C ]
```

dann ist *path(A,C)* eine Konsequenz aus der wirtsformalismusabhängigen Definition des Prädikats `path`.

7.6.4 XRDF-Beschreibungen und Wissensbasen

Wird Prolog als Wissensrepräsentationssystem verwendet, hat die Reihenfolge, in der Wissen (Fakten und Regeln) angegeben wird, eine Bedeutung. Dies passt zu der dem XRDF-Datenmodell inhärenten Ordnung.

Die Transformation eines XRDF-Dokuments in eine Wissenrepräsentation verläuft folgendermaßen:

1. Ein Modell, das aus Ausdrücken über A^* besteht, wird gemäß dem in Abschnitt 7.3 angegeben Algorithmus expandiert (in flache Triple transformiert).

2. Das expandierte Modell wird in die Wissensbasis eingetragen (engl. asserted). Dies sind die grundlegenden Fakten der Wissensbasis.

3. Die Regeln, die durch das Prädikat `isDefinedAs` für Prädikate definiert sind, werden der Wissenbasis hinzugefügt.

Das detaillierte Vorgehen in den Schritten 2 und 3 hängt vom gewählten Wissensrepräsentationssystem ab. Die grundlegenden strukturellen Transformationen, die im Abschnitt 7.5 beschrieben werden, können in einem vorverarbeitenden Schritt angewendet oder direkt durch Prädikat-Definitionen repräsentiert (und erst zur Laufzeit angewendet) werden.

Der XRDF-Parser verwendet einen generischen vorverarbeitenden Schritt zur Berechnung der Expansionen und bestimmt für alle Sub-Ausdrücke Adressen für zukünftige Referenzen.

In diesem Abschnitt wurden grundlegende Primitive und Mechanismen zur Erfassung von Semantik mittels XRDF-Beschreibungen vorgestellt. Weitere Primitive und Mechanismen zur Erfassung von Schema-Level-Semantik können hierauf aufbauen.

7.7 Verwandte Arbeiten

Es folgt eine kurze Diskussion von Ansätzen, die (wie XRDF) eine alternative auf Markup basierende Syntax vorgeschlagen.

Tim Berners-Lee schlägt in [12] die *Strawman Unstriped Syntax* – eine zu RDF alternative XML-Syntax – vor. Wesentliches Merkmal gegenüber der RDF-Syntax ist, daß die Kanten eines RDF-Graphen durch XML-Elemente repräsentiert werden und Knoten implizit angenommen werden (d.h. modelltheoretisch wird parseType="resource" als Default angenommen). Designziele sind: (1) RDF-Syntax soll innerhalb von beliebigem XML-Markup verwendet werden können und umgekehrt XML-Markup in RDF-Syntax. (2) Es soll möglich sein Dokumente zu erstellen, die effizient Information ausdrücken und die es RDF-Parsern ermöglichen, daraus RDF-Graphen zu generieren, ohne daß es notwendig ist, ein Namensraum-Schema zu lesen. (3) Optionale Erweiterungen sollen erlaubt sein.

Sergey Melnik schlägt in [121] einen Ansatz vor, der auf den Ideen aufbaut, die in [12] gezeigt werden. Eine Referenzimplementierung ist verfügbar. Weiter schlägt Melnik in [120] vor, es zu ermöglichen, für jedes ("legacy") XML-Dokument ein RDF-Modell angeben zu können.

Beiden Ansätzen ist gemein, daß (unstripped) Syntax verwendet wird, die einen RDF-Graph *nicht explizit* beschreibt. Es wird jedoch angenommen, daß aus einer Beschreibung, inklusive impliziter Konstrukte, ein RDF-Graph erstellt werden kann. Der XRDF-Ansatz ist komplementär zu den oben erwähnten

Ansätzen, da ein RDF-Graph durch XRDF-Syntax *explizit* beschrieben wird. Die beiden oben genannten Ansätze leiden unter den Restriktionen des RDF-Modells und den Unklarheiten, die aus der Verwendung von RDF-Schemata resultieren (z.B. keine formale Semantik, kein formalisierter Erweiterrungsmechanismus). Jedoch haben die Ansätze, wie auch der XRDF-Ansatz den Vorteil, daß Standard-XML-Syntax verwendet wird.

Drew McDermott et al. schlagen in [118] einen weiteren Ansatz vor, der unter anderem auf den hier gezeigten Ideen aufbaut. Dieser Ansatz kann genutzt werden, um logische Ausdrücke in RDF/DAML zu repräsentieren.

7.8 Diskussion

Das hier vorgestellte XRDF-Datenmodell nutzt verschachtelte Statements und Listen als Alternativen zur Reifikation und Containern, wie sie im RDF-Datenmodell verwendet werden. Eine XRDF-Beschreibung kann (1) als verschachtelter Graph, (2) durch verschachtelte Triple oder (3) in validierbarer XML-Syntax dargestellt werden. Das XRDF-Datenmodell bietet eine strukturelle Ebene, die klar von einer semantischen Interpretations-Ebene abgegrenzt ist. Aufbauend auf der strukturellen Ebene kann Semantik (für Prädikate) definiert werden, ohne daß syntaktische Auswertungen auf semantischer Eben notwendig sind.[19] Es wurde gezeigt, daß das XRDF-Datenmodell und RDF-Datenmodell ineinander transformierbar sind. Algorithmen für die entsprechenden Transformationen und zur Überprüfung der Wohlgeformtheit (d.h. es existieren keine Resourcen die sich selbst enthalten) werden gezeigt.

Im Folgenden werden einige Punkte diskutiert, die im Zusammenhang mit der RDF-Syntax und dem RDF-Datenmodell problematisch sind und die durch das hier vorgestellte XRDF-Datenmodell bzw. die dazugehörigen Repräsentationen vermieden werden.

In der RDF-Syntax wird das Attribut ID[20] zur Namensgebung für Resourcen verwendet. Obwohl die Möglichkeit besteht in der DTD der XRDF-Syntax ein

[19] "Die Referenzierung syntaktischer Elemente über rdf:...Konstrukte zur Strukturdefinition, wie sie im RDF-Datenmodell verwendet werden, sind nicht mehr notwendig. Diese Elemente werden bei der Transformation von RDF-Beschreibungen in XRDF-Beschreibungen eliminiert. Das verbleibende Prädikat rdf:type sollte umbenannt werden (z.B. in rdfs:type), um auch im RDF-Datenmodell die strukturelle Ebene von der Schema-Definitions-Ebene klar abzugrenzen.

[20] Die Art der Verwendung des XML-Attributs ID in der RDF-Syntax entspricht nicht der XML-Recommendation, da keine DTD für die RDF-Syntax angegeben wird, die ID als Attribut vom Typ ID definiert. Es ist nicht klar definiert, ob das Attribut ID gemäß der

ID-Attribut zu definieren, wird diese Möglichkeit verworfen. Ist der Name einer Resource zur Interpretation einer Beschreibung relevant, sollte dieser durch ein Triple definiert werden, so daß die Beziehung zwischen Resource und Resourcen-Name Bestandteil des Beschreibungsmodells ist. Hierzu kann ein Prädikat (z.b. `hasName`) definiert werden, durch welches diese Relation ausgedrückt wird. Die Referenzierung einer Resource über den Namen ist dann auf Modellinterpretationsebene möglich, womit eine klare Trennung zur Syntax-Ebene erreicht ist. Dies vermeidet die Notwendigkeit der Benutzung des Fragment-Identifiers, um damit eine Resource über ihren Namen zu referenzieren.[21]

In der RDF-Syntax werden XML-Namensräume zur Referenzierung von Schemata genutzt, welche die Semantik für Resourcen definieren. Die RDF-M+S-Spezifikation definiert jedoch, daß für jede Resource ein URI (inklusive Fragment-Identifier) angegeben werden muß. Im RDF-Datenmodell, bzw. der Triple-Notation, sind keine Primitive vorgesehen, um einen Namensraum-Namen, ein Namensraum-Präfix und den lokalen Resourcen-Namen (der entweder ein XML-Element-Name oder eine XML-Attribut-Name ist) zu erfassen. Auch die Bestimmung des Namensraum-Namen und des lokalen Resourcen-Namen aus einen URI ist nicht formal definiert.[22] Dies unterstützt zusätzlich das obige Argument, daß ein Resourcen-Name explizit auf semantischer Ebene (als Bestandteil eines Beschreibungsmodells) ausdrückbar sein sollte.

Außerdem ist für die RDF-Syntax nicht klar definiert, in welchen Fällen ein Namensraum-Präfix angegeben werden muß. Die in der RDF-M+S-Spezifikation gezeigten Beispiele wenden die Namensraum-Präfixe auf Attribute uneinheitlich an. Gemäß der XML-Namespace-Spezifikation sind Attribute ohne Namensraum-Präfix nicht äquivalent zu Attributen mit Namensraum-Präfix. Darüberhinaus ist, gemäß der EBNF-Notation für die RDF-Syntax, die Verwendung eines Namensraum-Präfixes für Attribute *nicht erlaubt*. Das bedeutet, daß alle RDF-Attribute faktisch nicht qualifiziert sind. Der XRDF-Ansatz dagegen

XML-Recommendation gemeint ist, oder ein spezielles RDF-ID mit anderer Bedeutung. Dies ist eine der Ungenauigkeiten und eines der Probleme, welche aus der Tatasche resultieren, das es es nicht möglich ist, eine generelle DTD für die RDF-Syntax anzugeben.

[21] Die Benutzung eines Fragment-Identifiers ist im Zusammenhang von RDF problematisch, da die Bedeutung eines Fragment-Identifiers vom MIME-Typ einer Web-Resource abhängt (siehe auch [17]). Problematisch ist dies, da (1) derzeit kein MIME-Typ für RDF-Dokumente existiert und (2) die MIME-Typ-Verhandlung (engl. MIME type negotiation) Bestandteil des HTTP-Protokolls ist und somit nicht Bestandteil einer RDF-Beschreibung. Für eine RDF-Beschreibung ist nicht klar, welche Entität durch einen Fragment-Identifier referenziert wird.

[22] Siehe auch die Diskussion in der www-rdf-interest Mailing-Liste, die bei http://lists.w3.org/Archives/Public/www-rdf-interest/2000Jul/0037.html beginnt.

erzwingt die Verwendung von XML-Namensräumen nicht, schließt diese jedoch auch nicht aus. XML-Namesräume können komplementär zum XRDF-Ansatz verwendet werden, um Markup, das durch mehrere DTDs definiert wird, mit der durch den XRDF-Ansatz definierten XML-Syntax in einem XML-Dokument zu verwenden.

Das Beispiel der XML-Namesräume zeigt, daß es Information gibt, die durch die RDF-Syntax erfaßt wird, aber nicht durch das RDF-Datenmodell. Weitere Beispiele sind das `xml:lang`-Attribut und das `aboutEachPrefix`-Attribut. Die Information, die durch diese Attribute ausgedrückt wird, hat keine Entsprechung in der Triple-Notation. Dies führt zu der generellen Frage, inwieweit die verschiedenen Repräsentationen von RDF äquivalent sind. Die RDF-M+S-Spezifikation zeigt vier Repräsentationen: (1) das Mengenmodell[23] bzw. Datenmodell, (2) die graphische Repräsentation, (3) die XML-Syntax (RDF-Syntax), (4) die Triple-Notation (eine Untermenge von (1) wird dem "Datenmodell" oft gleichgesetzt wird). Ein in diesem Zusammenhang interessantes Beispiel sind anonyme Resourcen. Die RDF-M+S-Spezifikation macht keine klaren Aussagen über die Darstellung von anonymen Resourcen in den verschiedenen Repräsentationen.

Die RDF-Syntax erlaubt syntaktische Konstrukte, mittels derer Aussagen über Resourcen gemacht werden, für die kein Name angegeben wird bzw. aufgrund der Syntax-Grammatik nicht angegeben werden kann (obwohl es in einigen Fällen wünschenswert wäre, einen Namen angeben zu können). Im Mengenmodell und in der Triple-Notation sind jedoch keine anonymen Resourcen erlaubt. Für das Mengenmodell gilt: "Gegegen ist eine Menge von Resourcen". Ein Alphabet, daß es ermöglicht zu bestimmen, welche Entitäten zur Menge *Resources* (siehe auch Abschnitt 3.2) gehören, ist nicht gegeben. Im folgenden sei deshalb angenommen, daß ein Alphabet notwendig ist, um die RDF-M+S-Spezifikation diesbezüglich zu vervollständigen. Dann gilt, daß "anonyme" Resourcen nicht Teil einer RDF-Beschreibung sein können, da jedes Atom r ein Element des Alphabets ist. Auf der mengentheoretischen Ebene ist dieser "Name" alles ist, was über r bekannt ist (in der Essenz ist r der Name, und sonst nichts). Eine Möglichkeit wäre, einen *Namen* (der durch ein Prädikat annotiert würde) zusätzlich zu einem Bezeichner, der explizit angegeben wird oder aus der Dokument-Struktur folgt, für eine Resource einzuführen. Dann wäre eine "anonyme" Resource (auf Semantik-Ebene) eine Resource ohne *Namen*. Dies ist für eine XRDF-Beschreibung einfach möglich, jedoch ist die Notwenigkeit an-

[23]Das Mengenmodell wird in der RDF-M+S-Spezifikation als formales Modell (enlg. formal modell) bezeichnet.

onymer Resourcen in XRDF-Beschreibungen im Allgemeinen nicht gegeben, da Statements über Statements und Gruppierungen auch ohne anonyme Resourcen ausgedrückt werden können. Im Gegensatz zum RDF-Ansatz sind, bezogen auf den XRDF-Ansatz, alle Repräsentationen gleich mächtig, d.h. das XRDF-Datenmodell, die XRDF-Syntax und der XRDF-Graph drücken die dem Ansatz zugrundeliegenden Konzepte äquivalent aus.

Die Definition von Identität oder Existenz von Resourcen führt auch zu der Frage, ob in einer RDF-Beschreibung mehrere Statements mit gleichem Subjekt/Prädikat/Objekt erlaubt sein sollten. Es erscheint sinnvoll, dies durch Schemata (auf Semantik-Ebene) zu definieren, wie auch die Semantik einer Anwendung. Anwendungen können dann das gleiche Statement mehrfach, aber in verschiedenen Kontexten, interpretieren. Es hängt dann von der Interpretation auf Schema-Ebene ab, ob identische Resourcen (zur Erinnerung: Statements, Atome oder Listen) als äquivalent (bei Vernachlässigung des Kontexts) betrachtet werden. Die verschachtelte Triple-Notation erfaßt den (strukturellen) Kontext, so daß kontextsensitive Anwendungen möglich sind.

Zur Definition der Beziehungen zwischen Sammlungen von Resourcen werden Konzepte zur Beschreibung von Gruppierungen bzw. Sammlungen benötigt. Als Gruppierungskonzepte sieht die RDF-M+S-Spezifikation die Klassen `Bag`, `Seq` und `Alt` vor. Die Semantik der verschiedenen Gruppierungskonzepte sollte auf Schema-Ebene ausgedrückt werden. Da aber Gruppierungen bzw. Listen ein mächtiges und bewährtes Konzept sind, ist es sinnvoll, entsprechende Primitive auf Syntax-Ebene zu unterstützen. Gruppierungen bzw. Listen sind grundlegende Ausdrucksprimitive von XML (ein XML-Dokument kann als Baum von Listen, bzw. als Verkettung von Listen betrachtet werden). Es ist deshalb nicht sinnvoll, Information, die mit diesen grundlegenden Primitive ausgedrückt werden kann, im Datenmodell nicht zu berücksichtigen. XRDF nutzt die grundlegenden Strukturierungseigenschaften von XML. Das XRDF-Datenmodell und die graphische Repräsentation stellen Primitive bereit, mittels derer Listen repräsentiert werden können. An Subjekt- und Objekt-Position eines verschachtelten Triples sind Listen erlaubt. Im Gegensatz zur RDF-Syntax sind Prädikate wie `rdf:_1, rdf:_2, ...`, die zur Darstellung von Listen genutzt werden, nicht nötig. Dies ist von Vorteil, da die Darstellung durch RDF-Syntax diverse Nachteile hat: (1) Ein RDF-Parser muß die recht komplexe Struktur der Gruppierungskonzepte verarbeiten können. Aus dem syntaktischen XML-Konstrukt `rdf:li` sind die Prädikate `rdf:_1`, `rdf:_2` usw. zu bestimmen und aus der Triple-Menge einer RDF-Beschreibung die entsprechenden Gruppierungen zu schließen. (2) Die RDF-M+S-Spezifikation definiert nicht, wie bei Modifizierung von Gruppierungen zu verfahren ist. Es ist z.B. nicht klar definiert, welche Sequenz-Zahl `_x` bei Hinzufügen einer Resource zu einer Gruppierung vergeben werden soll/muß, wenn zuvor eine Resource mit der Sequenz-Zahl, z.B. `rdf:_(n/2)` (also die mittlere Resource), entfernt wurde. Auch ist z.B. nicht klar, ob bei Entfernen einer Resource die Sequenz-Zahlen aller nachfolgenden Resourcen umbenannt werden müssen.

Das RDF-Datenmodell erlaubt Literale nur an Objekt-Position. Weiter können nur Aussagen über Resourcen formuliert werden, die an Subjekt-Position stehen. Das bedeutet, daß es nicht möglich ist, über Literale eine direkte[24] Aussage zu machen. Das XRDF-Datenmodell dagegen erlaubt, Literale[25] an Subjekt-Position, über die auch Aussagen gemacht werden können.

