SiRDF –
Ein simpleres Datenmodell für RDF bei gleicher Ausdrucksmächtigkeit

Konzeptionelles Modell und Java API

Studienarbeit

Institut für
Angewandte Informatik und Formale Beschreibungsverfahren
Lehrstuhl Studer
Fakultät für Wirtschaftswissenschaften
Universität Karlsruhe (TH)

cand. Inform. Benjamin Heitmann
Matrikel-Nummer: 1102894

Betreuer:
Dipl.-Inform. Max Völkel

verantwortlicher Betreuer:
Prof. Dr. Rudi Studer

Tag der Anmeldung: 23. Juni 2005
Tag der Abgabe: 18. Oktober 2005

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Im Rahmen der Vision des Semantic Web ist das *Resource Description Framework (RDF)* die „Vermittlungsschicht“ zwischen den reinen Daten und der semantischen Interpretation dieser Daten. Das konzeptionelle Modell von RDF ist verglichen mit den darauf aufbauenden Technologien vergleichsweise einfach.

Ein RDF Graph besteht aus Aussagen mit Subjekt, Prädikat und Objekt. Die Spezifikation von RDF enthält jedoch viele formale Details, welche in den bestehenden RDF-APIs ein hohes Maß an Komplexität beim Modellieren eines Sachverhalts in RDF verursachen.

Kritisch für den Übergang vom World Wide Web zum Semantic Web ist jedoch eine hinreichend große Zahl von semantisch annotierten Datenquellen.

1.2 Idee zur Vereinfachung der formalen Modellierung

Abhängig von den kognitiven Leistungen, die für das Modellieren erforderlich sind, kann zwischen zwei Ebenen der Modellierung unterschieden werden. Zuerst muss der Anwender den Sachverhalt in einem mentalen Modell ausdrücken, um zum Beispiel sicher darüber zu sein, das er alle notwendigen Aspekte des Sachverhaltes berücksichtigt hat. Erst danach kann eine Modellierung des Sachverhalts für das Semantic Web erfolgen. Dazu muss das mentale Modell des Anwenders in das formale Modell des Semantic Web transferiert werden. Bisher war RDF das Zielmodell des zweiten Transferschritts, doch die Modellierung in RDF ist teilweise sehr komplex.

Mit einem vereinfachten Modell auf der Basis von RDF, welches jedoch die gleiche Ausdrucksmächtigkeit wie RDF hat, wäre es für den Anwender möglich die Komplexität des Transfers von seinem mentalen Modell in ein formales Modell zu verringern. Wenn das vereinfachte Modell auf RDF aufbaut, und die gleiche semantische Ausdrucksmächtigkeit wie RDF hat, kann nach einer Möglichkeit gesucht werden, das vereinfachten Modell automatisiert in das alte RDF Modell zu konvertieren.

1.3 Aufgabenstellung

Ziel der Arbeit ist das Design eines simpleren Datenmodells für RDF bei gleicher semantischer Ausdrucksmächtigkeit. Das neue Datenmodell soll die Anzahl der verwendeten Entitäten auf ein Mindestmaß reduzieren, um eine leichte Erlernbarkeit zu ermöglichen. Die Probleme, welche am häufigsten beim Modellieren von Sachverhalten in RDF auftreten, sollen analysiert werden und beim Design des neuen Datenmodells und der darauf aufbauenden abstrakten Syntax und formalen Semantik berücksichtigt werden.

Auf diesem neuen Datenmodell soll eine möglichst einfache API in Java implementiert werden. Die API soll alle Vorteile des neuen Datenmodells für den Anwender der API zur Verfügung stellen. Im Rahmen der Implementierung soll auch eine portable Minimal-API entworfen werden. Die Minimal-API soll eine Abstraktionsschicht für die darunter liegenden austauschbaren RDF Implementierungen darstellen und die kleinstmögliche Menge der wichtigsten Funktionen von RDF zur Verfügung stellen.

1.4 Aufbau der Studienarbeit

Zu Beginn werden in Kapitel 2, ab Seite 6, die häufigsten Probleme beim Modellieren in RDF anhand eines Anwendungsfalls mit konkreten Beispielen demonstriert. Alle Probleme lassen sich schon beim Modellieren von sehr simplen Sachverhalten mit ein bis zwei zu modellierenden Aussagen erkennen.

Die Ursachen dieser Schwierigkeiten werden in Kapitel 3 analysiert, ab Seite 12. Diese Ursachen liegen sowohl in den Restriktionen, welche von der abstrakten Syntax von RDF an den Modelliervorgang gestellt werden, als auch in der formalen Semantik von RDF begründet.

In Kapitel 4 wird SiRDF ab Seite 20 vorgestellt. Das Datenmodell von SiRDF beruht auf nur drei Entitäten: Dem Graph, den Aussagen welche im Graph repräsentiert werden, und einem einzigen Knotentyp, aus dem alle Aussagen gebildet werden. Darauf aufbauend wird die abstrakte Syntax zur Konstruktion eines SiRDF Graphen spezifiziert, sowie die formale Semantik, welche es erlaubt die Information in einem SiRDF Graph bezüglich des Modells einer möglichen Welt zu Interpretieren.

Danach wird in Kapitel 5 ab Seite 29 untersucht, welche formalen Voraussetzungen erfüllt werden müssen, damit ein SiRDF Modell und ein RDF Modell die gleiche mögliche Welt beschreiben, und damit Information in SiRDF und RDF Graphen auf die gleiche Weise bezüglich dieser Welt interpretiert werden kann. Aufbauend auf dieser formalen Grundlage wird ein Algorithmus zur semantisch äquivalenten Transformation zwischen RDF und SiRDF Modellen gegeben. Diese Transformation kann in beide Richtungen ausgeführt werden.

Die Implementierung des *Application Program Interface (API)* zu SiRDF wird in Kapitel 6 ab Seite 44 beschrieben. Die API erlaubt das Manipulieren der Entitäten im SiRDF Modell und das Transformieren zwischen SiRDF und RDF Graphen.

In Kapitel 7 wird ab Seite 48 der Aufwand der Modellierung in SiRDF und RDF verglichen. Mehrere einfache Anwendungsfälle werden geschildert, und es wird das mentale Modell des Anwenders vor dem Modellieren beschrieben. Dann wird der jeweilige Sachverhalt in RDF und SiRDF modelliert und die resultierenden Graphen werden verglichen.

Abschließend wird in Kapitel 8 ab Seite 55 eine Zusammenfassung der Studienarbeit gegeben.

2 Motivation

In diesem Kapitel werden die Probleme mit RDF, welche die Entwicklung von SiRDF motivieren, demonstriert. Die theoretischen und formalen Hintergründe dieser Probleme werden im Kapitel 3 ab Seite 12 ausführlich behandelt.

Die aktuelle RDF Spezifikation [MM04] hat einen mehr als zweijährigen Designprozess hinter sich. Dabei wurden viele Entscheidungen getroffen, welche direkte oder indirekte Auswirkungen auf alle Aspekte von RDF haben. Die wichtigsten Schwierigkeiten beim Transfer eines mentalen Modells in das konzeptuelle Modell von RDF werden nun an einem Anwendungsfall demonstriert.

2.1 Anwendungsfall: Annotation in einer digitalen Bibliothek

Digitale Bibliotheken zeichnen sich nicht nur durch die Digitalisierung ihrer Inhalte aus, sondern auch durch das Bereitstellen von Metadaten über die gesammelten Medien, um das Suchen und Finden der gewünschten Informationen in der Bibliothek zu ermöglichen. Ein standardisiertes RDF Vokabular um bibliothekarische Metadaten zu beschreiben sind die *Metadata Terms* [Dub05] der *Dublin Core Metadata Initiative*. Wir betrachten nun den sehr minimalistischen Anwendungsfall, eines Dokuments „Dokument1“, über welches die Aussage gemacht werden soll, das es den Autor „Klaus“ hat. Später werden wir weitere Aussagen über „Dokument1“ und „Klaus“ machen.

2.1.1 Festhalten des Autors eines Dokuments

In RDF drücken wir die Tatsache, das „Dokument1“ von „Klaus“ verfasst wurde, durch folgende Aussage aus:

```
ex:Dokument1 dc:creator "Klaus"
```

Graphisch lässt sich eine Aussage durch zwei kreisförmige Knoten darstellen, welche durch eine gerichtete Kante miteinander verbunden sind, wie in Abbildung 2.1 zu sehen ist. Der erste Knoten repräsentiert das Subjekt, der zweite Knoten das Objekt, die gerichtete Kante repräsentiert das Prädikat und zeigt vom Subjekt-Knoten auf den Objekt-Knoten.