Ein kritischer Aspekt bzgl. des XRDF-Datenmodells ist die Behandlung der Typisierung von Containern. Werden zwei Container verschiedenen Typs (z.B. ein Bag-Container und ein Seq-Container) in einer RDF-Beschreibung verwendet, die die gleichen Resourcen referenzieren (was im Zusammenhang von Container semantisch einem Enthaltensein entspricht) können diese Container bei Transformation in eine XRDF-Beschreibung nicht mehr unterschieden werden. Hierzu folgende Vorschläge: (1) Eine Lösung wäre, die Typisierungskonzepte für Container auf Schema-Ebene zu behandeln. Dies, zusammen mit der Verwendung von Listen-Namen (z.B. unter Verwendung des *representedBy*-Prädikats das in Abschnitt 7.6.2 beschrieben wurde), kann dann zur Typisierung genutzt werden, indem durch ein Statement der Typ und der Listen-Name in Relation gesetzt werden. (2) Eine weitere Lösung wäre, keine Typisierungskonzepte für Container anzubieten, sondern Container zunächst nur als Listen zu behandeln. Die Information, wie diese "Container-Listen" zu behandeln sind, kann in den Prädikat-Definitionen angegeben werden. Abhängig davon, welches Prädikat (durch die Formulierung eines Statements) auf eine Liste angewandt wird, wird diese z.B. entweder als Bag oder als Seq interpretiert (dies ist der konsequentere, prädikatzentrische Web).

Es gibt sicher noch weitere Aspekte des XRDF-Ansatzes, die diskutiert werden können. Die vorliegende Demonstration sollte jedoch ausreichen, um zu zeigen, daß die Verwendung von verschachtelten Statements zusammen mit Listen (von Listen, Atomen oder Statements) eine geeignete Repräsentation von Struktu-

[24] Dies wäre nur indirekt durch die Adressierung eines Literals durch einen URI z.B. einen XPath-Ausdruck möglich.

[25] Anmerkung: Die Assoziation/Zuordnung/Typisierung von PCDATA (sei es an Subjekt- oder Objekt-Position) mit einer Klasse ist durch eine Primitive auf der semantischen Ebene zu behandeln.

ren ist, die gegenüber den Beschreibungsprimitiven, die das RDF-Datenmodell bietet, vorteilhaft sein kann.

Zusammenfassend sollen folgende Aspekte hervorgehoben werden. Der hier vorgestellte XRDF-Ansatz bietet eine einfache und validierbare XML-Syntax und ein Datenmodell (das wesentliche strukturelle Eigenschaften der XML-Syntax nutzt) zur (Web-basierten) Repräsentation von Relationen zwischen (Web-)Resourcen. Das Datenmodell erlaubt Listen an Objekt *und Subjekt*-Position und erhält den (strukturellen) Kontext, in dem Resourcen verwendet werden. Es wird ein Schichten-Ansatz angewendet. Die unterste (syntaktischen) Ebene dient nicht der semantischen Interpretation einer Beschreibung. Auf höheren semantischen (Schema-)Ebenen können Primitive eingeführt werden, durch die verschiedene semantische Interpretationen der untersten Ebene möglich werden. Syntaktische Konstrukte sind klar von semantischen Primitiven getrennt. Auf Schema-Ebene können Primitive eingeführt werden, die zur Formulierung anwendungsbereichabhängiger Semantik einsetzbar sind. Die formale Definition von Semantik und ein Erweiterungsmeachnismus wird unterstützt. Die zur Verfügung gestellten Repräsentationsmöglichkeiten (Datenmodell, graphische Repräsentation und XML-Syntax) für XRDF-Beschreibungen sind in der Ausdrucksmächtigkeit äquivalent. XRDF ist ein zu RDF kompatibler Ansatz. Entwicklungsziel von XRDF ist es, möglichst viele Kritikpunkte, die die RDF-Community bzgl. RDF anbringt, zu vermeiden und insbesondere die Formulierung bzw. Modellierung von (semantischen) Beschreibungen zu erleichtern.

8 Schlußbetrachtungen

Die Fragestellung und das Ziel der vorliegenden Arbeit sind in Abschnitt 1 motiviert und formuliert. Abschnitt 1 enthält zudem eine Zusammenfassung der Ergebnisse dieser Arbeit, weshalb in diesem Kapitel in sehr konzentrierter Form auf die Beantwortung der Fragestellung und die Zielerreichung eingegangen wird. Abschließend wird erläutert, wie die Ergebnisse dieser Arbeit fortgeführt werden können und es wird ein Zusammenhang der Ergebnisse mit zukünftigen Entwicklungen hergestellt.

8.1 Fazit

Die Frage, *ob RDF semantische Interoperabilität ermöglicht und eine geeignete technologische Grundlage für das Semantic-Web ist* kann bedingt positiv beantwortet werden. Die zu erfüllende Bedingung ist eine Formalisierung der durch die RDF-Spezifikationen intendierten Semantik. In der vorliegenden Arbeit wird eine Formalisierung der RDF-Semantik in Prädikatenlogik erster Ordnung vorgeschlagen. Ausgehend von dieser formalen Grundlage wird eine Implementierung der RDF-Semantik in Prolog entwickelt und erläutert. Die RDF-Syntax ist nicht durch einen XML-Validator überprüfbar und es gibt keine eindeutige und standardisierte Abbildung der RDF-Syntax in das RDF-Datenmodell und vice versa. XRDF, ein in dieser Arbeit vorgestellter Ansatz, definiert ein Datenmodell und eine validierbare XML-Syntax zur Repräsentation von Beschreibungsmodellen. Das XRDF-Datenmodell ist in das RDF-Datenmodell abbildbar. Eine eindeutige Abbildung der XRDF-Syntax in das XRDF-Datenmodell (und damit auch in das RDF-Datenmodell) ist bei diesem Ansatz gegeben. Insofern ist XRDF zu RDF kompatibel.

Weitere Fragestellung der Arbeit ist, *ob sich RDF zur formalen Beschreibung der Inhalte, Strukturen und Zusammenhänge der Bestandteile einer Web-Anwendung einsetzen läßt und ob die in Abschnitt 1.1 aufgestellten Anforderungen an ein entsprechendes Beschreibungsmodell erfüllt werden.* Diese Frage kann, bei

Anwendung des im Rahmen dieser Arbeit entwickelten XWMF-Ansatzes, positiv beantwortet werden. Für eine Diskussion des Ansatzes und eine detaillierten Überprüfung der Anforderungserfüllung siehe Abschnitt 6.6.

8.2 Ausblick

Soll das Semantic-Web Wirklichkeit werden, ist der weit verbreitete Einsatz von Technologien notwendig, mit denen es möglich ist, Semantik Web-basiert auszutauschen. Insofern ist die Entwicklung des Semantic-Web eine soziotechnische Aufgabe, da die technologischen Voraussetzungen zum Web-basierten Austausch von Semantik geschaffen werden müssen und der weit verbreitete Einsatz entsprechender Technologie zu erreichen ist.

Derzeit gilt RDF als technologische Grundlage des Semantic-Web. Ein weit verbreiteter Einsatz von RDF ist bisher jedoch noch nicht erreicht (vgl. [59]). Der in dieser Arbeit vorgestellte XWMF-Ansatz ermöglicht die Beschreibung von Web-basierten Informationssystemen mittels RDF-Syntax, womit RDF-Daten bereits bei der Beschreibung und Modellierung einer Web-Anwendung erstellt werden und nicht erst nachträglich in einem zusätzlichen Arbeitsschritt. Insofern kann der Ansatz dazu beitragen, den Einsatz von RDF zu fördern.

Der Ansatz ist erweiterbar, da zusätzliche RDF-Vokabulare dem Beschreibungsrahmenwerk hinzugefügt werden können. Dies stellt die Einsetzbarkeit des Ansatzes für bisher nicht berücksichtige bzw. zukünftige Aufgaben sicher. Die Entwicklung von zusätzlichen RDF-Vokabularen ist zudem eine wünschenswerte Fortführung der vorliegenden Arbeit. Darüberhinaus läßt sich die praktikable Einsetzbarkeit durch die Entwicklung eines Repositories, das vordefinierte und generisch einsetzbare Templates (vgl. Simplexons 6.3.2) enthält, erhöhen. Weiter kann die Verbesserung der in dieser Arbeit vorgestellten bzw. die Entwicklung zusätzlicher (graphischer) Werkzeuge die Praktikabilität des XWMF-Ansatzes erhöhen.

Die in Abschnitt 4 gezeigte Formalisierung der RDF-Semantik war eine der ersten verfügbaren Formalisierungen von RDF und wurde von der RDF-Forschungsgemeinschaft diskutiert (vgl. z.B. [13, 94, 148]) bzw. als Ausgangspunkt oder Grundlage für weitere Arbeiten verwendet (vgl. z.B. [49, 118, 128, 152]). Die Notwendigkeit einer Formalisierung der RDF-Semantik ist mittlerweile offensichtlich. Das W3C hat daher eine Formalisierung der RDF-Semantik (RDF-Modelltheorie) auf mengentheoretischer Grundlage spezifiziert und entwickelt diese weiter. In der vorliegenden Arbeit wurde zur Formalisierung der RDF-

Semantik Prädikatenlogik erster Ordnung eingesetzt. Eine Anpassung dieser Formalisierung an den aktuellen Stand der RDF-Modelltheorie wird in Abschnitt 4 gezeigt. Weitere Änderungen der RDF-Semantik bzw. der RDF-Modelltheorie sind jedoch nicht unwahrscheinlich, so daß auch weitere Anpassungen der hier vorgestellten Formalisierung notwendig werden können. Die Universalität des verwendeten Formalismus berechtigt zu der Erwartung, dass diese Anpassungen in direkter Weise durchführbar sein werden.

Es gibt eine Reihe von aktuellen Ansätzen (wie z.B. DAML/OIL oder OWL), die Erweiterungen der RDF-Semantik vorgeschlagen. Die in dieser Arbeit vorgeschlagene Formalisierung der RDF-Semantik ist Ausgangspunkt weiterer Arbeiten, die einen Ansatz zur Einbettung von RDF in Wirtsformalismen vorschlagen (vgl. [41, 43, 44]). Hierdurch soll eine flexible Erweiterung der RDF-Semantik durch explizite Definition der erweiterten Semantik in der Notation eines Wirtsformalismuses ermöglicht werden. Bei geeignet gewähltem Wirtsformalismus (z.B. Prolog) ist eine derart definierte Semantik ohne weitere Modifikationen durch einen Interpreter verarbeitbar. Dies steht im Gegensatz zur Definition von (erweiterter) RDF-Semantik durch eine informale Spezifikation, bei der letztendlich die (Neu-)Implementierung eines Interpretersystems eine formale Umsetzung der intendierten Semantik ist. Die in dieser Arbeit vorgestellten Werkzeuge RDF-Schema-Explorer und WebObjectComposer unterstützen den Ansatz zur Einbettung von RDF in Wirtsformalismen bereits (in diesem Fall wird Prolog verwendet). Weiterer Vorteil bei einem derartigen Ansatz ist, daß bei geeignet gewähltem Wirtsformalismus eine wohletablierte Abfragesprache zur Verfügung steht und nicht erst eine 'RDF-Query-Language' (vgl. z.B. [93]) entwickelt werden müßte.

Ziel der RDF-Core-Working-Group ist die Revision der RDF-Syntax und die verbesserte Integration von RDF und XML. Der in dieser Arbeit vorgestellte XRDF-Ansatz umfaßt ein zu RDF kompatibles Datenmodell und eine zur RDF-Syntax kompatible Syntax. Im Gegensatz zur RDF-Syntax existiert für die XRDF-Syntax eine XML-DTD. Sie ist somit durch XML-Validatoren überprüfbar, womit auch eine verbesserte Integration in XML gegeben ist. Zur Wahrung der Kompatibilität umfaßt die aktuelle Revision der RDF-Syntax (vgl. [7]) nur wenige Änderungen und eine XML-DTD wird nach wie vor nicht zur Verfügung gestellt. In der RDF-M+S-Spezifikaton wird jedoch darauf hingewiesen, daß verschiedene Arten der Repräsentation von RDF-Beschreibungen möglich sind/sein können: ... *'it is also important to understand that this XML syntax is only one possible syntax for RDF and that alternate ways to represent the same RDF data model may emerge'*. Der XRDF-Ansatz kann daher als

Beitrag aufgefaßt werden, Ideen zu liefern, wie eine zukünftige XML-konforme und sehr einfache Syntax zur Erfassung von RDF-Beschreibungen aufgebaut sein kann. McDermott et al. haben Ideen, die dem XRDF-Ansatz zugrunde liegen, bereits aufgegriffen (vgl. [118]). Das Interesse an dem Ansatz ist zudem durch eine eingeladene Veröffentlichung des Ansatzes (vgl. [46]) im Buch *The emerging Semantic-Web* [50] dokumentiert.

Grundlage des Semantic-Web sind (logische) Aussagen, die automatisch interpretiert werden sollen. Zu diesem Zweck spezifizieren die RDF-Spezifikationen eine Syntax und ein Datenmodell. Diese Arbeit analysiert RDF und zeigt, wie RDF-Beschreibungen logikbasiert interpretiert werden können. Dies kann als eine Grundlage zur Verarbeitung von logikbasierter RDF-Information im Web gesehen werden. Durch zukünftige Arbeiten ist z.B. zu behandeln, wie bei unvollständiger, unsicherer oder widersprüchlicher Information zu verfahren ist und wie Aussagen aus unterschiedlichen Quellen (mit unterschiedlichen Vertrauensgraden) zu bewerten sind. Zur Lösung dieser Fragestellungen kann auf schon vorhandene Ergebnisse der Logikforschung zurückgegriffen werden. Diese sind jedoch an die Web-spezifischen Erfordernisse anzupassen und in Web-Technologien umzusetzen. Dies öffnet ein weites Feld für zukünftige Forschungsarbeiten.

9 Anhang

9.1 Definition des RDF-Datenmodells

Die folgenden Definitionen (1)-(11) wurden dem Abschnitt 5 (Formal Model for RDF) der RDF Model & Syntax Specification [101] entnommen.

(1) There is a set called Resources.

(2) There is a set called Literals.

(3) There is a subset of Resources called Properties.

(4) There is a set called Statements, each element of which is a triple of the form
`{pred,sub,obj}` where `pred` is a property (member of Properties), `sub` is a resource (member of Resources), and `obj` is either a resource or a literal (member of Literals).

(5) There is an element of Properties known as `type`.

(6) Members of Statements of the form `{type, sub, obj}` must satisfy the following: `sub` and `obj` are members of Resources. [27] places additional restrictions on the use of `type`.

(7) There is an element of Resources, not contained in Properties, known as `RDF:Statement`.

(8) There are three elements in Properties known as `RDF:predicate`, `RDF:subject` and `RDF:object`.

(9) Reification of a triple `{pred, sub, obj}` of Statements is an element `r` of Resources representing the reified triple and the elements `s1, s2, s3,` and `s4` of Statements such that

  (9a) `s1: {RDF:predicate, r, pred}`
  (9b) `s2: {RDF:subject, r, subj}`
  (9c) `s3: {RDF:object, r, obj}`
  (9d) `s4: {RDF:type, r, [RDF:Statement]}`

(10) There are three elements of Resources, not contained in Properties, known as `RDF:Seq`, `RDF:Bag`, and `RDF:Alt`.

(11) There is a subset of Properties corresponding to the ordinals (1, 2, 3, ...) called `Ord`. We refer to elements of `Ord` as `RDF:_1, RDF:_2, RDF:_3, ...`

9.2 Mengentheoretische Interpretation des RDF-Datenmodelles

1. $Resources = \{x | x \text{ ist URI gemäß RFC2396}\}$

2. $Literals = \{x | x \text{ ist String gemäß der XML-Spezifikation}\}$

3. $Properties \subset Resources$

4. $Statements \subseteq Properties \times Resources \times (Resources \cup Literals)$

5. $\text{RDF:type} \in Properties$

6. $TypeRel = \{(\text{RDF:type}, sub, obj) \in Statements | sub \in Resources \wedge obj \in Resources\}$

7. $\text{RDF:Statement} \in Resources$
   $\text{RDF:Statement} \notin Properties$

8. $\text{RDF:predicate} \in Properties$
   $\text{RDF:subject} \in Properties$
   $\text{RDF:object} \in Properties$

9. $$\begin{aligned} ReifiedStatement = \{r \in Resources | \\ & (\text{RDF:predicate}, r, pred) \in Statements \wedge \\ & (\text{RDF:subject}, r, subj) \in Statements \wedge \\ & (\text{RDF:object}, r, obj) \in Statements \wedge \\ & (\text{RDF:type}, r, \text{Statement}) \in Statements\} \end{aligned}$$

10. $\text{RDF:Seq} \in (Resources \setminus Properties)$
    $\text{RDF:Bag} \in (Resources \setminus Properties)$
    $\text{RDF:Alt} \in (Resources \setminus Properties)$

    $$\begin{aligned} collection = \{c \in Resources | \\ & container \in \{\text{RDF:Seq}, \text{RDF:Bag}, \text{RDF:Alt}\} \wedge \\ & o \in Resources \cup Literals \wedge \\ & i \in Ord \wedge \\ & (\text{RDF:type}, c, container) \in Statements \wedge \\ & (Ord, c, o) \in Statements \end{aligned}$$

11. $Ord \subset Properties$
    $Ord = \{\text{RDF:\_x} \in Properties | x \in N\}$

9.3 EBNF der RDF-Syntax

```
 [6.1] RDF            ::= ['<rdf:RDF>'] obj* ['</rdf:RDF>']
 [6.2] obj            ::= description | container
 [6.3] description    ::= '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '/>'
                        | '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '>'
                              propertyElt* '</rdf:Description>'
                        | typedNode
 [6.4] container      ::= sequence | bag | alternative
 [6.5] idAboutAttr    ::= idAttr | aboutAttr | aboutEachAttr
 [6.6] idAttr         ::= ' ID="' IDsymbol '"'
 [6.7] aboutAttr      ::= ' about="' URI-reference '"'
 [6.8] aboutEachAttr  ::= ' aboutEach="' URI-reference '"'
                        | ' aboutEachPrefix="' string '"'
 [6.9] bagIdAttr      ::= ' bagID="' IDsymbol '"'
[6.10] propAttr       ::= typeAttr
                        | propName '="' string '"' (with embedded quotes escaped)
[6.11] typeAttr       ::= ' type="' URI-reference '"'
[6.12] propertyElt    ::= '<' propName idAttr? '>' value '</' propName '>'
                        | '<' propName idAttr? parseLiteral '>'
                             literal '</' propName '>'
                        | '<' propName idAttr? parseResource '>'
                             propertyElt* '</' propName '>'
                        | '<' propName idRefAttr? bagIdAttr? propAttr* '/>'
[6.13] typedNode      ::= '<' typeName idAboutAttr? bagIdAttr? propAttr* '/>'
                        | '<' typeName idAboutAttr? bagIdAttr? propAttr* '>'
                             propertyElt* '</' typeName '>'
[6.14] propName       ::= Qname
[6.15] typeName       ::= Qname
[6.16] idRefAttr      ::= idAttr | resourceAttr
[6.17] value          ::= obj | string
[6.18] resourceAttr   ::= ' resource="' URI-reference '"'
[6.19] Qname          ::= [ NSprefix ':' ] name
[6.20] URI-reference  ::= string, interpreted per [URI]
[6.21] IDsymbol       ::= (any legal XML name symbol)
[6.22] name           ::= (any legal XML name symbol)
[6.23] NSprefix       ::= (any legal XML namespace prefix)
[6.24] string         ::= (any XML text, with "<", ">", and "&" escaped)
[6.25] sequence       ::= '<rdf:Seq' idAttr? '>' member* '</rdf:Seq>'
                        | '<rdf:Seq' idAttr? memberAttr* '/>'
[6.26] bag            ::= '<rdf:Bag' idAttr? '>' member* '</rdf:Bag>'
                        | '<rdf:Bag' idAttr? memberAttr* '/>'
[6.27] alternative    ::= '<rdf:Alt' idAttr? '>' member+ '</rdf:Alt>'
                        | '<rdf:Alt' idAttr? memberAttr? '/>'
[6.28] member         ::= referencedItem | inlineItem
[6.29] referencedItem ::= '<rdf:li' resourceAttr '/>'
[6.30] inlineItem     ::= '<rdf:li' '>' value </rdf:li>'
                        | '<rdf:li' parseLiteral '>' literal </rdf:li>'
```