Weil wir eine Aussage über „Dokument1“ machen, ist „Dokument1“ das Subjekt der Aussage. Wir geben zu „Dokument1“ eine komplette Adresse an, unter

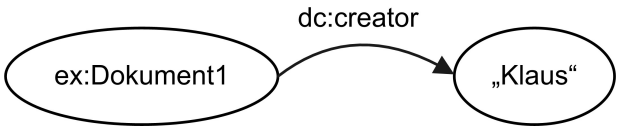

Abbildung 2.1: Festhalten des Autors eines Dokuments

der das Dokument im *World Wide Web (WWW)* zu finden ist. Diese Adresse ist innerhalb des WWW eindeutig und kann deswegen mit dem bezeichneten Dokument „gleichgesetzt“ werden. Mehr zu diesem kontroversen Thema in 3, Seite 12, und in [URI01].
Diese Art von Adresse wird *Uniform Resource Identifier (URI)* genannt. Die URI von „Dokument1“ besteht aus zwei Teilen, einem Präfix, welches bei URIs als *Namespace* bezeichnet wird, und der eigentlichen Adresse. Das hier verwendete Präfix `ex` steht hier für die Zeichenkette `http://ontoware.org/`, und zusammen mit dem Namen des Dokuments ergibt sich die vollständige Form der URI: `http://ontoware.org/Dokument1`

Das Verb „hat-Autor“ ist das Prädikat der Aussage, doch statt „hat-Autor“ nehmen wir das Verb `dc:creator`, welches von der *Dublin Core Metadata Initiative* wie folgt definiert wurde: „Die Entität, welche hauptsächlich für das Erstellen des Inhalts der Ressource verantwortlich ist.“ `dc` ist das Präfix aller Verben der Dublin Core Metadata Initiative, es steht für `http://purl.org/dc/elements/1.1/`.

Wir machen eine Aussage über „Klaus“, also ist er das Objekt der Aussage. RDF definiert die Reihenfolge *Subjekt*, *Prädikat*, *Objekt* für alle Aussagen. Eine solche Aussage bezeichnet man als *RDF Tripel*.

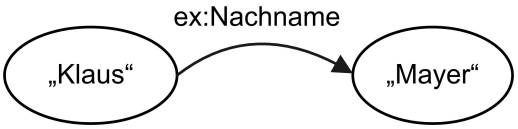

Abbildung 2.2: Autor als Literal modelliert: Resultierender Graph ist in RDF ungültig

Weitere Aussagen über den Autor Wir wollen nun weitere Aussagen über den Autor machen, um zusätzliche Informationen wie seinen Nachnamen und seine EMail-Adresse festzuhalten. Wir haben gesehen, dass Aussagen in RDF als Subjekt-Prädikat-Objekt-Tripel gemacht werden. Es würde sich also anbieten den Nachnamen von „Klaus“ wie folgt anzugeben: (siehe Abbildung 2.2)

```
"Klaus" ex:Nachname "Mayer"
```

Diese Aussage ist in RDF ungültig. „Klaus“ ist im Gegensatz zum „Dokument1“ nicht mit einer URI in der Aussage referenziert, sondern nur durch durch eine Zeichenkette, ein *RDF Literal*. *RDF Literale* haben keine eindeutige Adresse. Deswegen kann ein *Literal* nur die Rolle eines Objektes in einem *RDF Tripel* einnehmen. Hinzu kommen in RDF Vorschriften darüber, welche Art von *RDF Entität* welche Rolle innerhalb eines Tripels einnehmen darf. *Literale* dürfen nicht das Subjekt eines Tripels sein, was ein weiterer Grund ist, warum diese Aussage in RDF ungültig ist.

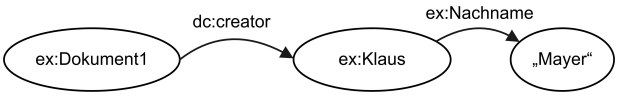

Abbildung 2.3: Autor als URI mit globaler Identität modelliert

2.1.2 Der Autor als eindeutige Entität

Um Aussagen über den Autor machen zu können, müssen wir ihm eine Adresse geben. Die erste Möglichkeit ist es, „Klaus“ mit einer URI zu assoziieren, etwa: `http://ontoware.org/Klaus`, abgekürzt als `ex:Klaus`. Dann würden wir folgende Aussagen in RDF machen: (siehe Abbildung 2.3)

```
ex:Dokument1    dc:creator      ex:Klaus
ex:Klaus        ex:Nachname     "Mayer"
```

Da wir nun nicht mehr über irgendeinen Klaus sprechen sondern über einen bestimmten „Klaus“, den wir mit der URI `ex:Klaus` identifiziert haben, können wir eine Aussage über diesen bestimmten „Klaus“ machen. Die *Dublin Core Metadata Terms* [Dub05] schreiben nicht vor, wie „Klaus“ modelliert werden muss.

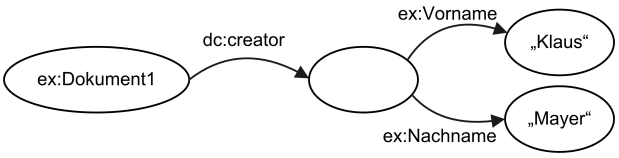

Abbildung 2.4: Autor als *Blank Node* modelliert

Der Autor als Blank Node In RDF gibt es jedoch nicht nur *Literale* ohne eigene Adresse, und *URI Referenzen* mit einer global gültigen Adresse, sondern auch noch einen konzeptuell komplexeren Mittelweg. Es ist möglich einer *RDF Entität* eine Adresse zu geben, welche nur in dem *RDF Graph*, in welchem diese

Entität definiert wurde, gültig und eindeutig ist. Dieser Typ von RDF Entität wird *Blank Node* genannt. Wenn wir den Autor von „Dokument1“ mit einem *Blank Node* identifizieren, und dem Autor dann einen Vornamen und Nachnamen geben, ergeben sich folgende Aussagen: (siehe Abbildung 2.4)

```
ex:Dokument1    dc:creator      _:1
_:1             ex:Vorname      "Klaus"
_:1             ex:Nachname     "Mayer"
```

Anders als bei *URI Referenzen* ist es bei *Blank Nodes* nicht notwendig ihre Adresse anzugeben. Die Adresse eines *Blank Nodes* wird mit einem eindeutigen Identifikator gleichgesetzt, welcher automatisch für jeden *Blank Node* erzeugt wird. Wenn die *RDF Tripel* aufgeschrieben werden, wird dieser automatisch erzeugte Identifikator meistens einfach als Nummer notiert, welche mit der Adresse des *Blank Nodes* gleichgesetzt werden kann. Die Einschränkung der Gültigkeit der Adresse eines *Blank Nodes* auf den Graph in dem er definiert wird, und die automatische Erzeugung seiner Adresse sind die wichtigsten Eigenschaften die einen *Blank Node* von einer URI Referenz unterscheiden. Beim Design von RDF wurden jedoch Ausnahmefälle in denen eine begrenzt gültige Adresse Vorteile bringt, für so wichtig eingestuft, das das abstrakte Konzept der *Blank Nodes* zur Modellierung von Information in RDF zur Verfügung gestellt wird. Näheres dazu in Kapitel 3, ab Seite 12.