```
                      | '<rdf:li' parseResource '>' propertyElt* </rdf:li>'
[6.31] memberAttr     ::= ' rdf:_n="' string '"' (where n is an integer)
[6.32] parseLiteral   ::= ' parseType="Literal"'
[6.33] parseResource  ::= ' parseType="Resource"'
[6.34] literal        ::= (any well-formed XML)
```

9.4 Interpretation der Triple-Notation des RDF-Datenmodells

```
triple    ::= '{' predicate',' subject',' object '}'
predicate ::=     URI | QName
subject   ::= '[' URI | QName ']'
object    ::= '[' URI | QName ']' | String
URI       ::= (URI gemäß RFC2396)
String    ::= '"'(String gemäß der XML-Spezifikation)'"'
QName     ::= (siehe Produktion 6 aus der XML-Namespace-Spezifikation)
```

9.5 RDF-Semantik erfaßt in Prädikatenlogik erster Ordnung

$$\forall s,p,o : arc(s,p,o) \land rfc2396Conform(p) \Rightarrow statement(s,p,o) \tag{4.1}$$

$$\forall s,p,o : statement(s,p,o) \Rightarrow res(s) \land uri(p) \land obj(o) \tag{4.2}$$

$$\forall r : uri(r) \Rightarrow res(r) \tag{4.3}$$

$$\forall r : res(r) \land rfc2396Conform(r) \Rightarrow namedRes(r) \tag{4.4}$$

$$\forall o : obj(o) \land \neg res(o) \Rightarrow lit(o) \tag{4.5}$$

$$\forall o : lit(o) \Rightarrow instanceOf(o, \text{'rdfs:Literal'}) \tag{4.6}$$

$$\begin{aligned} \forall s,s',p,o : res(s) \land uri(p) \land \\ statement(s', \text{'rdf:type'}, \text{'rdf:statement'}) \land \\ statement(s', \text{'rdf:subject'}, s) \land \\ statement(s', \text{'rdf:predicate'}, p) \land \\ statement(s', \text{'rdf:object',o}) \\ \Rightarrow reifies(s',s,p,o) \end{aligned} \tag{4.7}$$

$$\forall s,s',p,o : reifies(s',s,p,o) \Rightarrow reifyingStatement(s') \tag{4.8}$$

$$\begin{aligned}\forall s, s', p, o : reifies(s', s, p, o) \wedge statement(s, p, o) \\ \Rightarrow reifiesFact(s', s, p, o)\end{aligned} \tag{4.9}$$

$$\forall i, c : statement(i, \text{'rdf:type'}, c) \Rightarrow instanceOf(i, c) \tag{4.10}$$

$$\begin{aligned}\forall c1, c2 : statement(c1, \text{'rdfs:subClassOf'}, c2) \\ \Rightarrow subClassOf(c1, c2)\end{aligned} \tag{4.11}$$

$$\begin{aligned}\forall c1, c2, c3 : subClassOf(c1, c2) \wedge subClassOf(c2, c3) \\ \Rightarrow subClassOf(c1, c3)\end{aligned} \tag{4.12}$$

$$\begin{aligned}\forall i, c1, c2 : instanceOf(i, c1) \wedge subClassOf(c1, c2) \\ \Rightarrow instanceOf(i, c2)\end{aligned} \tag{4.13}$$

$$\begin{aligned}\forall c : subClassOf(c, c) \\ \Rightarrow violation('subClassCycle', c)\end{aligned} \tag{4.14}$$

$$\begin{aligned}\forall p1, p2 : statement(p1, \text{'rdfs:subPropertyOf'}, p2) \\ \Rightarrow subPropertyOf(p1, p2)\end{aligned} \tag{4.15}$$

$$\begin{aligned}\forall p1, p2, p3 : subPropertyOf(p1, p2) \wedge subPropertyOf(p2, p3) \\ \Rightarrow subPropertyOf(p1, p3)\end{aligned} \tag{4.16}$$

$$\begin{aligned}\forall s, p1, p2, o : statement(s, p1, o) \wedge subPropertyOf(p1, p2) \\ \Rightarrow statement(s, p2, o)\end{aligned} \tag{4.17}$$

$$\begin{aligned}\forall p : subPropertyOf(p, p) \\ \Rightarrow violation('subPropertyCycle', p)\end{aligned} \tag{4.18}$$

$$\begin{aligned}\forall p, i, c : statement(p, \text{rdfs:domain}, c) \wedge instanceOf(i, c) \\ \Rightarrow domain(i, p)\end{aligned} \tag{4.19}$$

$$\begin{aligned}\forall p, c : statement(p, \text{'rdfs:domain'}, c) \\ \Rightarrow domainConstrainedProp(p)\end{aligned} \tag{4.20}$$

$$\begin{aligned}\forall s, p, o : statement(s, p, o) \wedge \\ domainConstrainedProp(p) \wedge \\ \neg domain(s, p) \\ \Rightarrow violation('domain', s, p, o)\end{aligned} \tag{4.21}$$

$$\forall s, c : statement(p, \text{'rdfs:range'}, c) \Rightarrow isRange(c, p) \tag{4.22}$$

$$\forall p, c1, c2 : isRange(c1, p) \wedge isRange(c2, p) \wedge (c1 \neq c2) \Rightarrow violation('rangeCardinality', p) \tag{4.23}$$

$$\forall p, c : isRange(c, p) \Rightarrow hasRange(p) \tag{4.24}$$

$$\forall s, p, o, c : statement(s, p, o) \wedge instanceOf(o, c) \wedge isRange(c, p) \Rightarrow range(o, p) \tag{4.25}$$

$$\forall s, p, o : statement(s, p, o) \wedge hasRange(p) \wedge \neg range(o, p) \Rightarrow violation('range', s, p, o) \tag{4.26}$$

$$\forall s, o : statement(s, \text{'rdf:type'}, \text{'rdf:Seq'}) \wedge statement(s, \_1, o) \Rightarrow seq(s, o) \tag{4.27}$$

$$\forall s, o1, o2 : seq(s, o1) \wedge statement(s, \_2, o2) \Rightarrow seq(s, o1, o2) \tag{4.28}$$

$$statement(\text{'rdfs:Class'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.29}$$
$$statement(\text{'rdfs:Resource'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.30}$$
$$statement(\text{'rdfs:ConstraintResource'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.31}$$
$$statement(\text{'rdfs:Literal'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.32}$$
$$statement(\text{'rdf:Property'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.33}$$
$$statement(\text{'rdfs:ConstraintProperty'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.34}$$
$$statement(\text{'rdfs:ConstainerMembershipProperty'}, \text{'rdf:type'}, \text{'rdfs:Class'}) \tag{4.35}$$

$$statement(\text{'rdf:type'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.36}$$
$$statement(\text{'rdfs:subPropertyOf'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.37}$$
$$statement(\text{'rdfs:subClassOf'}, \text{'rdf:type'}, \text{'rdf:Property'}) \tag{4.38}$$
$$statement(\text{'rdfs:domain'}, \text{'rdf:type'}, \text{'rdfs:ConstraintProperty'}) \tag{4.39}$$
$$statement(\text{'rdfs:range'}, \text{'rdf:type'}, \text{'rdfs:ConstraintProperty'}) \tag{4.40}$$

$$statement(\text{'rdf:Statement'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.41}$$
$$statement(\text{'rdfs:Container'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.42}$$
$$statement(\text{'rdf:Bag'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.43}$$
$$statement(\text{'rdf:Seq'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.44}$$
$$statement(\text{'rdf:Alt'},\text{'rdf:type'},\text{'rdfs:Class'}) \tag{4.45}$$

$$statement(\text{'rdfs:subject','rdf:type','rdf:Property'}) \tag{4.46}$$
$$statement(\text{'rdfs:predicate','rdf:type','rdf:Property'}) \tag{4.47}$$
$$statement(\text{'rdfs:object','rdf:type','rdf:Property'}) \tag{4.48}$$
$$statement(\text{'rdfs:value','rdf:type','rdf:Property'}) \tag{4.49}$$

$$statement(\text{'rdf:\_1','rdf:type','rdfs:ContainerMembershipProperty'}) \tag{4.50}$$
$$\ldots$$
$$statement(\text{'rdf:\_n','rdf:type','rdfs:ContainerMembershipProperty'}) \tag{4.51}$$

$$statement(\text{'rdfs:Class','rdf:subClassOf','rdfs:Resource'}) \tag{4.52}$$
$$statement(\text{'rdf:Property','rdf:subClassOf','rdfs:Resource'}) \tag{4.53}$$
$$statement(\text{'rdfs:ConstraintResource','rdf:subClassOf','rdfs:Resource'}) \tag{4.54}$$
$$statement(\text{'rdfs:ConstraintProperty','rdf:subClassOf',} \\ \text{'rdfs:ConstraintResource'}) \tag{4.55}$$
$$statement(\text{'rdfs:ConstraintProperty','rdf:subClassOf','rdf:Property'}) \tag{4.56}$$
$$statement(\text{'rdfs:ConstainerMembershipProperty','rdf:subClassOf',} \\ \text{'rdf:Property'}) \tag{4.57}$$

$$statement(\text{'rdf:Statement','rdf:subClassOf','rdfs:Resource'}) \tag{4.58}$$
$$statement(\text{'rdfs:Container','rdf:subClassOf','rdfs:Resource'}) \tag{4.59}$$
$$statement(\text{'rdfs:Seq','rdf:subClassOf','rdfs:Container'}) \tag{4.60}$$
$$statement(\text{'rdfs:Bag','rdf:subClassOf','rdfs:Container'}) \tag{4.61}$$
$$statement(\text{'rdfs:Alt','rdf:subClassOf','rdfs:Container'}) \tag{4.62}$$

$$statement(\text{'rdf:type','rdfs:domain','rdfs:Resource'}) \tag{4.63}$$
$$statement(\text{'rdf:type','rdfs:range','rdfs:Class'}) \tag{4.64}$$
$$statement(\text{'rdfs:subClassOf','rdfs:domain','rdfs:Class'}) \tag{4.65}$$
$$statement(\text{'rdfs:subClassOf','rdfs:range','rdfs:Class'}) \tag{4.66}$$
$$statement(\text{'rdfs:subPropertyOf','rdfs:domain','rdf:Property'}) \tag{4.67}$$
$$statement(\text{'rdfs:subPropertyOf','rdf:range','rdf:Property'}) \tag{4.68}$$
$$statement(\text{'rdfs:domain','rdfs:domain','rdf:Property'}) \tag{4.69}$$
$$statement(\text{'rdfs:domain','rdfs:range','rdfs:Class'}) \tag{4.70}$$
$$statement(\text{'rdfs:range','rdfs:domain','rdf:Property'}) \tag{4.71}$$
$$statement(\text{'rdfs:range','rdfs:range','rdfs:Class'}) \tag{4.72}$$
$$statement(\text{'rdf:subject','rdfs:domain','rdf:Statement'}) \tag{4.73}$$
$$statement(\text{'rdf:subject','rdfs:range','rdfs:Resource'}) \tag{4.74}$$
$$statement(\text{'rdf:predicate','rdfs:domain','rdf:Statement'}) \tag{4.75}$$
$$statement(\text{'rdf:predicate','rdfs:range','rdf:Property'}) \tag{4.76}$$
$$statement(\text{'rdf:object','rdfs:domain','rdf:Statement'}) \tag{4.77}$$

9.6 RDF-Semantik erfaßt in Prolog

9.6.1 Prolog Regeln