2.2 Verschiedene Arten von Literalen

Ein *RDF Literal* kann nicht nur eine Zeichenkette sein, zusätzlich kann ein *Sprachbezeichner* oder ein *Datentyp* für ein *Literal* angegeben werden. Das ermöglicht es im Anwendungsfall den Nachnamen nicht nur auf Deutsch sondern auch in der japanischen Aussprache zu speichern. Für Werte, zu denen eine Einheit gespeichert werden muss, kann ein Datentyp angegeben werden. Dies kann zum Beispiel eine Währung (Yen, Euro) oder eine physikalische Masseinheit (Celsius, Fahrenheit) sein.

```
ex:Klaus        ex:Nachname     "Mayer"^^DE
ex:Klaus        ex:Nachname     "Ma-I-Ya"^^JP
ex:Klaus        ex:Gehalt       "2542"^^Euro
```

Literale mit und ohne *Sprachbezeichner* (engl.: *language tag*)werden als *einfache Literale* bezeichnet (engl.: *plain literals*). *Literale* mit *Datentyp* (engl.: *data type*)werden als *typisierte Literale* bezeichnet (engl.: *typed literals*). *Literale* mit *Sprachbezeichner* und *Datentyp* sind in RDF nicht zugelassen, obwohl ein solcher Anwendungsfall denkbar wäre. Ein *Literal* darf nur einen *Sprachbezeichner* oder

einen *Datentyp* haben, aber es ist nicht möglich Aussagen über den *Datentyp* oder den *Sprachbezeichner* zu machen, da beide nur als Erweiterungen eines *Literals* ohne eigene Adresse entworfen wurden. So kann keine Änderungsliste für den *Sprachbezeichner* des *Literals* gespeichert werden. Damit ist der Graph in Abbildung 2.5 nicht in RDF erlaubt. Näheres zu Literalen in in Kapitel 3, ab Seite 12.

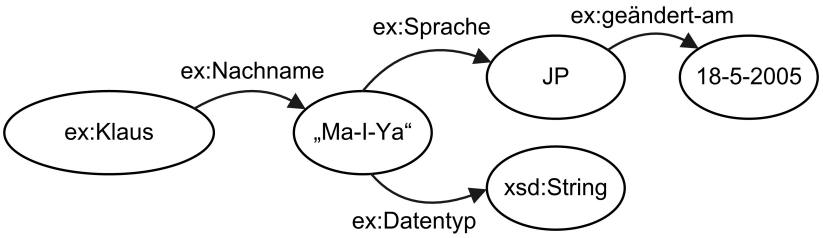

Abbildung 2.5: Aussagen über den Sprachbezeichner sind in RDF ungültig

Weil *Sprachbezeichner* und *Datentypen* keine Adresse haben, ist es unter anderem auch sehr schwer alle Informationen in einer bestimmten Sprache aus einer Wissenssammlung zu filtern, etwa alle Bücher in einer digitalen Bibliothek mit einem französischen Autor zu finden. Zu weiteren Problemen mit *Sprachbezeichnern* in RDF siehe [CP05].

2.3 Reification

Das Konzept mit dem vielleicht höchsten Grad an Komplexität in RDF ist eine Form der indirekten Rede, welche als *Reification* bezeichnet wird. Wenn Klaus nicht zur Arbeit erscheint, und sein Arbeitskollege Peter gefragt wird, was mit Klaus passiert ist, könnten wir von Peter die Antwort erhalten, das Klaus krank sei. In der deutschen Sprache bekommt die Aussage „Klaus ist krank“ durch den Konjunktiv „sei“ die Konotation der Ungewissheit. Im Gegensatz dazu macht der Satz „Peter sagt, das Klaus definitiv Krank ist“ eine Aussage über den Wahrheitswert der Tatsache, das Klaus krank ist.

Um in einer Aussage eine andere Aussage referenzieren zu können ist es notwendig das die erste Aussage, die zweite Aussage adressieren kann. In RDF wird aber die Adresse einer Aussage nicht mit der Aussage, die adressiert wird, gleich gesetzt. Die formalen Aspekte der *Reification* werden in Kapitel 3 behandelt, ab Seite 12.

2.4 Identifizierte Probleme

Wie in den vorangegangenen Abschnitten gezeigt wurde, gibt es mehrere konzeptionelle Schwächen von RDF, welche beim Transfer eines mentalen Modells in das Modell von RDF deutlich werden:

- *Literale* lassen sich nicht adressieren.
- *Datentypen* und *Sprachbezeichner* sind in RDF Bestandteile eines *Literals* und keine eigenen Entitäten. Deswegen lassen sie sich auch nicht adressieren.
- *Blank Nodes* sind bei der Interoperabilität zwischen Graphen problematisch, da sie sich nicht global adressieren lassen.
- Die Semantik der *Reification* in RDF erlaubt keine Adressierung von Aussagen.
- Es existieren Einschränkungen darüber, welche Art von Entität welche Rolle in einem RDF Tripel einnehmen darf

Im nächsten Kapitel werden die theoretischen Grundlagen für die Komplexität beim Modellieren in RDF erörtert, und es wird gezeigt, das alle Schwierigkeiten inhärent in den formalen Grundlagen begründet liegen.

3 Grundlagen

Um die identifizierten Schwierigkeiten beim Transfer eines mentalen Modells in das konzeptionelle Modell von RDF, analysieren zu können, muss man sie auf das Datenmodell, die abstrakte Syntax und die formale Semantik von RDF zurückführen. Ein Verständnis der Grundlagen von RDF ist auch notwendig, um diese Probleme beim Entwurf eines vereinfachten neuen Modells zu vermeiden. Wir geben deswegen nun die wichtigsten Definitionen und Formalismen zu RDF an. Für weitere Details wird auf die jeweiligen Teile der RDF Spezifikation verwiesen. Danach werden die im vorherigen Kapitel identifizierten Probleme mit den formalen Grundlagen von RDF in Zusammenhang gebracht.

3.1 Formale Grundlagen von RDF

3.1.1 Datenmodell

Das Datenmodell von RDF [KC04, Kapitel 3] definiert, wie Information im RDF Modell gespeichert werden. Eine Informationssammlung wird als RDF **Graph** bezeichnet.

Ein Graph ist eine Menge von **Aussagen**, in der englischen Literatur als **Statements** oder **Tripel** bezeichnet. Eine Aussage ist die kleinste atomare Einheit von Information im RDF Modell. Das bedeutet, dass semantische Schlussfolgerungen auf der Ebene von Aussagen und Mengen von Aussagen gezogen werden.

Eine Aussage wird im Graph durch ein **Tripel** repräsentiert. Das erste Element des Tripels hat die Rolle des **Subjekts** der Aussage, das zweite Element hat die Rolle des **Prädikats**, das dritte Element hat die Rolle des **Objekts**.

3.1.2 Abstrakte Syntax

Für jede Rolle innerhalb eines Tripels gibt es Restriktionen bezüglich der Art von RDF Entität, welche ein Rolle in einem Tripel übernehmen kann, siehe [KC04, Kapitel 6]. In RDF gibt es drei Typen von Entitäten: *URI Referenzen*, *Literale* und *Blank Nodes*.

URI Referenzen sind *Unicode* [Con03] Zeichenketten, in denen keine Kontrollzeichen vorkommen. Sie ergeben eine gültige URI Zeichenkette, welche eine absolute URI, gefolgt von einer optionalen Fragment-Kennzeichnung, darstellt,

wenn die URI gemäß [BLFM98, Abschnitt 2.1], interpretiert wird. Zwei *URI Referenzen* sind genau dann gleich, wenn ihre Unicode Zeichenketten gleich sind.

An dieser Stelle ist es wichtig darauf hinzuweisen, dass je nach Anwendungsfall URIs nicht nur als Adressen, sondern als Namen von Ressource verwendet werden. [URI01] beschäftigt sich ausführlich mit dieser Problematik. Um bei einer konkreten vorliegenden URI zwischen dem Fall einer Namensgebung und dem Fall einer Adressierung unterscheiden zu können ist weiteres Wissen nötig. Es gibt kein inhärentes Merkmal einer URI, welches diese Unterscheidung ermöglicht.

Literale eines RDF Graphen bestehen aus einer oder zwei Komponenten. Alle Literale haben eine *lexikalische Form*, welche aus einer Unicode Zeichenkette besteht. *Einfache Literale* (engl.: *plain literals*) haben zusätzlich zu ihrer lexikalischen Form einen optionalen *Sprachbeichner* (engl.: *language tag*), welcher auch als Unicode Zeichenkette dargestellt wird (siehe [Alv01]). *Typisierte Literale* (engl.: *typed literals*) haben eine lexikalische Form und eine *Datentyp* URI Referenz. Literale sind genau dann gleich, wenn ihre lexikalischen Formen zeichenweise gleich sind, und wenn Datentypen, falls bei beiden vorhanden, zeichenweise gleich sind, sowie Sprachbezeichner, falls bei beiden vorhanden und bis auf Groß- und Kleinschreibung zeichenweise gleich.

Blank Nodes werden in der RDF Spezifikation nur sehr knapp definiert [KC04, Abschnitt 6.6]. Die Menge der *Blank Nodes* und die Mengen der Literale und der *URI Referenzen* sind jeweils paarweise disjunkt voneinander. Zwei *Blank Nodes* können auf Gleichheit getestet werden. Die Erstellung eines Kennzeichners für jeden *Blank Nodes* ist einer Implementierung überlassen, sie muss aber innerhalb eines Graphen eindeutig sein. Deswegen sind *Blank Nodes* RDF Entitäten mit einer auf einen RDF Graph beschränkten Adressierbarkeit. Im Gegensatz dazu, haben *URI Referenzen* eine global gültige Adresse und *Literale* keine Adresse.

Für diese drei RDF Entitäten gelten Beschränkungen bezüglich der Rolle, welche die Entitäten in einem *Tripel* einnehmen dürfen: eine *URI Referenz* darf überall in einem Tripel vorkommen, ein *Blank Node* darf nicht als Prädikat vorkommen und ein *Literal* darf nur als Objekt vorkommen. Siehe dazu [KC04, Abschnitt 6.1] und die Tabelle in Abbildung 3.1.

	URI Referenz	BlankNode	Literal
Subjekt	X	X	
Prädikat	X		
Objekt	X	X	X

Abbildung 3.1: Restriktionen für die Rollen in einem *RDF Tripel*

3.1.3 Die formale Semantik von RDF

Aufbauend auf dem Datenmodell und der abstrakten Syntax ist die formale Semantik von RDF definiert. Durch sie wird eine Interpretation auf dem RDF Modell spezifiziert, welche es erlaubt eine mögliche Welt mit einer ausreichenden Menge von Informationen zu beschreiben um den Wahrheitswert für einen RDF Graph in dieser möglichen Welt zu bestimmen. Die Semantik wird in mehreren aufeinander aufbauenden Schritten definiert.

Die Interpretation eines Graphen ohne Blank Nodes Der erste Schritt ist die Definition einer Interpretation für einen **Ground Graph**, ein RDF Graph in dem keine *Blank Nodes* vorkommen.

Definition 3.1: Interpretation eines RDF Graphen Eine Interpretation I auf einem Vokabular V, ist nach [Hay04, Abschnitt 1.3] definiert durch:

- Das **Unversium IR** von I, ist die nichtleere Menge aller Ressourcen von I. Sie wird auch als **Domäne** von I bezeichnet.
- Die Menge **IP** der **Properties** von I. Die Prädikate eines Tripels werden als Property bezeichnet.
- Die Abbildung $\mathbf{IExt}() : IP \longmapsto < IR \times IR >$, welches die Menge der Paare (x, y) mit x und y in IR ist.
- Die Abbildung $\mathbf{IS}()$ von der Menge der *URI Referenzen* in $V \longmapsto IR \cup IP$
- Die Abbildung $\mathbf{IL}()$ von der Menge der *typisierten Literale* in $V \longmapsto IR$
- Die Menge **LV** ist die Untermenge der *einfachen Literale* aus IR

Die Denotation eines Graphen ohne Blank Nodes Der Wahrheitswert eines RDF Graphen bezüglich einer Interpretation wird rekursiv über die Denotation jedes Bestandteils des Graphen definiert. RDF kennt zwei Arten von Denotation:

- Namen haben die Denotation eines „Dinges“ im Universum I
- Tripel haben die Denotation eines Wahrheitswertes

Die folgende Definition nach [Hay04, Abschnitt 1.4] erweitert die Denotation einer Menge von Tripeln zur Denotation eines ganzen Graphen. Die Bestimmung des Wahrheitswertes wird rekursiv über die syntaktischen Bestandteile eines Graphen bestimmt.

Definition 3.2: Denotation eines Graphen ohne Blank Nodes

- Sei E ein *einfaches Literal* "aaa" aus V, dann ist $I(E) = aaa$
- Sei E ein *Literal mit Sprachbezeichner* "aaa"@ttt aus V, dann ist $I(E) =< aaa, ttt >$
- Sei E ein *typisiertes Literal* aus V, dann ist $I(E) = IL(E)$
- Sei E eine *URI Referenz* aus V, dann ist $I(E) = IS(E)$
- Sei E ein Tripel $< s, p, o >$ ohne *Blank Nodes*, dann ist

  $$I(\text{s p o}) = true \iff (s, p, o \in V) \wedge (I(p) \in IP) \wedge (< I(s), I(o) > \in IExt(I(p)))$$

  sonst ist $I(\text{s p o}) = false$
- Sei E ein *Ground Graph*, dann ist $I(E) = false \iff I(T) = false$ für ein Triple $T \in E$, sonst ist $I(E) = true$

Die Interpretation eines Graphen mit Blank Nodes Die rekursive Definition des Wahrheitswerts eines Graphen lässt sich nach nach [Hay04, Abschnitt 1.5] um *Blank Nodes* erweitern.

Definition 3.3: Denotation eines Graphen mit *Blank Nodes* Sei I eine Interpretation und sei A eine Abbildung von der Menge der *Blank Nodes* in das Universum IR von I. Sei $blank(E)$ die Menge der *Blank Nodes* in E. Dann definiere $I + A()$ als erweiterte Interpretation mit:

- Sei E ein *Blank Node*, und sei $A(E)$ definiert, dann ist $[I + A](E) = A(E)$
- Sei E kein *Blank Node*, dann ist $[I + A](E) = I(E)$
- Sei E ein RDF Graph, dann ist $I(E) = true$ wenn $[I + A](E) = true$ für eine Abbildung $A() : blank(E) \longmapsto IR$, sonst ist $I(E) = false$

3.1.4 Semantik der Reification

Reification beschreibt wie schon in Abschnitt 2.3, Seite 10, ausgeführt, eine Form der indirekten Rede. Die RDF Spezifikation gibt das Vokabular vor, also die normierten Prädikate, um eine *Reifizierung* auszudrücken. Gegeben sei folgendes Tripel:

```
ex:a    ex:b    ex:c
```

Dann ist eine *Reifizierung* des Tripels eine Menge mehrerer Tripel der folgenden Form:

```
_:xx    rdf:type        rdf:Statement
_:xx    rdf:subject     ex:a
_:xx    rdf:predicate   ex:b
_:xx    rdf:object      ex:c
```

`_:xx` bezeichnet man als das *reifizierte Tripel*. In diesem Fall ist dies ein *Blank Node*, es kann auch durch eine URI dargestellt werden kann. Das *reifizierte Tripel* kann Subjekt oder Objekt einer anderen Aussage sein.

Es bietet sich intuitiv an das *reifizierte Tripel* als Adresse des echten Tripels zu verwenden. `_:xx` wäre also die Adresse des Tripels `ex:a ex:b ex:c`. Doch in [Hay04, Abschnitt 3.3.1] wird explizit keine semantische Verbindung zwischen dem *reifizierten Tripel* und dem „echten Tripel“ gleichen Inhalts definiert, sondern nur eine nicht vorgeschriebene Möglichkeit zur Erweiterung der Semantik erörtert.

Bildlich gesprochen lässt sich das mit dem Adressieren eines Briefs für den Postversand vergleichen. Die Adresse auf dem Briefumschlag impliziert, das die Person, an die der Brief adressiert ist, auch in dem Haus wohnt. In manchen Fällen, etwa bei Wohngemeinschaften, kann es passieren das ein Brief zugestellt wird, obwohl die Person nicht mehr in dem Haus wohnt, weil der Briefträger nicht überprüfen kann, ob die Person innerhalb des Haushaltes an den der Brief adressiert ist, noch in dem Haushalt wohnt. Die Adresse auf dem Briefumschlag impliziert eine starke Bindung zwischen dem Haushalt des Empfängers und dem Empfänger selbst, diese Verbindung ist aber nicht immer gegeben.

In ähnlicher Weise gibt es in RDF keine semantisch spezifizierte Bindung zwischen der Adresse einer Aussage und der Aussage selbst. Das führt dazu, das es nicht möglich ist eine Aussage von einer anderen Aussage aus zu adressieren. Da das *reifizierte Tripel* nicht mit dem zu *adressierenden Tripel* assoziert wird, kann es formal nicht als Adresse verwendet werden. Weitere Informationen zu Problemen im Umgang mit Blank Nodes zu finden in [DFZD, Kapitel 2.2], [YK02] und [Pat].