```
uri(P) :- statement(_,P,_).
obj(O) :- statement(_,_,O).
res(S) :- statement(S,_,_).
res(R) :- uri(R).
lit(O) :- obj(O) , not(res(O)).

reifies(R,S,P,O) :-
    statement(R,rdf:type,rdf:'Statement'),
    statement(R,rdf:subject,S),
    statement(R,rdf:predicate,P),
    statement(R,rdf:object,O).

reifyingStatement(R) :- reifies(R,_,_,_), !.
reifiesFact(R,S,P,O) :- reifies(R,S,P,O) ,  statement(S,P,O).

subClassOf(C,D) :-
        statement(C,rdfs:subClassOf,D).
subClassOf(C,E) :-
        statement(C,rdfs:subClassOf,D) ,
        subClassOf(D,E).

instanceOf(O,rdfs:'Literal') :- lit(O).
instanceOf(O,rdfs:'Resource') :- res(O).
instanceOf(P,rdf:'Property') :- statement(_,P,_).

instanceOf(I,C) :-
        statement(I,rdf:type,C).
instanceOf(I,D) :-
        statement(I,rdf:type,C), subClassOf(C,D).

violation('subClassCycle',C) :- subClassOf(C,C).
violation('subClassCycle',C,rdfs:subClassOf,C) :-
         violation('subClassCycle',C).

subPropertyOf(A,B) :-
   statement(A,rdfs:subPropertyOf,B).
subPropertyOf(A,C) :-
   statement(A,rdfs:subPropertyOf,B),
   subPropertyOf(B,C).
violation('subPropertyCycle',P) :- subPropertyOf(P,P).
violation('subPropertyCycle',P,rdfs:subPropertyOf,P) :-
         violation('subPropertyCycle',P)
```

```
domain_constrained_prop(P) :-
        statement(P,rdfs:domain,_).
domain(X,P) :-
        statement(P,rdfs:domain,C), instanceOf(X,C).
violation('domain',S,P,O) :-
        statement(S,P,O),
        domain_constrained_prop(P),
        not(domain(S,P)).

is_range(X,P) :-
        statement(P,rdfs:range,X).
violation('rangeCardinality',P) :-
        is_range(X,P),
        is_range(Y,P),
        X \== Y.
violation('rangeCardinality',S,rdfs:range,O) :-
        violation('rangeCardinality',S),
        statement(S,rdfs:range,O).

has_range(P) :- is_range(_,P).
range(X,P) :- is_range(C,P) , instanceOf(X,C).
violation('range',S,P,O) :- statement(S,P,O) , has_range(P) , not(range(O,P)).

statement(rdfs:'Container', rdf:type, rdfs:'Class').
statement(rdfs:'Container', rdfs:subClassOf, rdfs:'Resource').

statement(rdf:'Bag', rdf:type, rdfs:'Class').
statement(rdf:'Bag', rdfs:subClassOf, rdfs:'Container').
statement(rdf:'Alt', rdf:type, rdfs:'Class').
statement(rdf:'Alt', rdfs:subClassOf, rdfs:'Container').
statement(rdf:'Seq', rdf:type, rdfs:'Class').
statement(rdf:'Seq', rdfs:subClassOf, rdfs:'Container').

statement(rdf:'_1', rdf:type, rdfs:'ContainerMembershipProperty').
...
statement(rdf:'_10', rdf:type, rdfs:'ContainerMembershipProperty').
```

9.6.2 Prolog Konzept-Fakten

```
statement(rdfs:'Literal',rdf:type,rdfs:'Class').
statement(rdfs:'Class',rdf:type,rdfs:'Class').
statement(rdfs:'Resource',rdf:type,rdfs:'Class').
statement(rdf:'Property',rdf:type,rdfs:'Class').
statement(rdfs:'ConstraintResource',rdf:type,rdfs:'Class').
statement(rdfs:'ConstraintProperty',rdf:type,rdfs:'Class').
statement(rdfs:'ContainerMembershipProperty',rdf:type,rdfs:'Class').
statement(rdfs:range,rdf:type,rdfs:'ConstraintProperty').
```

```
statement(rdfs:domain,rdf:type,rdfs:'ConstraintProperty').
statement(rdf:type,rdf:type,rdf:'Property').
statement(rdfs:subPropertyOf,rdf:type,rdf:'Property').
statement(rdfs:subClassOf,rdf:type,rdf:'Property').
statement(rdfs:seeAlso,rdf:type,rdf:'Property').
statement(rdfs:isDefinedBy,rdf:type,rdf:'Property').
statement(rdfs:comment,rdf:type,rdf:'Property').
statement(rdfs:label,rdf:type,rdf:'Property').
statement(rdf:subject,rdf:type,rdf:'Property').
statement(rdf:predicate,rdf:type,rdf:'Property').
statement(rdf:object,rdf:type,rdf:'Property').
statement(rdf:'Statement',rdf:type,rdfs:'Class').

statement(rdfs:'Class',rdfs:subClassOf,rdfs:'Resource').
statement(rdfs:'ConstraintResource',rdfs:subClassOf,rdfs:'Resource').
statement(rdf:'Property',rdfs:subClassOf,rdfs:'Resource').
statement(rdfs:'ConstraintProperty',rdfs:subClassOf,rdf:'Property').
statement(rdfs:'ConstraintProperty',rdfs:subClassOf,rdfs:'ConstraintResource').
statement(rdfs:'ContainerMembershipProperty',rdfs:subClassOf,rdf:'Property').
```

9.6.3 Prolog Constraint-Fakten

```
statement(rdfs:range,rdfs:range,rdfs:'Class').
statement(rdf:type,rdfs:range,rdfs:'Class').
statement(rdfs:subClassOf,rdfs:range,rdfs:'Class').
statement(rdfs:domain,rdfs:range,rdfs:'Class').
statement(rdfs:comment,rdfs:range,rdfs:'Literal').
statement(rdfs:label,rdfs:range,rdfs:'Literal').
statement(rdf:subject,rdfs:range,rdfs:'Resource').
statement(rdf:predicate,rdfs:range,rdf:'Property').
statement(rdfs:subPropertyOf,rdfs:range,rdf:'Property').

statement(rdfs:subPropertyOf,rdfs:domain,rdf:'Property').
statement(rdfs:range,rdfs:domain,rdf:'Property').
statement(rdfs:domain,rdfs:domain,rdf:'Property').
statement(rdf:subject,rdfs:domain,rdf:'Statement').
statement(rdf:predicate,rdfs:domain,rdf:'Statement').
statement(rdf:object,rdfs:domain,rdf:'Statement').
statement(rdf:type,rdfs:domain,rdfs:'Resource').
statement(rdfs:subClassOf,rdfs:domain,rdfs:'Class').
statement(rdfs:comment,rdfs:domain,rdfs:'Resource').
statement(rdfs:label,rdfs:domain,rdfs:'Resource').
```

9.7 RDF-Semantik erfaßt in Datalog

9.7.1 Datalog Regeln

Folgende Regeln und Fakten können in den Datalog-Parser SiLRI (mit der Option -simpel) geladen und danach befragt werden. Die Fakten aus einem RDF-Graph werden mit dem Prädikatensymbol `s` in die Wissensbasis eingetragen. Verwendet wird Datalog mit Negations.

```
res(S) & uri(P) & obj(O) <- s(S,P,O).
res(R) <- uri(R).
lit(O) <- obj(O) & -res(O).
instanceOf(O,rdfs_Literal) <- lit(O).

reifies(R,S,P,O) <- res(S) & uri(P) &
        s(R,rdf_type,rdf_statement) & s(R,rdf_subject,S) &
        s(R,rdf_predicate,P)        & s(R,rdf_object,O).
reifyingStatement(R) <- reifies(R,S,P,O).
reifiesFact(R,S,P,O) <- reifies(R,S,P,O) &  s(S,P,O).

instanceOf(I,C) <- s(I,rdf_type,C).
subClassOf(C,D) <- s(C,rdfs_subClassOf,D).
subClassOf(C,E) <- subClassOf(C,D) & subClassOf(D,E).
instanceOf(I,D) <- instanceOf(I,C) & subClassOf(C,D).
violation("subClassCycle",C) <- subClassOf(C,C).
violation("subClassCycle",C,rdfs_subClassOf,C) <-
                  violation("subClassCycle",C).

subPropertyOf(A,B) <- s(A,rdfs_subPropertyOf,B).
subPropertyOf(A,C) <- subPropertyOf(A,B) & subPropertyOf(B,C).
s(S,P2,O) <- s(S,P1,O) & subPropertyOf(P1,P2).
violation("subPropertyCycle",P) <- subPropertyOf(P,P).
violation("subPropertyCycle",P,rdfs_subPropertyOf,P) <-
                     violation("subPropertyCycle",P).

domain_constrained_prop(P) <- s(P,rdfs_domain,X).
domain(X,P) <- s(P,rdfs_domain,C) & instanceOf(X,C).
violation("domain",S,P,O) <- s(S,P,O) &
                           domain_constrained_prop(P) &
                           -domain(S,P).

is_range(X,P) <- s(P,rdfs_range,X).
violation("rangeCardinality",P) <- is_range(X,P) &
                                   is_range(Y,P) & -unify(X,Y).
violation("rangeCardinality",S,rdfs_range,O) <-
          violation("rangeCardinality",S) & s(S,rdfs_range,O).
```

```
has_range(P) <- is_range(X,P).
range(X,P) <- is_range(C,P) & instanceOf(X,C).
violation("range",S,P,O) <- s(S,P,O) & has_range(P) & -range(O,P).
```

9.7.2 Datalog Konzept-Fakten

```
s(rdfs_Literal,rdf_type,rdfs_Class).
s(rdfs_Class,rdf_type,rdfs_Class).
s(rdfs_Resource,rdf_type,rdfs_Class).
s(rdf_Property,rdf_type,rdfs_Class).
s(rdfs_ConstraintResource,rdf_type,rdfs_Class).
s(rdfs_ConstraintProperty,rdf_type,rdfs_Class).
s(rdfs_ContainerMembershipProperty,rdf_type,rdfs_Class).
s(rdfs_range,rdf_type,rdfs_ConstraintProperty).
s(rdfs_domain,rdf_type,rdfs_ConstraintProperty).
s(rdf_type,rdf_type,rdf_Property).
s(rdfs_subPropertyOf,rdf_type,rdf_Property).
s(rdfs_subClassOf,rdf_type,rdf_Property).
s(rdfs_seeAlso,rdf_type,rdf_Property).
s(rdfs_isDefinedBy,rdf_type,rdf_Property).
s(rdfs_comment,rdf_type,rdf_Property).
s(rdfs_label,rdf_type,rdf_Property).

s(rdf_subject,rdf_type,rdf_Property).
s(rdf_predicate,rdf_type,rdf_Property).
s(rdf_object,rdf_type,rdf_Property).
s(rdf_Statement,rdf_type,rdfs_Class).

s(rdfs_Class,rdfs_subClassOf,rdfs_Resource).
s(rdfs_ConstraintResource,rdfs_subClassOf,rdfs_Resource).
s(rdf_Property,rdfs_subClassOf,rdfs_Resource).
s(rdfs_ConstraintProperty,rdfs_subClassOf,rdf_Property).
s(rdfs_ConstraintProperty,rdfs_subClassOf,
                                rdfs_ConstraintResource).
s(rdfs_ContainerMembershipProperty,
                           rdfs_subClassOf,rdf_Property).
```

9.7.3 Datalog Constraint-Fakten

```
s(rdfs_range,rdfs_range,rdfs_Class).
s(rdf_type,rdfs_range,rdfs_Class).
s(rdfs_subClassOf,rdfs_range,rdfs_Class).
s(rdfs_domain,rdfs_range,rdfs_Class).
s(rdfs_comment,rdfs_range,rdfs_Literal).
s(rdfs_label,rdfs_range,rdfs_Literal).
```

```
s(rdf_subject,rdfs_range,rdfs_Resource).
s(rdf_predicate,rdfs_range,rdf_Property).
s(rdfs_subPropertyOf,rdfs_range,rdf_Property).

s(rdfs_subPropertyOf,rdfs_domain,rdf_Property).
s(rdfs_range,rdfs_domain,rdf_Property).
s(rdfs_domain,rdfs_domain,rdf_Property).
s(rdf_subject,rdfs_domain,rdf_Statement).
s(rdf_predicate,rdfs_domain,rdf_Statement).
s(rdf_object,rdfs_domain,rdf_Statement).
s(rdf_type,rdfs_domain,rdfs_Resource).
s(rdfs_subClassOf,rdfs_domain,rdfs_Class).
s(rdfs_comment,rdfs_domain,rdfs_Resource).
s(rdfs_label,rdfs_domain,rdfs_Resource).
```

9.8 WebObjectComposition-Schema (woc.s)

```
<?xml version="1.0"?>
              <!-- WebObjectComposition Schema -->
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>


<rdf:RDF
     xmlns:rdf="&rdf;"
     xmlns:rdfs="&rdfs;"
     xmlns:xwmf="this#">

   <!-- Note: this RDF vocabulary can be used with RDF data
        by referencing it with an XML namespace declaration, for example
        xmlns:xwmf="http://.../schemata/woc.s#". This allows
        to use abbreviations such as xwmf:template to refer
        unambiguously the  property 'template' defined in this
        schema.
   -->

 <rdfs:Class rdf:ID="Simplexon">
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
  <rdfs:comment>
    A simplexon defines by means of the property template
    the Web implementation of objects of that class. Thus, it
    supports the separation of content from its Web
    implementation because as developer can define an object´s content
    independently from its Web implementation
  </rdfs:comment>
 </rdfs:Class>

 <rdf:Property rdf:ID="template">
  <rdfs:domain rdf:resource="#Simplexon"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - defines the implementation markup/code of a simplexon instance
   - within the implementation markup/code the content
     of the xml element var is a placeholder
   - these property can be overriden by sub-classes
   - at most one template per simplexon class is allowed
   - formerly called implementation
  </rdfs:comment>
 </rdf:Property>
```

```
<!-- ELEMENT var (#PCDATA)
  an XML element which is interpreted by the WebObjectComposer for
  filling a slot/replace a placeholder with the value of a property
  defining content of a Web application. It's usage is allowed
  only in the value of the property template. It is not interpreted by
  an rdf parser or xml parser, if (as recommended) the the RDF-attribute
  parseType='Literal' is used.
-->

 <rdf:Property rdf:ID="view">
  <rdfs:domain rdf:resource="#Simplexon"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - to specify the context (view) a simplexon or component
     has to be used in
  </rdfs:comment>
 </rdf:Property>

 <rdfs:Class rdf:ID="Complexon">
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   <rdfs:comment>
    - defines struture by means of the property hasPart
   </rdfs:comment>
 </rdfs:Class>

 <rdf:Property rdf:ID="hasPart">
  <rdfs:domain rdf:resource="#Complexon"/>
  <rdfs:range  rdf:resource="&rdf;Seq"/>
  <rdfs:comment>
   - arranges complexons and simplexons in an directed acyclic graph
   - the value is a sequence of references to complexons or simplexons
  </rdfs:comment>
 </rdf:Property>

 <rdfs:Class rdf:ID="Component">
   <rdfs:subClassOf rdf:resource="#Complexon"/>
   <rdfs:comment>
     - represents a (Web-)document
   </rdfs:comment>
 </rdfs:Class>

 <rdf:Property rdf:ID="isComponent">
  <xwmf:domain rdf:resource="#Component"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - the value of this Property defines the URI from which
     an component can be fechted
  </rdfs:comment>
```

```
 </rdf:Property>

 <rdf:Property rdf:ID="isView">
  <xwmf:domain rdf:resource="#Comonent"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - used to specify the view which is to be applied for
     composing/generating the implementation code of a component
  </rdfs:comment>
 </rdf:Property>

 <rdfs:Class rdf:ID="Query">
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   <rdfs:comment>
    Some description
   </rdfs:comment>
 </rdfs:Class>

 <rdf:Property rdf:ID="isDefinedAs">
  <rdfs:domain rdf:resource="&rdfs;Resource"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - can be assigned to an resource
   - defines prolog-predicates which will be asserted
   - the name of a prolog-predicate has to be ns:name
   - this property is allowed a most once per resource
   - for an example see sort_list
  </rdfs:comment>
 </rdf:Property>

 <rdf:Property rdf:ID="select">
  <rdfs:domain rdf:resource="#Query"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - can be assigned to an instance of class Query
   - defines a query formulated in prolog code
   - this property is allowed a most once per instance
     of type Query
   - behaves like isDefinedAs
  </rdfs:comment>
 </rdf:Property>

<rdf:Property rdf:ID="sort_list">
 <rdfs:isDefinedAs rdf:parseType="Literal">
 % xwmf:make_key_value_list(+List, -Key_Value_List)
   xwmf:make_key_value_list(List,Property, Temp_List, Key_Value_List) :-
        [Item|T] = List,
        % Object is Literal
```

```
        statement(Item, Property, literal(Label)),
        New_Temp_List = [Label-Item|Temp_List],
        xwmf:make_key_value_list(T, Property, New_Temp_List, Key_Value_List).

   xwmf:make_key_value_list(List,Property, Temp_List, Key_Value_List) :-
        [Item|T] = List,
        % Object is Resource
        statement(Item, Property, Label),
        New_Temp_List = [Label-Item|Temp_List],
        xwmf:make_key_value_list(T, Property, New_Temp_List, Key_Value_List).

   xwmf:make_key_value_list(List, _, New_Temp_List, Key_Value_List) :-
        [] = List,
        Key_Value_List = New_Temp_List.

   xwmf:make_list(Sorted_Key_Value_List,Temp_List,List) :-
         [Key_Value_Pair|T] = Sorted_Key_Value_List,
         _-Item = Key_Value_Pair,
         New_Temp_List = [Item| Temp_List],
         xwmf:make_list(T,New_Temp_List,List).

   xwmf:make_list(Sorted_Key_Value_List,Temp_List,List) :-
        [] = Sorted_Key_Value_List,
        reverse(Temp_List, List).

   xwmf:sort_list(Input_List, Property, List) :-
        xwmf:make_key_value_list(Input_List,Property,[], Key_Value_List),
        keysort(Key_Value_List, Sorted_Key_Value_List),
        xwmf:make_list(Sorted_Key_Value_List,[],List).
 </rdfs:isDefinedAs>
 <rdfs:comment>
   - sort a list of instances in standard order
   - use sort_list with the xwmf property select
      e.g. xwmf:sort_list(+Input_List, +Property, -List)
   - example of how to apply the extension mechanism
  </rdfs:comment>
 </rdf:Property>

</rdf:RDF>
```

9.9 Beispiel einer Web-Anwendungsbeschreibung

Für Erläuterungen siehe Kapitel 6.

9.9.1 swt.wad

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
 <!ENTITY gwas 'file:./htdocs/descr/generic.was#'>
 <!ENTITY gwad 'file:./htdocs/descr/generic.wad#'>
 <!ENTITY ewad 'file:./htdocs/descr/employee.wad#'>
 <!ENTITY pwad 'file:./htdocs/descr/publication.wad#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:gwas="&gwas;"
      xmlns:gwad="&gwad;"
      xmlns:ewad="&ewad;"
      xmlns:pwad="&pwad;"
      xmlns:swt="this#">

   <!-- entry page - html format -->
   <gwas:MenueHTML rdf:ID="i1MenueHtml">
    <xwmf:isComponent>/index.html</xwmf:isComponent>
    <xwmf:isView>html</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
       <li resource="#i1text"/>
       <li resource="&gwad;documentEnd"/>
     </Seq>
    </xwmf:hasPart>
   </gwas:MenueHTML>

   <!-- entry page - wml format -->
   <gwas:MenueHTML rdf:ID="i1MenueWml">
    <xwmf:isComponent>/index.wml</xwmf:isComponent>
    <xwmf:isView>wml</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
```

```
      <li resource="&gwad;documentBegin"/>
      <li resource="#i1text"/>
      <li resource="&gwad;documentEnd"/>
    </Seq>
   </xwmf:hasPart>
  </gwas:MenueHTML>

  <rdf:Description ID="i1text">
   <rdf:type resource="&gwas;HtmlText"/>
   <rdf:type resource="&gwas;WmlText"/>
    <gwas:text>
     <![CDATA[
      The content presented by this Web server is ...
    ]]>
    </gwas:text>
  </rdf:Description>

</rdf:RDF>
```

9.9.2 generic.wad

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY gwas 'file:./htdocs/descr/generic.was#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:gwas="&gwas;">

  <rdf:Description ID="documentBegin">
   <rdf:type resource="&gwas;HtmlDocumentBegin"/>
   <rdf:type resource="&gwas;WmlDocumentBegin"/>
    <gwas:title>
       Information Systems and Software Techniques
    </gwas:title>
  </rdf:Description>

  <rdf:Description ID="documentEnd">
   <rdf:type resource="&gwas;HtmlDocumentEnd"/>
   <rdf:type resource="&gwas;WmlDocumentEnd"/>
  </rdf:Description>
</rdf:RDF>
```

9.9.3 generic.was

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:gwas="this#">

   <rdfs:Class rdf:ID="DocumentBegin">
    <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   </rdfs:Class>

   <Property ID="title">
    <rdfs:domain rdf:resource="#DocumentBegin"/>
    <rdfs:range  rdf:resource="&rdfs;Literal"/>
   </Property>

   <xwmf:Simplexon rdf:ID="HtmlDocumentBegin">
   <rdfs:subClassOf rdf:resource="#DocumentBegin"/>
   <xwmf:view>html</xwmf:view>
    <xwmf:template rdf:parseType="Literal">
      <![CDATA[
      <HTML>
       <HEAD>
         <TITLE>
           <var>gwas:title</var>
        </TITLE>
       </HEAD>
       <BODY bgcolor="FFFFFF">
       <DIV ALIGN=CENTER>
       <H4><A HREF="http://www.uni-essen.de">University of Essen</A></H4>

       <H3><A href="/index.html">Information Systems and Software Techniques</A></H3>
       <H4>XWMF Demonstration Server</H4>
       [<A HREF="/index.html">Entry</A>]
       [<A HREF="/index_of_employees.html">Employees</A>]
       [<A HREF="/index_of_publications.html">Publications</A>]
       [<A HREF="/descr/">RDF Descriptions</A>]
       </DIV>
```

```
    <HR>
   ]]>
</xwmf:template>
</xwmf:Simplexon>

<xwmf:Simplexon rdf:ID="WmlDocumentBegin">
<rdfs:subClassOf rdf:resource="#DocumentBegin"/>
<xwmf:view>wml</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
   <![CDATA[
     <wml>

      <template>
        <do type="prev" label="Back">
          <prev/>
        </do>
      </template>

      <card id = "cardOne" title="<var>gwas:title</var>">

    University of Essen<br/>
    Information Systems and Software Techniques</br>
    XWMF Demonstration Server</br>
    [<a href="/index.wml">Entry</a>]
    [<a href="/index_of_employees.wml">Employees</a>]
    [<a href="/index_of_publications.wml">Publications</a>]
    [<a HREF="/descr">RDF Descriptions</a>]</br></br>

    <b>XWMF Demonstration Server</b>
   ]]>
</xwmf:template>
</xwmf:Simplexon>

<rdfs:Class rdf:ID="DocumentEnd">
 <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<xwmf:Simplexon rdf:ID="HtmlDocumentEnd">
<rdfs:subClassOf rdf:resource="#DocumentEnd"/>
<xwmf:view>html</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
   <![CDATA[
     </BODY>
    </HTML>
   ]]>
</xwmf:template>
</xwmf:Simplexon>
```

```
   <xwmf:Simplexon rdf:ID="WmlDocumentEnd">
   <rdfs:subClassOf rdf:resource="#DocumentEnd"/>
   <xwmf:view>wml</xwmf:view>
    <xwmf:template rdf:parseType="Literal">
      <![CDATA[
        </card>
       </wml>
      ]]>
   </xwmf:template>
   </xwmf:Simplexon>

   <rdfs:Class rdf:ID="Text">
    <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   </rdfs:Class>

   <Property ID="text">
    <rdfs:domain rdf:resource="#Text"/>
    <rdfs:range  rdf:resource="&rdfs;Literal"/>
   </Property>

   <xwmf:Simplexon rdf:ID="HtmlText">
   <rdfs:subClassOf rdf:resource="#Text"/>
   <xwmf:view>html</xwmf:view>
    <xwmf:template rdf:parseType="Literal">
     <![CDATA[
      <var>gwas:text</var>
      ]]>
   </xwmf:template>
   </xwmf:Simplexon>

   <xwmf:Simplexon rdf:ID="WmlText">
   <rdfs:subClassOf rdf:resource="#DocumentEnd"/>
   <xwmf:view>wml</xwmf:view>
    <xwmf:template rdf:parseType="Literal">
       <![CDATA[
      <var>gwas:text</var>
      ]]>
   </xwmf:template>
   </xwmf:Simplexon>

<rdfs:Class rdf:ID="MenueHTML">
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
</rdfs:Class>

</rdf:RDF>
```

9.9.4 employee.wad

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
 <!ENTITY cm   'file:./htdocs/descr/cm.s#'>
 <!ENTITY gwad 'file:./htdocs/descr/generic.wad#'>
 <!ENTITY ewas 'file:./htdocs/descr/employee.was#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:cm="&cm;"
      xmlns:gwad="&gwad;"
      xmlns:ewas="&ewas;"
      xmlns:ewad="this#">

   <rdf:Description ID="i1Contact">
    <rdf:type resource="&ewas;HtmlContact"/>
    <rdf:type resource="&ewas;WmlContact"/>
     <ewas:name>Reinhold Klapsing</ewas:name>
     <ewas:streetANDNo>Universitätstr. 9</ewas:streetANDNo>
     <ewas:residence>45141 Essen</ewas:residence>
     <ewas:phone>+49 (201) 183 4078</ewas:phone>
     <ewas:fax>+49 (201) 183 4073</ewas:fax>
     <ewas:email>Reinhold.Klapsing@uni-essen.de</ewas:email>
     <ewas:imguri>/klapsing.jpg</ewas:imguri>
     <cm:expires>2003-01-01</cm:expires>
   </rdf:Description>

<!--           employee Klapsing
     The following two components describe
     HTML and WML documents delivering
     information  about employee Klapsing
-->
   <ewas:EmployeeHTML rdf:ID="i1EmployeeHtml">
    <xwmf:isComponent>/Employee_Klapsing.html</xwmf:isComponent>
    <xwmf:isView>html</xwmf:isView>
    <ewas:linklabel>Klapsing (former)</ewas:linklabel>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
```

```
    <li resource="#i1Contact"/>
    <li resource="&gwad;documentEnd"/>
  </Seq>
 </xwmf:hasPart>
</ewas:EmployeeHTML>

<ewas:EmployeeWML rdf:ID="i1EmployeeWml">
 <xwmf:isComponent>/Employee_Klapsing.wml</xwmf:isComponent>
 <xwmf:isView>wml</xwmf:isView>
 <ewas:linklabel>Klapsing (former)</ewas:linklabel>
 <xwmf:hasPart>
  <Seq>
    <li resource="&gwad;documentBegin"/>
    <li resource="#i1Contact"/>
    <li resource="&gwad;documentEnd"/>
  </Seq>
 </xwmf:hasPart>
</ewas:EmployeeWML>

... to shorten the appendix the another instances are deleted ...

<!-- employee index - html format -->
<xwmf:Query rdf:ID="listOfEmployeeHTML">
<xwmf:select rdf:parseType="Literal">
 select(ewad:listOfEmployeeHTML,List) :-
  setof(X, statement(X, rdf:type, ewas:'EmployeeHTML'), EmployeeHTML_List),
  sort_list(xwmf, EmployeeHTML_List,ewas:linklabel, List).
 </xwmf:select>
<rdfs:comment>
 List of Instances of EmployeeHTML.
 List is sorted in standard order for the predicate ewas:linklabel
</rdfs:comment>
</xwmf:Query>

<!-- employee index - wml format -->
<xwmf:Query rdf:ID="listOfEmployeeWML">
<xwmf:select rdf:parseType="Literal">
 select(ewad:listOfEmployeeWML,List) :-
  setof(X, statement(X, rdf:type, ewas:'EmployeeWML'), EmployeeWML_List),
  sort_list(xwmf, EmployeeWML_List,ewas:linklabel, List).
 </xwmf:select>
<rdfs:comment>
 List of Instances of EmployeeWML.
 List is sorted in standard order for the predicate ewas:linklabel
</rdfs:comment>
</xwmf:Query>

<ewas:IndexOfEmployeeHTML rdf:ID="indexEmployeeHtml">
```

```
    <xwmf:isComponent>/index_of_employees.html</xwmf:isComponent>
    <xwmf:isView>html</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
       <li resource="#listOfEmployeeHTML"/>
       <li resource="&gwad;documentEnd"/>
     </Seq>
    </xwmf:hasPart>
   </ewas:IndexOfEmployeeHTML>

   <ewas:IndexOfEmployeeWML rdf:ID="indexEmployeeWml">
    <xwmf:isComponent>/index_of_employees.wml</xwmf:isComponent>
    <xwmf:isView>wml</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
       <li resource="#listOfEmployeeWML"/>
       <li resource="&gwad;documentEnd"/>
     </Seq>
    </xwmf:hasPart>
   </ewas:IndexOfEmployeeWML>

</rdf:RDF>
```

9.9.5 employee.was

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:ewas="this#">

 <rdfs:Class rdf:ID="Contact">
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
 </rdfs:Class>

 <Property ID="name">
```

```
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="streetANDNo">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="residence">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="phone">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="fax">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="email">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<Property ID="imguri">
 <rdfs:domain rdf:resource="#Contact"/>
 <rdfs:range  rdf:resource="&rdfs;Literal"/>
</Property>

<xwmf:Simplexon rdf:ID="HtmlContact">
 <rdfs:subClassOf rdf:resource="#Contact"/>
 <xwmf:view>html</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
  <![CDATA[
     <CENTER>
     <TABLE bgcolor="#e8e8e8" ><TR>
     <TD valign="bottom">
      <var>ewas:name</var><br>
      <var>ewas:streetANDNo</var><br>
      <var>ewas:residence</var><br>
      Phone: <var>ewas:phone</var><br>
      Fax: <var>ewas:fax</var><br>
      <A HREF="mailto:<var>ewas:email</var>">
```

```
      <var>ewas:email</var></A><BR>
     </TD>
     <TD><ifp><img src="<var>ewas:imguri</var>"></ifp></TD>
    </TABLE>
    </CENTER>
  ]]>
 </xwmf:template>
</xwmf:Simplexon>


<xwmf:Simplexon rdf:ID="WmlContact">
 <rdfs:subClassOf rdf:resource="#Contact"/>
 <xwmf:view>wml</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
  <![CDATA[
    <var>ewas:name</var><br/>
    <var>ewas:streetANDNo</var><br/>
    <var>ewas:residence</var><br/>
     Phone: <var>ewas:phone</var><br/>
     Fax: <var>ewas:fax</var><br/>
     <var>ewas:email</var><br/>
   ]]>
 </xwmf:template>
</xwmf:Simplexon>

<rdfs:Class rdf:ID="Employee">
 <rdfs:subClassOf rdf:resource="&xwmf;Complexon"/>
</rdfs:Class>

<rdfs:Class rdf:ID="IndexOfEmployeeHTML">
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
</rdfs:Class>

<rdfs:Class rdf:ID="IndexOfEmployeeWML">
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
</rdfs:Class>

<rdfs:Class rdf:ID="EmployeeHTML">
 <rdf:type rdf:resource="&xwmf;Simplexon"/>
 <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
 <xwmf:view>html</xwmf:view>
 <xwmf:template rdf:parseType="Literal">
  <![CDATA[<A HREF="<var>xwmf:isComponent</var>">
                  <var>ewas:linklabel</var></A><P>
  ]]>
 </xwmf:template>
</rdfs:Class>
```

```
 <rdfs:Class rdf:ID="EmployeeWML">
  <rdf:type rdf:resource="&xwmf;Simplexon"/>
  <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
  <xwmf:view>wml</xwmf:view>
  <xwmf:template rdf:parseType="Literal">
   <![CDATA[<a href="<var>xwmf:isComponent</var>">
                    <var>ewas:linklabel</var></a><br/>
   ]]>
  </xwmf:template>
 </rdfs:Class>

 <Property ID="linklabel">
  <rdfs:domain rdf:resource="#EmployeeHTML"/>
  <rdfs:domain rdf:resource="#EmployeeWML"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
 </Property>

</rdf:RDF>
```

9.9.6 publication.wad

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
 <!ENTITY gwad 'file:./htdocs/descr/generic.wad#'>
 <!ENTITY dc   'file:./htdocs/descr/dc.s#'>
 <!ENTITY pwas 'file:./htdocs/descr/publication.was#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:gwad="&gwad;"
      xmlns:dc="&dc;"
      xmlns:pwas="&pwas;"
      xmlns:pwad="this#">

...