3.2 Beispiel einer RDF Interpretation

Zum besseren Verständnis geben wir nun ein Beispiel für eine RDF Interpretation (siehe Abbildung 3.2):

Es sei das Vokabular V gegeben, mit
V := `{ ex:a, ex:b, ex:c, "whatever", "whatever"^^ex:b }`. Dann ist I wie folgt definiert:

- $IR := LV \cup 1, 2$
- $LV :=$ {”whatever”}
- $IP := \{1\}$

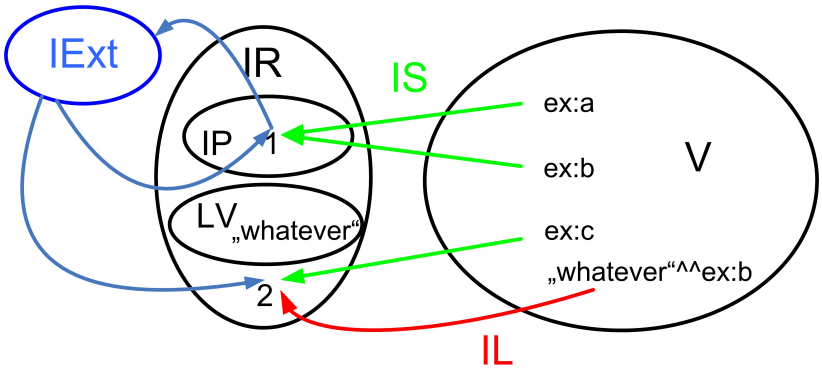

Abbildung 3.2: Illustration der gegebenen RDF Interpretation

- $IExt := (1 \Longrightarrow \{< 1, 2 >, < 2, 1 >\})$
- $IS := ex : a \Longrightarrow 1, ex : b \Longrightarrow 1, ex : c \Longrightarrow 2$
- $IL :=$ `"whatever"^^ex:b` $\Longrightarrow 2$

Erläuterungen V ist das Vokabular auf dem die Interpretation definiert ist. In V sind *URI Referenzen*, *Literale* und *Blank Nodes* enthalten. Die Interpretation $I()$ ist für Elemente aus V als Abbildung vom Vokabular V in die Menge der *Properties* IP und/oder die Menge der *Ressourcen* IR definiert. In IR sind die „Dinge“ auf denen die Interpretation definiert ist, enthalten, in IP sind die binären Operatoren auf diesen Dingen enthalten, aber noch nicht die binären Relationen selber. Diese sind über die Relation $IExt()$ von der Menge der *Properties* IP in das Kreuzprodukt von der Menge der *Ressourcen* IR definiert.

Es ist über $IExt()$ möglich zu überprüfen ob eine *Property* eine binäre Relation zwischen zwei *Ressourcen* definiert hat. Deswegen kann jedem Tripel aus dem RDF Graph ein Wahrheitswert zugeordnet werden, in dem überprüft wird, ob nach dem Interpretieren des Subjekts und des Objekts, die ihnen zugeordneten „Dinge“ in der binären Relation vorkommen, welche über das Prädikat des Tripels definiert ist.

Denotation eines Graphen Die Denotation der folgenden Tripel ist im gegebenen Beispiel jeweils einzeln interpretiert wahr:

```
ex:a    ex:b    ex:c
ex:c    ex:b    ex:a
ex:a    ex:b    "whatever"^^ex:b
```

Ein RDF Graph, welcher nur aus diesen drei Tripeln besteht, ist folglich auch wahr unter der Interpretation I.

3.3 Die Ursachen der konzeptionellen Schwächen von RDF

Einschränkungen durch die abstrakte Syntax In RDF sind drei Typen von Entität definiert: *URI Referenzen*, *Blank Nodes* und *Literale*. Die abstrakte Syntax legt fest, welche Art von Entität welche Rolle in einem Tripel und damit in einer RDF Aussage annehmen darf. Diese Einschränkungen lassen sich aber nicht nur auf das Datenmodell zurückführen. Um das Problem der Adressierbarkeit von Literalen zu lösen, muss das RDF Datenmodell und die abstrakte Syntax geändert werden.

Adressierbarkeit von Literalen Eine Stufe höher, auf der Ebene der formalen Semantik, findet sich der zweite Grund für das Problem der Nicht-Adressierbarkeit von *Literalen*. Die Interpretation $I()$ bildet das Vokabular V auf das Universum von I ab, das ist die Menge der *Ressourcen* IR, und auf die Menge der *Properties* IP. *Einfache Literale* werden von $I()$ gar nicht auf IR abgebildet, sondern sie sind als Untermenge von IR definiert. *Typisierte Literale* dagegen, werden von der Abbildung $IL()$ von V nach IR abgebildet. *Einfache Literale* und *typisierte Literale* werden sehr unterschiedlich behandelt. Um *Literale* zu adressieren, wäre ein denkbarer Lösungsansatz, alle Formen von *Literalen* gleich zu behandeln und ihnen eine identifizierende Kennzeichnung zu geben, durch welche sie auch eindeutig adressierbar sind.

Datentypen und Sprachbezeichner *Datentypen* und *Sprachbezeichner* sind in RDF nur als Erweiterungen eines *Literals* modelliert, und nicht als eifenständige Entitäten. Sie sind im Rahmen der abstrakten Syntax nur als Bestandteile von *einfachen* und *typisierten Literalen* definiert. Eine Modellierung, in der *Datentypen* und *Sprachbezeichner* entkoppelt von den *Literalen*, zu denen sie gehören betrachtet werden, und in dem sie adressierbar sind, würde diese Hürde entfernen.

Blank Nodes Wie wir gesehen haben, sind *Blank Nodes* erst als Erweiterung der formalen Semantik von RDF eingeführt worden. Die abstrakte Syntax definiert sie als disjunkt jeweils zu den Mengen der *URI Referenzen* und der *Literale*. Zusätzlich ist es möglich zwischen *Blank Nodes* zu Unterscheiden. Semantisch gesehen ist es möglich eine Abbildung von den *Blank Nodes* in das Universum IR zu definieren, so das auch Tripel mit *Blank Nodes* auf ihren Wahrheitswert untersucht werden können. Durch Skolemisierung ist es jedoch möglich jeden *Blank Node* umzubennen, siehe dazu [Hay04, Appendix A, Skolemization Lemma]. Das bedeutet das die Kennzeichnung des *Blank Nodes* im Graph durch eine andere Kennzeichnung ausgetauscht wird, wobei die Identität des *Blank Nodes* nicht verloren geht, da die neue Kennzeichnung den gleichen Gültigkeitsbereich hat. Dabei

muss darauf geachtet, das es zu keinen globalen Namenskollisionen kommt. Beim Prozess der Skolemisierung kann als neue Kennzeichnung nicht nur ein anderer *Blank Node* Identifikator gewählt werden, sondern alternativ auch eine Referenz auf eine eigens dafür erzeugte URI.

Semantik der Reification Die RDF Spezifikation schreibt keine Möglichkeit zur semantisch eindeutig interpretierbaren Adressierung eines Tripel vor, obwohl Adressierung ein sehr häufiger Anwendungsfall sein kann. Indirekte Rede ist im Gegensatz dazu eher auf höheren konzeptuellen Ebenen erforderlich.

3.4 Komplexität der Programmierschnittstelle

Die Schwierigkeiten beim Transfer eines mentalen Modells in das Modell von RDF werden durch ein *Application Programming Interface (API)*, welches den vollen Funktionsumfang von RDF implementiert, an den Anwender weitergegeben.

Das Beispiel in Abbildung 2.4, Seite 8, modelliert den Autor „Klaus“ als *Blank Node*. Mit der stabilen RDF API Jena [McB01] wird das RDF Modell mit folgendem Programm-Code implementiert:

```
Model model = ModelFactory.createDefaultModel();
Resource klausMayer
  = model.createResource("ex:Dokument1")
         .addProperty("dc:creator",
                      model.createResource()
                           .addProperty("ex:Vorname", "Klaus")
                           .addProperty("ex:Nachname", "Mayer"));
```

Zuerst wird ein leeres Modell erzeugt, diesem wird „Dokument1“ als *Ressource* hinzugefügt. „Dokument1“ bekommt dann die *Property* `dc:creator` bezogen auf einen *Blank Node*. Dieser *Blank Node* bekommt wiederum die zwei eigenen *Properties* `test:Vorname` und `test:Nachname` mit den jeweiligen *Literalen*. Zunächst fällt auf, das die Unterscheidung zwischen *Property* und *Ressource* sich auch in der Namensgebung der `add` Funktionen niederschlägt. Dort vermischen sich dann aber auch wieder sehr schnell Konzepte aus dem RDF Modell, da bei der `addProperty()` Funktion auch noch das Objekt zu der Property angegeben werden muss. Generell fällt auf, das der Gesamtaufbau der Funktionen nicht auf das inhärente Datenmodell von RDF schließen lässt. Wo ein Statement aufhört und ein anderes anfängt lässt sich ohne weitere Kenntnisse nicht aus der Syntax des Programmcodes erschließen.