<rdf:Description rdf:ID="i36Publication">
 <rdf:type resource="&pwas;PublicationHTML"/>
 <rdf:type resource="&pwas;PublicationWML"/>
```

```
 <dc:creator>Reinhold Klapsing, Gustaf Neumann,
             and Wolfram Conen
 </dc:creator>
 <dc:title>Semantics in Web Engineering:
           Applying the Resource Description Framework
 </dc:title>
 <dc:coverage>
   IEEE MultiMedia, Vol. 8, No. 2, April-June 2001
 </dc:coverage>
 <dc:publisher>IEEE</dc:publisher>
 <dc:date>2001</dc:date>
 <dc:format>PDF</dc:format>
 <dc:identifier>
  http://computer.org/multimedia/mu2001/u2toc.htm#u2062
 </dc:identifier>
 <dc:type>journal article</dc:type>
 <dc:language>en</dc:language>
</rdf:Description>

   ... to shorten the appendix the another instances are deleted ...

   <!-- list of publications -->
   <xwmf:Query rdf:ID="listOfPublicationHtml">
    <xwmf:select rdf:parseType="Literal">
    select(pwad:listOfPublicationHtml,List) :-
          setof(X, direct_instanceOf(X, pwas:'PublicationHTML'),
          PublicationHTML_List),
          sort_list(xwmf,PublicationHTML_List,dc:date, SortedList),
          reverse(SortedList, List).
    </xwmf:select>
   <rdfs:comment>
    List of Instances of PublicationHtml
    List is sorted in standard order for the predicate pwas:date
   </rdfs:comment>
   </xwmf:Query>

   <pwas:IndexOfPublicationHTML rdf:ID="indexPublicationHtml">
    <xwmf:isComponent>/index_of_publications.html</xwmf:isComponent>
    <xwmf:isView>html</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
       <li resource="#listOfPublicationHtml"/>
       <li resource="&gwad;documentEnd"/>
     </Seq>
    </xwmf:hasPart>
   </pwas:IndexOfPublicationHTML>
```

```
   <xwmf:Query rdf:ID="listOfPublicationWml">
    <xwmf:select rdf:parseType="Literal">
    select(pwad:listOfPublicationWml,List) :-
          setof(X, direct_instanceOf(X, pwas:'PublicationWML'),
          PublicationWML_List),
          sort_list(xwmf,PublicationWML_List,dc:date, SortedList),
          reverse(SortedList, List).
    </xwmf:select>
   <rdfs:comment>
    List of Instances of PublicationHtml
    % List is sorted in standard order for the predicate pwas:date
   </rdfs:comment>
   </xwmf:Query>

   <pwas:IndexOfPublicationWML rdf:ID="indexPublicationWml">
    <xwmf:isComponent>/index_of_publications.wml</xwmf:isComponent>
    <xwmf:isView>wml</xwmf:isView>
    <xwmf:hasPart>
     <Seq>
       <li resource="&gwad;documentBegin"/>
       <li resource="#listOfPublicationWml"/>
       <li resource="&gwad;documentEnd"/>
     </Seq>
    </xwmf:hasPart>
   </pwas:IndexOfPublicationWML>

</rdf:RDF>
```

9.9.7 publication.was

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xwmf 'file:./htdocs/descr/woc.s#'>
 <!ENTITY dc   'file:./htdocs/descr/dc.s#'>
]>

<rdf:RDF
      xmlns="&rdf;"
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:xwmf="&xwmf;"
      xmlns:dc="&dc;"
      xmlns:pwas="this#">
```

```
   <rdfs:Class rdf:ID="Publication">
    <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
   </rdfs:Class>

   <xwmf:Simplexon rdf:ID="PublicationHTML">
   <rdfs:subClassOf rdf:resource="#Publication"/>
    <xwmf:view>html</xwmf:view>
    <xwmf:template rdf:parseType="Literal"><![CDATA[
     <p>
     <UL>
     <LI><b><var>dc:creator</var></b>:
     <var>dc:title</var>,
     <ifp><var>dc:coverage</var>,</ifp>
     <ifp><var>dc:publisher</var>,</ifp>
     <var>dc:date</var>.
     <ifp>[<A HREF="<var>dc:identifier</var>">
           <var>dc:format</var></A>]</ifp><p>
     </UL>
     ]]>
   </xwmf:template>
  </xwmf:Simplexon>

  <xwmf:Simplexon rdf:ID="PublicationWML">
   <rdfs:subClassOf rdf:resource="#Publication"/>
    <xwmf:view>wml</xwmf:view>
    <xwmf:template rdf:parseType="Literal">
     <![CDATA[
         <p>
         <var>dc:creator</var>,<br/>
         <var>dc:title</var>,<br/>
         <var>dc:coverage</var>,<br/>
         <var>dc:date</var>.</p>
     ]]>
   </xwmf:template>
   </xwmf:Simplexon>

  <rdfs:Class rdf:ID="IndexOfPublicationHTML">
   <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="IndexOfPublicationWML">
   <rdfs:subClassOf rdf:resource="&xwmf;Component"/>
  </rdfs:Class>

</rdf:RDF>
```

9.9.8 cm.s

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

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>

<rdf:RDF
      xmlns:rdf="&rdf;"
      xmlns:rdfs="&rdfs;"
      xmlns:cm="this#">

 <rdf:Description rdf:ID="expires">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="&rdfs;Class"/>
  <rdfs:range  rdf:resource="&rdfs;Literal"/>
  <rdfs:comment>
   - used to specify the expireation date of information
   - format YYYY-MM-DD'T'HH:MM:SS'Z'
  </rdfs:comment>
 </rdf:Description>

</rdf:RDF>
```

9.9.9 dc.s

```
<?xml version="1.0" ?>
 <rdf:RDF
      xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<!--                dublin core element set
See http://dublincore.org/documents/1999/07/02/dces/# I wonder that
there is no RDF schema for the dublin core element set defined by the
Dublin Core Metadata Initiative. Is there an official RDF Schema
expressing the dublin core element set?
http://www.dublincore.org/documents/2000/11/dcmes-xml/ contains a DTD
designed with the goal to make the encoding also be valid RDF. For our
purposes we need a pure RDF schema, that can be feed up to an RDF
parser.

There is a draft created by the DC Architecture Working Group
"Expressing Qualified Dublin Core in RDF/Draft/Version-2001-3-30" see
http://www.mathematik.uni-osnabrueck.de/projects/dcqual/qual21.3.1/
```

```
that extends the dublin core element set.

However for our purposes we want to use the basic dublin core element
set.  In the following there are an RDF schema descriptions expressing
the dublin core elements with RDF properties as suggested in the
various DC drafts.

For more context informations about the creation an RDF schema for DC
see:
http://lists.w3.org/Archives/Public/www-rdf-interest/2001Mar/0249.html
http://lists.w3.org/Archives/Public/www-rdf-interest/2001Mar/0248.html
http://dublincore.org/documents/#proposedrecommendations

                Comments concerning this RDF-schema please to:
                      Reinhold.Klapsing@uni-essen.de
-->

<rdf:Property rdf:ID="title">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
    Definition: A name given to the resource.
    Comment:    Typically, a Title will be a name by which the resource is
                formally known.
 </rdfs:comment>
</rdf:Property>


<rdf:Property rdf:ID="creator">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: An entity primarily responsible for making the content of
               the resource.
   Comment:    Examples of a Creator include a person, an organisation,
               or a service. Typically, the name of a Creator should be
               used to indicate the entity.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="subject">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: The topic of the content of the resource.
   Comment:    Typically, a Subject will be expressed as keywords,
               key phrases or classification codes that describe a topic
               of the resource.
               Recommended best practice is to select a value from a
               controlled vocabulary or formal classification scheme.
 </rdfs:comment>
```

```
</rdf:Property>

<rdf:Property rdf:ID="description">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: An account of the content of the resource.
   Comment:    Description may include but is not limited to: an abstract,
               table of contents, reference to a graphical representation
               of content or a free-text account of the content.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="publisher">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: An entity responsible for making the resource available
   Comment:    Examples of a Publisher include a person, an organisation,
               or a service.
               Typically, the name of a Publisher should be used to
               indicate the entity.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="contributor">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: An entity responsible for making contributions to the
               content of the resource.
   Comment:    Examples of a Contributor include a person, an organisation,
               or a service.
               Typically, the name of a Contributor should be used to
               indicate the entity.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="date">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: A date associated with an event in the life cycle of the
               resource.
   Comment:    Typically, Date will be associated with the creation or
               availability of the resource.  Recommended best practice
               for encoding the date value is defined in a profile of
               ISO 8601 [W3CDTF] and follows the YYYY-MM-DD format.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="type">
```

```
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: The nature or genre of the content of the resource.
   Comment:    Type includes terms describing general categories, functions,
               genres, or aggregation levels for content. Recommended best
               practice is to select a value from a controlled vocabulary
               (for example, the working draft list of Dublin Core Types
               [DCT1]). To describe the physical or digital manifestation
               of the resource, use the FORMAT element.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="format">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: The physical or digital manifestation of the resource.
   Comment:    Typically, Format may include the media-type or dimensions of
               the resource. Format may be used to determine the software,
               hardware or other equipment needed to display or operate the
               resource. Examples of dimensions include size and duration.
               Recommended best practice is to select a value from a
               controlled vocabulary (for example, the list of Internet Media
               Types [MIME] defining computer media formats).
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="identifier">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: An unambiguous reference to the resource within a given
               context.
   Comment:    Recommended best practice is to identify the resource by means
               of a string or number conforming to a formal identification
               system.
               Example formal identification systems include the Uniform
               Resource Identifier (URI) (including the Uniform Resource
               Locator (URL)), the Digital Object Identifier (DOI) and the
               International Standard Book Number (ISBN).
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="source">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: A Reference to a resource from which the present resource
               is derived.
   Comment:    The present resource may be derived from the Source resource
               in whole or in part.  Recommended best practice is to
```

```
               reference the resource by means of a string or number
               conforming to a formal identification system.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="language">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
  Definition: A language of the intellectual content of the resource.
   Comment:    Recommended best practice for the values of the Language
               element is defined by RFC 1766 [RFC1766] which includes
               a two-letter Language Code (taken from the ISO 639
               standard [ISO639]), followed optionally, by a two-letter
               Country Code (taken from the ISO 3166 standard [ISO3166]).
               For example, 'en' for English, 'fr' for French, or
               'en-uk' for English used in the United Kingdom.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="relation">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: A reference to a related resource.
   Comment:    Recommended best practice is to reference the resource by means
               of a string or number conforming to a formal identification
               system.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="coverage">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
   Definition: The extent or scope of the content of the resource.
   Comment:    Coverage will typically include spatial location (a place name
               or geographic coordinates), temporal period (a period label,
               date, or date range) or jurisdiction (such as a named
               administrative entity).
               Recommended best practice is to select a value from a
               controlled vocabulary (for example, the Thesaurus of Geographic
               Names [TGN]) and that, where appropriate, named places or time
               periods be used in preference to numeric identifiers such as
               sets of coordinates or date ranges.
 </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="rights">
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 <rdfs:comment>
```

```
   Definition: Information about rights held in and over the resource.
   Comment:    Typically, a Rights element will contain a rights
               management statement for the resource, or reference
               a service providing such information. Rights information
               often encompasses Intellectual Property Rights (IPR),
               Copyright, and various Property Rights.
               If the Rights element is absent, no assumptions can be made
               about the status of these and other rights with respect to
               the resource.
 </rdfs:comment>
</rdf:Property>
</rdf:RDF>
```

9.10 XRDF-Datenmodell

Sei A ein geeignet gewähltes Alphabet.[1] Sei A^* die Menge der Strings, die über dem Alphabet A definiert ist. Die folgende Grammatik definiert Ausdrücke der Form R über A^* rekursiv:

$$R \quad ::= \quad r \quad | \quad '(' \, R \, ',' \, R \, ')' \quad | \quad '[' \, R \, ',' \, R \, ',' \, R \, ']'$$

Wobei r Elemente von A^* bezeichnet. Ein (Sub-)Ausdruck dieser Form wird *Atom* genannt. Ein (Sub-)Ausdruck der Form R wird *Resource* genannt. Ein (Sub-)Ausdruck der Form R, welcher mit dem Muster $[R, R, R]$ übereinstimmt, wird *Statement* genannt. Ein (Sub-)Ausdruck der Form R, welcher mit dem Muster (R, R) übereinstimmt, wird *Liste* genannt. Es werden nur endliche Mengen von endlichen Ausdrücken betrachtet.

9.11 XRDF-Syntax