4 Simple RDF (SiRDF)

4.1 Die Idee hinter SiRDF

Wie in Kapitel 2, ab Seite6, gezeigt wurde, gibt es einige Schwierigkeiten beim Transfer eines mentalen Modells in das konzeptuelle Modell von RDF. Diese Probleme beim Modellieren eines Sachverhaltes sind inhärent in der abstrakten Syntax und der formalen Semantik von RDF begründet, wie in Kapitel 3, ab Seite 12, gezeigt wurde. Abhängig von den Anforderungen an die kognitiven Leistungen, welche an den Prozess der Modellierung gestellt werden, kann zwischen zwei Ebenen der Modellierung unterschieden werden, siehe Abbildung 4.1. Auf der kognitiv höheren Ebene findet das Identifizieren von Ideen, Konzepten, Sachverhalten und Informationen statt, sowie das Erstellen eines internen mentalen Modells in dem diese einzelnen Informationen zu einem zusammenhängenden System verknüpft werden. Die Komplexität der Modellierung auf dieser Ebene ist in den zu modellierenden Sachverhalten begründet. Siehe dazu [Sch02].

Auf der kognitiv tieferen Ebene findet das Modellieren des internen mentalen Modells als Instanz des Zielmodells statt. Bisher wurde das Zielmodell in RDF modelliert, um es im Rahmen des Semantic Web weiterverwenden zu können. Dabei wird die Information der höheren Modellierungsebene in das Zielmodell übertragen. Die Komplexität dieses Schritts kann isoliert betrachtet werden, und auf die Komplexität des Zielsystem zurückgeführt werden.

Um den kognitiven Prozess des Modellierens so einfach wie möglich zu Ge-

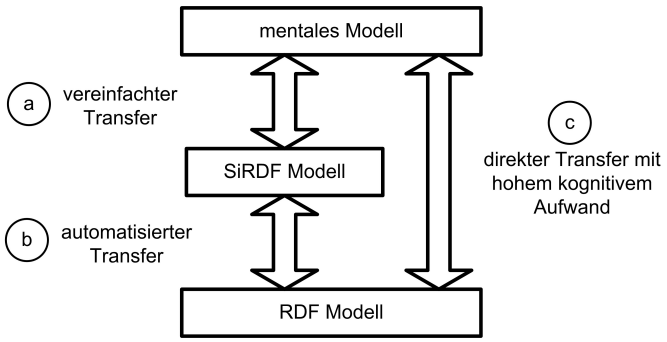

Abbildung 4.1: Transferprozess vom mentalen Modell des Anwenders über SiRDF als Vermittlungsmodell zum konzeptuellen Modell von RDF

stalten, ist es notwendig, einen Rahmen für ein neues konzeptuelles Modell zu schaffen. Dieser Rahmen kann von Grund auf neu gestaltet werden, oder er kann evolutionär auf der schon bestehenden Basis von RDF entwickelt werden. Für diese Arbeit wurde der evolutionäre Ansatz gewählt.

Die identifizierten Schwächen beim Modellieren mit RDF *(Pfeil c)* wurden beim Design des neuen Modells von vorne herein berücksichtigt. Dieses neue Modell wird Simple RDF (SiRDF) genannt, da es vor allem durch Vereinfachungen an der bestehenden abstrakten Syntax und formalen Semantik von RDF entstanden ist.

SiRDF hat bezüglich des Gesamtprozesses der Modellierung eine Vermittlungsfunktion zwischen dem mentalen Modell des Anwenders und dem konzeptuellen Modell von RDF. Der Transfer des mentalen Modells in das Zielsystem SiRDF *(Pfeil a)* ist für den Anwender mit einer geringeren kognitiven Komplexität zu bewältigen, da das Datenmodell und die formalen Grundlagen von SiRDF das Verständnis von weniger Vorschriften und Konzepten erfordert.

Das Ziel des Anwenders, den zu modellierenden Sachverhalt extern festzuhalten ist damit erfüllt. Um nun die Interoperabilität der festgehaltenen Information im Rahmen des Semantic Web zu gewährleisten, muss das SiRDF Modell ein weiteres Mal transferiert werden *(Pfeil b)*. Erst wenn die Information durch RDF modelliert wurde, kann sie für das Semantic Web nutzbar gemacht werden. Dieser zweite Transferschritt lässt sich Aufgrund der Zusammenhänge zwischen der formalen Semantik von RDF und SiRDF automatisieren. Mehr noch, dieser Transferschritt lässt sich automatisch bidirektional implementieren (engl.: round tripping).

Wir haben damit eine vollständige Transferkette für den gesamten Modellierungsprozess, vom mentalen Modell des Anwenders in das konzeptuelle Modell von RDF und wieder zurück. Weil SiRDF als Zwischenebene verwendet wird, ist die kognitive Komplexität des Modellierens für den Anwender verringert.

In diesem Kapitel wird die Spezifikation von SiRDF vorgestellt. SiRDF definiert ein neues Datenmodell, und darauf aufbauend eine eigene abstrakte Syntax und formale Semantik. Kapitel 5, ab Seite 29, befasst sich dann mit dem bidirektionalen Transformationsprozess zwischen RDF und SiRDF.

4.2 Spezifikation von SiRDF

4.2.1 Datenmodell und Konzepte

Das Datenmodell spezifiziert die Form, in der Daten und Information im Rahmen von SiRDF dargestellt werden. Das Datenmodell und die Nomenklatur der Konzepte orientiert sich bei SiRDF stärker als bei RDF an der Tatsache, das von Graphen und ihren Bestandteilen gesprochen wird. Im SiRDF Modell gibt

es nur 3 Arten von Entitäten: dem SiRDF Graph, den Aussageknoten und den Wertknoten.

SiRDF Graph Ein SiRDF Graph besteht aus **Aussageknoten** und **Wertknoten**.

Aussageknoten Ein Aussageknoten (englisch: statement node) ist die atomarste Form von „Bedeutung“ („meaning“) in einem SiRDF Graph. Jeder Aussageknoten besteht aus einem Subjekt, einem Prädikat und einem Objekt. Ein Aussageknoten besitzt ausserdem einen eindeutigen Kennzeichner, mit dem er adressiert werden kann.

Wertknoten Jeder Wertknoten (engl.: value node) besteht aus einem Wert und einem eindeutigen Kennzeichner. Ein Wertknoten kann mit seinem Kennzeichner adressiert werden.

4.2.2 Abstrakte Syntax

Die abstrakte Syntax legt die Regeln fest, nach denen ein SiRDF Graph erstellt werden muss, sie baut direkt auf dem Datenmodell und den Konzepten auf.

Definition 4.1: SiRDF Graph Ein SiRDF Graph ist ein Tupel $< S, V >$, dabei ist S die Menge der Aussageknoten und V die Menge der Wertknoten des Graph.

Ein Wertknoten ist ein Tupel $< U, L >$ mit einer URI Referenz U und einem Wert L. U und L sind disjunkte Teilmengen des Vokabulars, auf welchem der Graph definiert ist.

Ein Aussageknoten ist ein Tupel $< U, \{A, W\} \times \{A, W\} \times \{A, W\} >$ mit einer URI Referenz U und einem Tripel, welches aus 3 Elementen besteht. Jedes dieser Elemente kann ein Aussageknoten A oder ein Wertknoten W sein.

Inhalt eines SiRDF Graph Ein SiRDF Graph besteht aus beliebig vielen **Wertknoten**, und beliebig vielen **Aussageknoten**.

Aufbau eines Aussageknoten Jeder **Aussageknoten** besteht aus einem Subjekt, einem Prädikat und einem Objekt. Jede Rolle innerhalb eines Aussageknotens kann entweder von einem **Wertknoten** oder von einem Aussageknoten eingenommen werden.

Kennzeichner von Aussagenknoten und Wertknoten Ein Kennzeichner wird stets durch eine **URI Referenz** repräsentiert.

Definition einer URI Referenz *URI Referenzen* sind in SiRDF und RDF gleich zu handhaben. Siehe dazu die Beschreibung von *URI Referenzen* in Abschnitt 3.1.2, Seite 12.

Werte von Knoten Der **Wert** eines Knotens ist eine Unicode Zeichenkette [Con03]. Der Wert ist optional.

Definition von Gleichheit

- Knoten sind genau dann *gleich*, wenn ihre Kennzeichner und ihre Werte gleich sind.
- Tripel sind genau dann *gleich*, wenn ihre Rollen jeweils durch gleiche Knoten oder Aussagen belegt sind.
- Aussagen sind genau dann *gleich*, wenn ihre Tripel gleich sind.
- Graphen sind genau dann *gleich*, wenn wenn alle ihre Knoten und alle ihre Aussagen gleich sind.

4.2.3 Formale Semantik

Aufbauend auf dem Datenmodell und der abstrakten Syntax ist die formale Semantik von SiRDF definiert. Die formale Semantik spezifiziert eine Interpretation auf dem SiRDF Modell, welche es erlaubt, eine mögliche Welt zu beschreiben um den Wahrheitswert für einen SiRDF Graph in dieser Welt zu bestimmen. Die Semantik wird in zwei aufeinander aufbauenden Schritten definiert. Der erste Schritt ist die Definition der Interpretation eines SiRDF Graphen. Der zweite Schritt ist die Definition der Denotation eines ganzen SiRDF Graphen.

Die Interpretation eines SiRDF Graphen Der erste Schritt ist die Definition der Bestandteile einer Interpretation. Die Abbildungen, welche die Interpretation bilden, beschreiben zusammen eine Modell der Wirklichkeit, eine mögliche Welt.

Definition 4.2: Interpretation eines SiRDF Graphen Das Vokabular **V** ist die Menge $V = U \cup L$, U und L sind disjunkt.

- **U** ist die Menge der *URI Referenzen*. Jeder Wertknoten und jeder Aussageknoten wird mit einer *URI Referenz* $u \in U$ identifiziert.
- **L** ist die Menge der *Literale*, welche wir in SiRDF auch als *Werte* bezeichnen.
- **IL** : $L \longmapsto U$ ist eine injektive Abbildung, welche jedem *Literal* eine *URI Referenz* zuordnet.
- **SExt** : $U \longmapsto < U \times U \times U >$ ist eine injektive Abbildung von der Menge U der *URI Referenzen* in die Menge der Tripel, welche sich aus U bilden lassen.

Eine Interpretation **I** auf dem Vokabular V ist definiert durch:

- die Menge **IR** aller *Ressourcen* von I. IR wird das Universium oder auch die Domäne von I genannt.
- die injektive Abbildung **IS** : $U \longmapsto IR$ der *URI Referenzen* in U auf das Universum von I

- die Abbildung $\mathbf{IExt} : IR \longmapsto < IR \times IR >$ vom Universum IR in das Kreuzprodukt $IR \times IR$, welches die Menge der Paare aus IR ist.

Erklärungen Im Vokabular V befinden sich sowohl *URI Referenzen* als auch *Literale*. Jedes *Literal* wird durch die injektive Abbildung $IL()$ auf eine *URI Referenz* abgebildet, weil jeder Wertknoten eine *URI Referenz* hat, durch den er adressiert werden kann, und einen Wert, der aber auch leer sein darf. Durch $SExt()$ wird die *URI Referenz*, durch welche ein Aussageknoten adressiert werden kann, auf das Tripel abgebildet, welches durch das Subjekt, das Prädikat und das Objekt des Aussageknotens gebildet wird. Das ermöglicht auch die Interpretation eines Verweises von einem Aussageknoten auf einen anderen Aussageknoten.

Die Verbindung zwischen dem Vokabular V und dem Universum der Ressourcen IR wird durch $IS()$ hergestellt. Genauer genommen bildet $IS()$ nur *URI Referenzen* in U auf *Ressourcen* in IR ab. Jedes Element des Graphen muss vor der Interpretation auf eine *URI Referenz* abgebildet werden, was durch die Kennzeichnung von Aussageknoten und Wertknoten mit *URI Referenzen* ermöglicht wird. $IExt()$ ermöglicht es, für ein Element aus dem Universum IR, eine Erweiterung zu einer binären Relation auf IR zu definieren. Die Interaktion der Bestandteile einer Interpretation wird später an einem Beispiel deutlich werden.

Die Denotation eines SiRDF Graphen Der zweite Schritt ist die Definition der Bestimmung eines Wahrheitswerts für einen Graph bezüglich einer Interpretation. Dies entspricht der Überprüfung einer Gruppe von Fakten in einer möglichen Welt. Die Bestimmung eines Wahrheitswerts wird Denotation genannt. Die Denotation eines SiRDF Graph ist rekursiv über seine Bestandteile definiert.

Definition 4.3: Denotation eines SiRDF Graph

- Sei E ein **Wertknoten** mit Wert W und der *URI Referenz* U, dann ist $I(E) = IS(U)$. Für die Bestandteile des Wertknotens ist $I(U) = IS(U)$ und $I(W) = IS(IL(W))$, so das für den gesamten Wertknoten gilt:

$$I(E) = I(U) = I(W) = IS(U)$$

- Sei E ein **Aussageknoten** mit der *URI Referenz* U und den *URI Referenzen* S,P,O, welche belegt werden durch $SExt(U) =< S, P, O >$.

Dann ist

$$I(E) = true \iff (IS(S), IS(P), IS(O) \in IR)$$
$$\wedge(< IS(S), IS(O) > \in IExt(IS(P)))$$

sonst ist $I(E) = false$

- Sei E ein SiRDF Graph, dann ist $I(E) = false \iff I(A) = false$ für einen beliebigen Aussageknoten $A \in E$, sonst ist $I(E) = true$

Erklärungen Die Denotation eines Wertknotens kann auf die Denotation seiner *URI Referenz* zurückgeführt werden. Da $IL()$ jedem Literal eindeutig eine *URI Referenz* zuordnet, ist die Denotation eines Wertknotens auch mit der Denotation seines Wertes identisch. Ein Wertknoten, seine *URI Referenz* und sein Wert haben die gleiche Denotation.

Um den Wahrheitswert eines Aussageknotens mit der *URI Referenz* U zu ermitteln, muss zunächst $SExt(U)$ ausgewertet werden. $SExt(U)$ liefert die *URI Referenzen* zum Subjekt, Prädikat und Objekt des Aussageknotens. Durch $IS()$ werden die *URI Referenzen* der Bestandteile des Aussageknotens auf das Universum IR von I abgebildet, so dass wir nun die Ressourcen im Universum, auf die sie verweisen, kennen. Sind Subjekt-Ressource und Objekt-Ressource in der binären Relation, welche durch die Prädikat-Ressource beschrieben wird, so ist die Denotation des Aussageknotens wahr, sonst ist sie falsch.

Die Denotation eines SiRDF Graphen ist letztlich genau dann wahr, wenn er keinen Aussageknoten enthält, dessen Denotation falsch ist. Ein leerer Graph ist genauso wahr, wie ein Graph, der nur aus Wertknoten besteht.

4.2.4 Schriftliche Notation

SiRDF besitzt keine fest vorgeschriebene Syntax, etwa zur Serialisierung eines SiRDF Graph in eine Datei. Es werden aber eine textuelle und eine graphische Notation vorgeschlagen, um einen SiRDF Graph schriftlich festzuhalten.

Notation als Text

Wertknoten `(URI "Wert")` Beispiel: `(ex:Klaus "Klaus")`
Falls der Wert leer ist, kann er auch ausgelassen werden: `(ex:test)`
Falls der Wertknoten jedoch einen Wert hat, so muss der Wert immer mit angegeben werden. Diese Redundanz ist wichtig, damit an jeder Stelle im Graph klar ist, dass der Wertknoten einen definierten Wert hat.