```
<!ELEMENT statement (subject, predicate, object)>
<!ELEMENT list      (statement | atom | list)+>
<!ELEMENT atom      (#PCDATA)>
<!ELEMENT subject   (atom|statement|list)>
<!ELEMENT predicate (atom|statement|list)>
<!ELEMENT object    (atom|statement|list)>
```

[1] Unter einem geeigneten Alphabet wird hier ein Alphabet verstanden, daß es ermöglicht URIs, Literale und XML-Dokumente auszudrücken: z.B. das im XML-Standard – Abschnitt *Character Classes* gezeigte Alphabet.

Literaturverzeichnis

[1] Apache Software Foundation: *Apache Tcl Project*. http://tcl.apache.org/tcl.ttml.

[2] Apache Software Foundation: *Apache XML Project*. http://xml.apache.org/.

[3] Apache Software Foundation: *The Apache/Perl Integration Project*. http://perl.apache.org/.

[4] Apache Software Foundation: *The Jakarta Project*. http://jakarta.apache.org/.

[5] Bakken, S. S., A. Aulbach, E. Schmid, J. Winstead, L. T. Wilson, R. Lerdorf, A. Zmievski und J. Ahto: *PHP Handbuch*. http://www.php.net/manual/de/.

[6] Becker, J., W. König, R. Schütte, O. Wendt und S. Zelewski: *Wirtschaftsinformatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven*. 1999. http://www-wi.uni-muenster.de/is/Tagung/wiss97/11Okt.htm.

[7] Beckett, D. und B. McBride: *RDF/XML Syntax Specification (Revised)*. Working Draft, W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-syntax-grammar-20021108.

[8] Bergmann, E. und H. Noll: *Mathematische Logik mit Informatik-Anwendungen*. Springer-Verlag, Berlin, 1977.

[9] Berners-Lee, T.: *The World-Wide Web*. Communications of the ACM, 37(8):76–82, August 1994. http://www.acm.org/pubs/articles/journals/cacm/1994-37-8/p76-berners-lee/p76-berners-lee.pdf.

[10] Berners-Lee, T.: *Semantic Web Road Map*. Personal Note, W3C, September 1998. http://www.w3.org/DesignIssues/Semantic.html.

[11] Berners-Lee, T.: *Why RDF model is different from the XML model*. Personal Note, W3C, October 1998. http://www.w3.org/DesignIssues/RDF-XML.html.

[12] Berners-Lee, T.: *A strawman Unstriped syntax for RDF in XML*. Personal Note, W3C, May 1999. http://www.w3.org/DesignIssues/Syntax.html.

[13] Berners-Lee, T.: *RDFS bug "A property can have at most one range property"*. http://lists.w3.org/Archives/Public/www-rdf-interest/2000Sep/0107.html, October 2000.

[14] Berners-Lee, T.: *Notation 3: Ideas about Web Architecture - yet another notation*. Personal Note, W3C, November 2001. http://www.w3.org/DesignIssues/Notation3.

[15] Berners-Lee, T.: *Primer: Getting into RDF & Semantic Web using N3*. Personal Note, W3C, November 2001. http://www.w3.org/2000/10/swap/Primer.html.

[16] Berners-Lee, T., D. Connolly und R. R. Swick: *Web Architecture: Describing and Exchanging Data*. Note, W3C, June 1999. http://www.w3.org/1999/04/WebData.

[17] Berners-Lee, T., R. Fielding und L. Masinter: *Uniform Resource Identifiers (URI): Generic Syntax*. RFC, Category: Standards Track, IETF, August 1998. http://www.ietf.org/rfc/rfc2396.txt.

[18] Berners-Lee, T., L. Masinter und M. McCahill: *Uniform Resource Locators (URL)*. RFC, Category: Standards Track, IETF, December 1994. http://www.ietf.org/rfc/rfc1738.txt.

[19] Bernes-Lee, T., J. Hendler und O. Lassila: *The Semantic Web*. Scientific American, 2001. http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.

[20] Biron, P. V. und A. Malhotra: *XML Schema Part 2: Datatypes*. Recommendation, W3C, May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[21] Block, A.: *Entwurf und Implementierung eines objektorientierten Klassensystems zur Verarbeitung von RDF-Modellen*. Diplomarbeit, Universität Essen, 2000.

[22] Booch, G., J. Rumbaugh und I. Jacobson: *Das UML-Benutzerhandbuch*. Addison-Wesley, 1999.

[23] Bos, B., H. W. Lie, C. Lilley und I. Jacobs: *Cascading Style Sheets, level 2*. Recommendation, W3C, May 1998. http://www.w3.org/TR/1998/REC-CSS2-19980512.

[24] Bray, T., D. Hollander und A. Layman: *Namespaces in XML*. Recommendation, W3C, 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/.

[25] Bray, T., J. Paoli, C. M. Sperberg-McQueen und E. Maler: *Extensible Markup Language (XML) 1.0 (Second Edition)*. Recommendation, W3C, October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[26] Brickley, D.: *RDF: Understanding the Striped RDF/XML Syntax*. Personal Note, W3C, November 2001. http://www.w3.org/2001/10/stripes/.

[27] Brickley, D. und R. Guha: *Resource Description Framework (RDF) Schema Specification 1.0*. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[28] Brickley, D., R. Guha und B. McBride: *RDF Vocabulary Description Language 1.0: RDF Schema*. Working Draft, W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-schema-20021112/.

[29] Broekstra, J., M. Klein, D. Fensel, S. Decker und I. Horrocks: *OIL: a case-study in extending RDF-Schema*. Techn. Ber., Ontoknowledge, 2000. http://www.ontoknowledge.org/oil/oil-rdfs.pdf.

[30] Bush, V.: *As We May Think*. The Atlantic Monthly, 176(1):101–108, July 1945.

[31] Carroll, J. J.: *Unparsing RDF/XML*. In: *Proceedings of the Eleventh International World Wide Web Conference*. Hewlett-Packard Labs, May 2002. http://www2002.org/CDROM/refereed/184/index.html.

[32] Champin, P.-A.: *RDF Tutorial*, March 2000. http://www710.univ-lyon1.fr/~champin/rdf-tutorial/rdf-tutorial.ps.gz.

[33] Clark, J.: *Comparison of SGML and XML*. Note, W3C, December 1997. http://www.w3.org/TR/NOTE-sgml-xml-971215.

[34] Clark, J.: *XSL Transformations (XSLT) - Version 1.0*. Recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[35] Clark, J. und S. DeRose: *XML Path Language (XPath) Version 1.0*. Recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[36] Coar, K. und D. R. Robinson: *The WWW Common Gateway Interface Version 1.1*. INTERNET-DRAFT, IETF, June 1999. http://Web.Golux.Com/coar/cgi/draft-coar-cgi-v11-03.html.

[37] Coda, F., C. Ghezzi, G. Vigna und F. Garzotto: *Towards a Software Engineering Approach to Web Site Development*. In: *9th International Workshop on Software Specification and Design (IWSSD)*. IEEE, April 1998.

[38] Conen, W. und R. Klapsing: *Web-based RDF Schema Explorer*. http://wonkituck.wi-inf.uni-essen.de/rdfs.html.

[39] Conen, W. und R. Klapsing: *A Logical Interpretation of RDF*. Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, 5(13), December 2000. http://www.ep.liu.se/ea/cis/2000/013/.

[40] Conen, W. und R. Klapsing: *Announcement of Discussion-Paper: A Logical Interpretation of RDF*. http://lists.w3.org/Archives/Public/www-rdf-interest/2000Aug/0122.html, August 2000.

[41] Conen, W. und R. Klapsing: *Exchanging Semantics with RDF*. In: Buhl, H. U. und others (Hrsg.): *Information Age Economy: 5. Internationale Tagung Wirtschaftsinformatik*, September 2001. ISBN 3-7908-1427-X.

[42] Conen, W. und R. Klapsing: *Logical Interpretations of RDFS - A Compatibility Guide*. Techn. Ber., University of Essen, November 2001. http://nestroy.wi-inf.uni-essen.de/rdf/new_interpretation/.

[43] Conen, W. und R. Klapsing: *Utilizing Host Formalisms to Extend RDF Semantics*. In: *Proceedings of the Semantic Web Working Symposium (SWWS)*, Stanford, August 2001. http://www.semanticweb.org/SWWS/program/full/paper24.pdf.

[44] Conen, W. und R. Klapsing: *XML - Komponenten in der Praxis*, Kap. RDF – Grundlage des Semantic Web, S. 337–380. Xpert.press. Springer, ISBN 3-540-44046-1, 2003.

[45] Conen, W., R. Klapsing und E. Köppen: *XRDF - an eXtensible Resource Description Framework, Discussion Paper*. Techn. Ber., Universität Essen, 2001. http://nestroy.wi-inf.uni-essen.de/rdf/xrdf/.

[46] Conen, W., R. Klapsing und E. Köppen: *The Emerging Semantic Web: Selected papers from the First Semantic Web Working Symposium*, Kap. RDF M&S revisited: From reification to nesting, from containers to lists, from dialect to pure XML. Frontiers in Artificial Intelligence and Applications. IOS press, 2002. ISBN 1-58603-255-0.

[47] Connolly, D.: *Transforming RDF with XSLT*. Personal Note, W3C, 2001. http://www.w3.org/XML/2000/04rdf-parse/.

[48] Cowan, J. und R. Tobin: *XML Information Set*. Recommendation, W3C, October 2001. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[49] Cranefield, S.: *UML and the Semantic Web*. In: *Proceedings of the Semantic Web Working Symposium (SWWS)*, Stanford, August 2001. http://www.semanticweb.org/SWWS/program/full/paper1.pdf.

[50] Cruz, I., S. Decker, J. Euzenat und D. McGuinness (Hrsg.): *The Emerging Semantic Web*. Frontiers in Artificial Intelligence and Applications. IOS press, 2002. ISBN 1-58603-255-0.

[51] *The DARPA Agent Markup Language*. http://www.ontoknowledge.org/.

[52] Decker, S., D. Brickley, J. Saarela und J. Angele: *A Query and Inference Service for RDF*. In: *Online Proceedings of the QL'98 - The Query Languages Workshop*. http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[53] Decker, S., S. Melnik, F. V. Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann und I. Horrocks: *The Semantic Web: The Roles of XML and RDF*. IEEE Internet Computing, S. 63–74, September 2000.

[54] DeRose, S.: *XML XLink Requirements Version 1.0*. Note, W3C, 1999. http://www.w3.org/TR/1999/NOTE-xlink-req-19990224/.

[55] DeRose, S., E. Maler und J. David Orchard: *XML Linking Language (XLink) Version 1.0*. Recommendation, W3C, June 2001. http://www.w3.org/TR/2000/REC-xlink-20010627/.

[56] DeRose, S., E. Maler und R. D. Jr.: *XML Pointer Language (XPointer) Version 1.0*. Candidate Recommendation, W3C, September 2001. http://www.w3.org/TR/2001/CR-xptr-20010911/.

[57] Diaz, A., J. Ferraiolo, S. Kulseth, P. L. Hégaret, C. Lilley, C. McCathieNevile, T. Roy und R. Whitmer: *Component Extension (CX) API Requirements Version 1.0*. Note, W3C, December 2001. http://www.w3.org/TR/CX.

[58] Doan, A., J. Madhavan, P. Domingos und A. Halevy: *Learning to Map between Ontologies on the Semantic Web*. In: *Proceedings of the Eleventh International World Wide Web Conference*. University of Washington, May 2002. http://www2002.org/CDROM/refereed/232/index.html.

[59] Eberhart, A.: *Survey of RDF data on the Web*. Techn. Ber., International University in Germany, August 2002. http://www.i-u.de/schools/eberhart/rdf/rdf-survey.pdf.

[60] ECMA: *ECMAScript Language Specification*. Standard ECMA-262, August 1998. http://www.ecma.ch/stand/ECMA-262.htm.

[61] Fallside, D. C.: *XML Schema Part 0: Primer*. Recommendation, W3C, MayMay 2001. http://www.w3.org/TR/xmlschema-0/.

[62] Fensel, D.: *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, Heidelberg, 2001.

[63] Fielding, R.: *Relative Uniform Resource Locators*. RFC, Category: Standards Track, IETF, June 1995. http://www.ietf.org/rfc/rfc1808.txt.

[64] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee: *Hypertext Transfer Protocol – HTTP/1.1*. RFC, Category: Standards Track, IETF, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[65] Frank, U.: *Zur Verwendung formaler Sprachen in der Wirtschaftsinformatik: Notwendiges Merkmal eines wissenschaftlichen Anspruchs oder Ausdruck eines übertriebenen Szientismus?*. In: Becker, König, Schütte, Wendt und Zelewski (Hrsg.): *Wirtschaftsinformatik und*

Wissenschaftstheorie: Bestandsaufnahme und Perspektiven, S. 128–158. Gabler, 1999.

[66] Freed, N. und N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC, Category: Standards Track, IETF, November 1996. http://www.ietf.org/rfc/rfc2046.txt.

[67] Freed, N., J. Klensin und J. Postel: *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC, Category: Best Current Practice, IETF, November 1996. http://www.ietf.org/rfc/rfc2048.txt.

[68] Garzotto, F., P. Paolini und D. Schwabe: *HDM – A Model for the Design of Hypertext Applications*. In: *Hypertext 1991*, S. 313–328. ACM, December 1991.

[69] Garzotto, F., P. Paolini und D. Schwabe: *HDM – A Model-Based Approach to Hypertext Application Design*. ACM Transactions on Information Systems, 11(1):1–26, January 1993. http://www.acm.org/pubs/articles/journals/tois/1993-11-1/p1p45-garzotto/p1-garzotto.pdf.

[70] Gellersen, H.-W., R. Wicke und M. Gaedke: *WebComposition: an object-oriented support system for the Web Engineering Lifecycle*. In: *6th International World-Wide Web Conference*, Bd. 29 d. Reihe *Computer Networks and ISDN Systems*, S. 1429–1437. IW3C2, April 1997.

[71] Grant, J., D. Beckett und B. McBride: *RDF Test Cases*. Working Draft, W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-testcases-20021112/.

[72] Guha, R., O. Lassila, E. Miller und D. Brickley: *Enabling Inferencing*. In: *W3C Query Languages Workshop*, 1998. http://www.ilrt.bris.ac.uk/discovery/rdf-dev/purls/papers/QL98-enabling/.

[73] Halasz, F. und M. Schwartz: *The Dexter Hypertext Reference Model*. Communications of the ACM, 37(2):30–39, February 1994.

[74] Harmelen, F. van und I. Horrocks: *DAML+OIL ontology markup language*. Techn. Ber., DARPA, 2001.

[75] Harold, E. R.: *XML Bible, Second Edition, Chapter: XSL Formatting Objects*. http://www.ibiblio.org/xml/books/bible2/chapters/ch18.html, 2001.

[76] Hayes, P.: *RDF Model Theory*. Editor's Working Draft (updates W3C Public Working Draft - 20010925), W3C, November 2001. http://nestroy.wi-inf.uni-essen.de/rdf/working-draft-mt-retrived-05-11-2001-12:00-GMT.ps.

[77] Hayes, P. und B. McBride: *RDF Semantics*. Working Draft, W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-mt-20021112/.

[78] Hayes, P., S. Melnik und P. Stickler: *RDF Datatyping*. Working Draft, W3C, April 2002. http://www-nrc.nokia.com/sw/rdf-datatyping.html.

[79] Heflin, J., R. Volz und J. Dale: *Requirements for a Web Ontology Language*. Working Draft, W3C, March 2002. http://www.w3.org/TR/2002/WD-webont-req-20020307/.

[80] Hermes, H.: *Einführung in die mathematische Logik*. Teubner, Stuttgart, 1976.

[81] Holl, A.: *Empirische Wirtschaftsinformatik und Erkenntnistheorie*. In: Becker, König, Schütte, Wendt und Zelewski (Hrsg.): *Wirtschaftsinformatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven*, S. 163–207. Gabler, 1999.

[82] HTML Working Group: *XHTML[tm] 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0*. Recommendation, W3C, January 2000. http://www.w3.org/TR/2000/REC-xhtml1-20000126.

[83] HTML Working Group: *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) - A Reformulation of HTML 4 in XML 1.0*. Draft, W3C, October 2001. http://www.w3.org/TR/2001/WD-xhtml1-20011004.

[84] IEEE: *Software Engineering Standards Collection*. April 1999. ISBN 1-7381-1563-0.

[85] IEEE: *Web Engineering Part 1*. January-March 2001. http://computer.org/multimedia/mu2001/u1toc.htm.

[86] IEEE: *Web Engineering Part 2*. April-June 2001. http://computer.org/multimedia/mu2001/u2toc.htm.

[87] IEEE – Internet Best Practices Working Group: *Recommended Practice for Internet Practices, Web Page Engineering, Intranet/Extranet Applications*. June 1999. ISBN 0-7381-1652-1.

[88] International Organization for Standardization. http://www.iso.ch/.

[89] International Organization for Standardization: *ISO/IEC 10744-1992, Information Technology -Hypermedia/Time-based Structuring Language (HyTime).* http://www.y12.doe.gov/sgml/wg8/document/1920.htm.

[90] International Organization for Standardization: *Information Processing – Text and Office systems – Standard Generalized Markup Language (SGML)*. ISO 8879, 1986. http://www.iso.ch/cate/d16387.html.

[91] Internet Engineering Task Force. http://www.ietf.org.

[92] Isakovitz, T., E. A. Stohr und P. Balasubramanian: *RMM: A Methodology for Structured Hypermedia Design*. Communications of the ACM, 38(8):34–44, August 1995.

[93] Karvounarakis, G., S. Alexaki, V. Christophides, D. Plexousakis und M. Scholl.: *RQL: A Declarative Query Language for RDF*. In: *Proceedings of The Eleventh International World Wide Web Conference (WWW'02)*, May 2002.

[94] Klapsing, R.: *Discussions about the Logical Interpretation of RDF*. http://nestroy.wi-inf.uni-essen.de/rdf/logical_interpretation/discussion.html, October 2000.

[95] Klapsing, R. und G. Neumann: *Applying the Resource Description Framwork to Web Engineering*. In: *Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies: EC-Web 2000*, LNCS. Springer, September 2000. http://nestroy.wi-inf.uni-essen.de/xwmf/paper/xwmf_EcWeb/.

[96] Klapsing, R., G. Neumann und W. Conen: *Semantics in Web Engineering: Applying the Resource Description Framework*. IEEE MultiMedia, 8(2):62–68, April-June 2001. http://computer.org/multimedia/mu2001/u2toc.htm#u2062.

[97] Klyne, G., J. Carroll und B. McBride: *Resource Description Framework (RDF): Concepts and Abstract Data Model*. Working Draft,

W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/.

[98] König, W., A. Heinzl und A. v. Poplotzki: *Die zentralen Forschungsgegenstände der Wirtschaftsinformatik in den nächsten zehn Jahren*. Wirtschaftsinformatik, (6):558–569, 1995.

[99] König, W., A. Heinzl, M. Rumpf und A. v. Poplotzki: *Kombinierte Delphi-AHP Studie: Die zentralen Forschungsmethoden und Theoriekerne der Wirtschaftsinformatik in den nächsten zehn Jahren*. In: Heilmann, H., L. Heinrich und F. Roithmayr (Hrsg.): *Information Engineering*. Oldenbourg, 1996.

[100] Kuhn, T.: *Die Struktur wisschenschaftlicher Revolution*. Frankfurt, 2 Aufl., 1976.

[101] Lassila, O. und R. R. Swick: *Resource Description Framework (RDF) Model and Syntax Specification*. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[102] Lehner, F.: *Theoriebildung in der Wirtschaftsinformatik*. In: Becker, König, Schütte, Wendt und Zelewski (Hrsg.): *Wirtschaftsinformatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven*, S. 6–24. Gabler, 1999.

[103] Lehner, F., K. Hildebrand und R. Maier: *Wirtschaftsinformatik: Theoretische Grundlagen*. Hanser, 1995.

[104] Lie, H. W. und B. Bos: *Cascading Style Sheets, level 1*. Recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-CSS1-19990111.

[105] Lloyd, J.: *Foundations of Logic Programming*. Springer Verlag, 2 Aufl., 1987.

[106] *Mailinglist (ML) rdf-comments*. http://lists.w3.org/Archives/Public/www-rdf-comments.

[107] *ML rdf-interest*. http://lists.w3.org/Archives/Public/www-rdf-interest.

[108] *ML rdf-logic*. http://lists.w3.org/Archives/Public/www-rdf-logic.

[109] *ML rdfcore-wg*. http://lists.w3.org/Archives/Public/w3c-rdfcore-wg.

[110] *ML webont-wg*. http://lists.w3.org/Archives/Public/www-webont-wg.

[111] *ML rdf-dev*. http://www.mailbase.ac.uk/lists/rdf-dev/archive.html.

[112] Malhotra, A. und N. Sundaresan: *RDF Query Specification*. In: *W3C Query Languages Workshop*, 1998. http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html.

[113] Manola, F., E. Miller und B. McBride: *RDF Primer*. Working Draft, W3C, November 2002. http://www.w3.org/TR/2002/WD-rdf-primer-20021111/.

[114] Marchiori, M. und J. Saarela: *Query + Metadata + Logic = Metalog*. In: *Online Proceedings of the QL'98 - The Query Languages Workshop*. http://www.w3.org/TandS/QL/QL98/pp/metalog.html.

[115] Masinter, L.: *The 'data' URL scheme*. Techn. Ber., Xerox Corporation, August 1998. ftp://ftp.isi.edu/in-notes/rfc2397.txt.

[116] Maurer, H., L. Gardiner, S. Murugesan, Y. Deshpande, A. Ginige, S. Hansen, S. Kankanahalli, R. J. Paul, G. Salter, S. Simoff und S. Venkatesh: *First International Workshop on Web Engineering WebE 98: Seventh International World Wide Web Conference*. http://fistserv.macarthur.uws.edu.au/san/webe98/proceedings.htm, April 1998.

[117] McBride, B. und D. Beckett: *A Proposed Interpretation of RDF Containers*. Draft, HP Labs/University of Bristol, 2000. http://www.hpl.hp.co.uk/people/bwm/rdf/issues/containersyntax/current.htm.

[118] McDermott, D., J. Borden, M. Burstein, D. Smith und R. Waldinger: *A Proposal for Encoding Logic in RDF/DAML*. Techn. Ber., Yale University, 2001.

[119] Melnik, S.: *Algebraic Specification for RDF Models*. http://www-diglib.stanford.edu/diglib/ginf/WD/rdf-alg/rdf-alg.ps, July 1999.

[120] Melnik, S.: *Bridging the Gap between RDF and XML*. Techn. Ber., Stanford University, December 1999. http://www-db.stanford.edu/~melnik/rdf/fusion.html.

[121] Melnik, S.: *Simplified Syntax for RDF*. Techn. Ber., Stanford University, December 1999. http://www-db.stanford.edu/~melnik/rdf/syntax.html.

[122] Mendelson, E.: *Introduction to Mathematical Logic*. Wadsworth & Brooks, Pacific Grove/Cal., 1987.

[123] Microsoft Corporation: *Microsoft Active Server Pages: Frequently Asked Questions*. http://msdn.microsoft.com/library/default.asp ?url=/library/en-us/dnasp/html/msdn_aspfaq.asp.

[124] Moats, R.: *URN Syntax*. Techn. Ber., IETF, May 1997. http://www.ietf.org/rfc/rfc2396.txt.

[125] Murata, M., S. St.Laurent und D. Kohn: *XML Media Types*. RFC, Category: Standards Track, IETF, January 2001. http://www.ietf.org/rfc/rfc3023.txt.

[126] Murugesan, S. und Y. Deshpande: *Web Engineering*. In: *International Conference on Software Engineering (ICSE´99)*, S. 693–694. ACM, May 1999.

[127] NCSA HTTPd Development Team: *Server Side Includes (SSI)*. http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html.

[128] Nejdl, W., H. Dhraief und M. Wolpers: *O-Telos-RDF: A Resource Description Format with Enhanced Meta-Modeling Functionalities based on O-Telos*. In: *Proceedings of the Workshop on Knowledge Markup and Semantic Annotation at the First International Conference on Knowledge Capture (K-CAP'2001)*, Canada, October 2001. http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2001/kcap01_final.pdf.

[129] Nejdl, W., M. Wolpers und C. Capelle: *The RDF Schema Specification Revisited*. In: *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Beiträge des Workshops Modellierung 2000*, Bd. 15 d. Reihe *Koblenzer Schriften zur Informatik*, 2000. http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2000/modeling2000/wolpers.pdf.

[130] Neumann, G.: *Meta-Programmierung und Prolog*. Addison-Wesley, 1988.

[131] Neumann, G. und S. Nusser: Wafe- *An X Toolkit Based Frontend for Application Programs in Various Programming Languages*. In: *USENIX Winter 1993 Technical Conference*, January 1993.

[132] Neumann, G. und U. Zdun: *xoRDF*. http://nestroy.wi-inf.uni-essen.de/xoRDF-0.9b.tar.gz.

[133] Neumann, G. und U. Zdun: *XOTcl, an Object-Oriented Scripting Language*. In: *Proceedings of 7th Usenix Tcl/Tk Conference (Tcl2k)*, February 2000.

[134] Olson, M.: *RDF Inference Language (RIL)*. http://xml.coverpages.org/RIL-20010510.html, 2001.

[135] *On-To-Knowledge: Content-driven Knowledge-Management through Evolving Ontologies*. http://www.ontoknowledge.org/.

[136] *OntoWeb-Home-Page*. http://www.ontoweb.org/.

[137] Patel-Schneider, P. und J. Siméon: *The Yin/Yang Web: XML Syntax and RDF Semantics*. In: *Proceedings of the Eleventh International World Wide Web Conference*, May 2002. http://www2002.org/CDROM/refereed/231/index.html.

[138] Popper, K. R.: *Die beiden Grundprobleme der Erkenntnistheorie*. Mohr, 1994, Wiederauflage.

[139] Popper, K. R.: *Logik der Forschung*. Mohr, 1994, Wiederauflage.

[140] Postel, J.: *Internet Protocol: DARPA Internet Program - Protocol Specification*. RFC, IETF, Septermber 1981. http://www.ietf.org/rfc/rfc791.txt.

[141] Postel, J.: *Transmission Control Protocol: DARPA Internet Program - Protocol Specification*. RFC, IETF, September 1981. http://www.ietf.org/rfc/rfc793.txt.

[142] Postel, J. und J. Reynolds: *File Transfer Protocol (FTP)*. RFC, IETF, October 1985. http://www.ietf.org/rfc/rfc2616.txt.

[143] *PrologHttp*. http://gollem.swi.psy.uva.nl/twiki/pl/bin/view/Library/PrologHttp.

[144] Raggett, D., A. L. Hors und I. Jacobs: *HTML 4.01 Specification*. Recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-html401-19991224.

[145] Rayavarapu, S.: *W3C Query Languages*. Techn. Ber., W3C, January 2001. http://www1.coe.neu.edu/~srayavar/W3CQL/ql.htm.

[146] *Online RDF-to-XRDF Converter*. http://wonkituck.wi-inf.uni-essen.de/xrdf.html.

[147] RDFCore Working Group. http://www.w3.org/2001/sw/RDFCore/.

[148] *Semantic Web Activity: RDF Interest Group*. http://www.w3.org/RDF/Interest/, 2001. W3C.

[149] Reimer, U.: *Einführung in die Wissensrepräsentation*. Leitfäden der angewandten Informatik. Teubner, 1991.

[150] Robinson, J.: *A Maschine-Logic Based on the Resolution Principle*. JACM, 12(1):23–41, 1965.

[151] Rurik, G.: *Wege zur Wissenschaftstheorie: Begleitmaterial zur Vorlesung Einführung in die Wissenschaftstheorie*. http://www.fb12.uni-dortmund.de/wtheorie/JPEG/GLOSSAR.HTM.

[152] R.V.Guha und P. Hayes: *LBase: Semantics for Languages of the Semantic Web*. NOT-A-Note, W3C, November 2002. http://www.coginst.uwf.edu/~phayes/Lbase_on_Friday.html.

[153] Schöning, U.: *Logik für Informatiker*. BI Wissenschaftsverlag, Mannheim, 1989.

[154] Schwabe, D., G. Rossi und S. D. Barbosa: *Systematic Hypermedia Application Design with OOHDM*. In: *Hypertext 1996*. ACM, 1996.

[155] Schwabe, D., G. Rossi, M. Gaedke und H.-W. Gellersen: *Thirty-third Annual Hawai'i International Conference on Systems Sciences (HICSS): Minitrack Web Engineering (Part of the Internet and the Digital Economy Track)*. http://www.hicss.hawaii.edu/HICSS_33/intmini.htm#webeng, January 2000.

[156] Shapiro, S.: *Classical Logic, In Stanford Encyclopedia of Philosophy*. http://plato.stanford.edu/entries/logic-classical/, 2000.

[157] Shklar, L. A., J. G. Davis, S. Murugesan und C. F. Enguix: *International Workshop Web Engineerung '99: Eighth International World Wide Web Conference*. http://budhi.uow.edu.au/web-engineering99/web_engineering.html, May 1999.

[158] *SiLRI* . http://ontobroker.semanticweb.org/silri/.

[159] *SiRPAC*. http://dev.w3.org/cvsweb/java/classes/org/w3c/rdf/implementation/syntax/sirpac/SiRPAC.java.

[160] Sowa, J. F.: *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[161] Sowa, J. F.: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole/Thomson Learning, 2000.

[162] Sperberg-McQueen, C. M. und L. Burnard: *Guidelines for Electronic Text Encoding and Interchange*, 1994. Text Encoding Initiative, Association for Computers and the Humanities (ACH), Association for Computational Linguistics (ACL), Association for Literary and Linguistic Computing (ALLC).

[163] Staab, S., M. Erdmann und A. Maedche: *Ontologies in RDF(S)*. Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, 6, 2001.

[164] Staab, S., M. Erdmann, A. Mädche und S. Decker: *An extensible approach for Modeling Ontologies in RDF(S)*. In: *Proceedings of ECDL-2000 Workshop SSemantic Web: Models, Architectures and Management*, September 2000. http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/onto-rdfs.pdf.

[165] Stickler, P.: *An Extended Class Taxonomy for Uniform Resource Identifier Schemes*. Techn. Ber., Nokia Research Center, January 2002. http://www-nrc.nokia.com/sw/draft-pstickler-uri-taxonomy-00.html.

[166] Stickler, P.: *The 'auth:' URI Scheme for Hierarchical Authority Identifiers*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-auth-00.html.

[167] Stickler, P.: *The 'qname:' URI Scheme for XML Namespace Qualified Names*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-qname-00.html.

[168] Stickler, P.: *The 'tdl:' URI Scheme for Typed Data Literals*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-tdl-00.html.

[169] Stickler, P.: *The 'uri:' URI Scheme for URI Reification*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-uri-00.html.

[170] Stickler, P.: *The 'voc:' URI Scheme for Vocabulary Terms and Codes*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-voc-00.html.

[171] Stickler, P.: *The 'xmlns:' URI Scheme for XML Namespace Reification and Namespace Prefix Declaration*. Techn. Ber., Nokia Research Center, 2002. http://www-nrc.nokia.com/sw/draft-pstickler-xmlns-00.html.

[172] Sun Microsystems: *Applets*. http://java.sun.com/applets/.

[173] Sun Microsystems: *Java Server Pages – White Paper*. http://java.sun.com/products/jsp/whitepaper.html.

[174] Swick, R. R. und H. S. Thompson: *The Cambridge Communiqué*. Note, W3C, October 1999. http://www.w3.org/TR/1999/NOTE-schema-arch-19991007.

[175] *SWI-Prolog*. http://www.swi-prolog.org/.

[176] Teubner, R. A.: *Wirtschaftsinformatik als Wissenschaft*. http://www-wi.uni-muenster.de/wi/lehre/ewi/ws00-01/WI_Wiss.pdf, 2001.

[177] The Internet Corporation for Assigned Names and Numbers. http://www.icann.org/.

[178] Thompson, H. S., D. Beech, M. Maloney und N. Mendelsohn: *XML Schema Part 1: Structures*. Recommendation, W3C, May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[179] URI Planning Interest Group: *URIs, URLs, and URNs: Clarifications and Recommendations 1.0 – Report from the joint W3C/IETF URI Planning Interest Group*. Techn. Ber., W3C/IETF, September 2001. http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921.

[180] *VisiRDF*. http://swt.wi-inf.uni-essen.de/∼rgoertze/.

[181] W3C: *Naming and Addressing Overview (URIs, URLs, ...)*. http://www.w3.org/Addressing/.

[182] *Semantic Web Activity*. http://www.w3.org/2001/sw.

[183] *Semantic Web Advanced Development*. http://www.w3.org/2000/01/sw/.

[184] W3C: *The Extensible Stylesheet Language (XSL)*. http://www.w3.org/Style/XSL/.

[185] *Web-Ontology (WebOnt) Working Group*. http://www.w3.org/2001/sw/WebOnt/.

[186] *World Wide Web Consortium (W3C)*. http://www.w3.org.

[187] W3C DOM WG: *Document Object Model (DOM) Technical Reports*. http://www.w3.org/DOM/DOMTR.

[188] Walsh, N. und L. Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc., 1999. http://www.docbook.org/.

[189] WAP-Forum: *Wireless Application Protocol, Wireless Markup Language Specification, Version 1.3*. WAP June 2000 Technical specifications, February 2000. http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf.

[190] WAP-Forum: *Wireless Markup Language Version 2.0*. Techn. Ber., June 2001. http://www1.wapforum.org/tech/documents/WAP-238-WML-20010626-p.pdf.

[191] *XPCE*. http://www.swi.psy.uva.nl/projects/xpce/.

[192] *XWMF-Home-Page*. http://nestroy.wi-inf.uni-essen.de/xwmf/.

[193] *XWMF Demonstration Server*. http://willimantic.wi-inf.uni-essen.de/.