Aussageknoten `(URI ->> Subjekt Prädikat Objekt)` Ist eines der Elemente Subjekt, Prädikat oder Objekt eine Instanz eines anderen Aussageknoten, so kann der andere Aussageknoten durch seine URI ohne Klammern, abgekürzt

notiert werden, um etwa Rekursion zu vermeiden. Beispiel:
`(ex:Aussage1 ->> (ex:Klaus "Klaus") (ex:sagt) ex:Aussage2)`

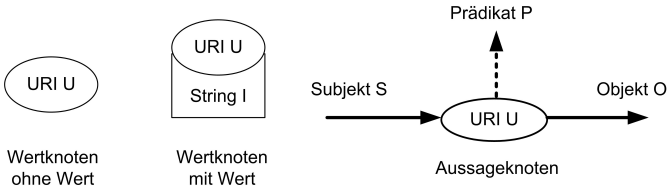

Abbildung 4.2: Vorschlag zur Notation eines SiRDF Graph als Diagramm

Notation als Diagramm Anschaulich als Diagramm lässt sich ein SiRDF Graph wie in Abbildung 4.2 vorgeschlagen festhalten. Ein konkretes Beispiel für einen SiRDF Graph ist in Abbildung 4.3 zu sehen.

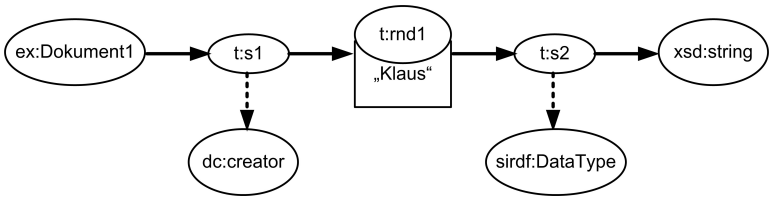

Abbildung 4.3: konkretes Beispiel für einen SiRDF Graph

4.2.5 Vokabular

SiRDF verfügt über ein normiertes Vokabular von Prädikaten. Mit diesem Vokabular können Sprachbezeichner und Datentyp eines Literals gespeichert werden, der Typ eines Knotens kann festgelegt werden, und durch das Vokabular wird das Speichern eines SiRDF *Reification* Tripels normiert. Dafür wird der Namespace `http://sirdf.ontoware.org/ns#` mit der Abkürzung `sirdf:` verwendet.

Datentypen und Sprachbezeichner

Sprachbezeichner `sirdf:LanguageTag`

Datentyp `sirdf:DataType`

Ein Knoten kann einen *Datentyp* (engl.: *data type*) haben, so dass, eine vom Datentyp definierte, Abbildung für den Wert des Knotens vom lexikalischen Raum des Datentyps in seinen Wertraum erfolgen kann. Ein Knoten kann ausserdem

einen *Sprachbezeichner* haben (engl.: *language tag*). Ein Knoten kann beliebig viele Sprachbezeichner und Datentypen haben. Sprachbezeichner und Datentyp sind keine Komponenten des Knotens, sie werden im Graph durch Aussagen über den zu kennzeichnenden Knoten repräsentiert. Beispiel für die Verwendung eines Sprachbezeichners:
`(ex:s1 ->> (ex:test1 "Mayer") (sirdf:LanguageTag) (ex:temp1 "DE"))`

Der Typ eines Knotens `sirdf:type` kann analog wie `rdf:type` verwendet werden um den Typ eines Knotens festzulegen.

Blank Nodes in SiRDF Obwohl es keine *Blank Nodes* als Entitäten in SiRDF gibt, kann ein SiRDF Wertknoten durch `sirdf:type` als Wertknoten des Typs „Blank Node“ ausgezeichnet werden. Auf diese Weise ausgezeichnete Wertknoten, können dann analog zu *RDF Blank Nodes* behandelt werden. Das ist speziell für das „round tripping“ von *Blank Nodes* zwischen RDF und SiRDF Graphen in Kapitel 5, ab Seite 29, wichtig.

Vokabular der Reification in SiRDF In SiRDF wird ein sehr ähnliches Vokabular zur *Reification* verwendet wie in RDF, siehe [Hay04, Abschnitt 3.3.1]. Der wichtigere Fall der Adressierung wird in SiRDF durch die eindeutige Adressierung von Aussagenknoten und Wertknoten realisiert. Deswegen ist die Verwendung des *Reification* Vokabulars nur notwendig, wenn in einer Aussage eine andere Aussage referenziert werden soll, ohne das der Wahrheitswert der referenzierten Aussage bekannt ist. Die Vokabeln `sirdf:Reification`, `sirdf:Subject`, `sirdf:Predicate`, `sirdf:Object` werden verwendet um *Reification* auszudrücken. Wir verwenden dafür das Beispiel „Peter sagt, das Klaus krank sei.“

```
(ex:s1 ->> (ex:Peter "Peter") (ex:sagt) (ex:IndirekteRede1))
(ex:s2 ->> (ex:IndirekteRede1) (sirdf:type) (sirdf:Reification))
(ex:s3 ->> (ex:IndirekteRede1) (sirdf:Subject) (ex:Klaus "Klaus"))
(ex:s4 ->> (ex:IndirekteRede1) (sirdf:Predicate) (ex:ist))
(ex:s5 ->> (ex:IndirekteRede1) (sirdf:Object) (ex:krank))
```

4.3 Beispiel der Interpretation eines SiRDF Graph

Das Zusammenspiel der einzelnen Funktionen einer Interpretation eines SiRDF Graphen soll nun an einem Beispiel demonstriert werden. (siehe Abbildung 4.4)

Gegeben seien folgende Wertknoten: `(ex:a) (ex:b) (ex:c)`
`(ex:d "whatever")`
Daraus ergeben sich als Bestandteile des Vokabulars V, die Menge der *URI Referenzen* $U :=$ `{ ex:a, ex:b, ex:c, ex:d }` und die Menge der Literale $L :=$ `{ "whatever" }`, wenn $IL()$ definiert ist als $IL :=$ ”whatever” $\Longrightarrow$ ex:d.

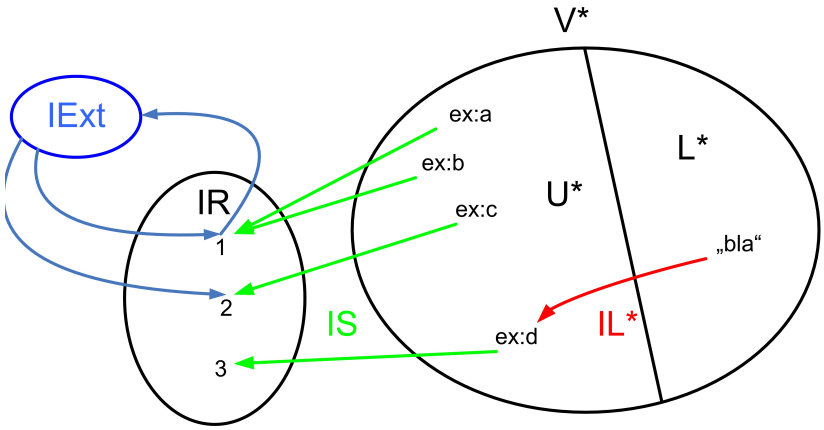

Abbildung 4.4: Illustration des Beispiels einer SiRDF Interpretation

Die Interpretation $I()$ sei nun wie folgt definiert:

- $IR := \{1, 2, 3\}$
- $IExt := (1 \Longrightarrow \{<1, 2>, <2, 1>, <1, 3>\})$
- $IS :=$ ex:a $\Longrightarrow 1$, ex:b $\Longrightarrow 1$, ex:c $\Longrightarrow 2$, ex:d $\Longrightarrow 3$

Dann sind folgende Aussagen jeweils einzelnen und auch zusammen als Graph wahr:

```
(ex:s1 ->> (ex:a) (ex:b) (ex:c))
(ex:s2 ->> (ex:c) (ex:b) (ex:a))
(ex:s3 ->> (ex:a) (ex:b) (ex:d "whatever"))
```

5 Von RDF zu SiRDF und zurück

Um den Transferprozess zwischen dem mentalen Modell des Anwenders und dem konzeptuellen Modell von RDF formal beschreiben zu können, ist es notwendig diesen Transferprozess in einzelne Schritte zu zerlegen. Diese Schritte sind in Abbildung 5.1 veranschaulicht.

Der erste Schritt ist der Transfer des mentalen Modells in ein Zielmodell. Beim Transfer in RDF als Zielmodell *(Pfeil b)* hat der Anwender mehrere Arten von Problemen zu erwarten, wie in Kapitel 2 beschrieben wurde. Entscheidet sich der Anwender für den Transfer in das Zielmodell SiRDF *(Pfeil a)*, so gestaltet sich das Modellieren des Sachverhalts einfacher und intuitiver, wie in Kapitel 4 dargestellt, ab Seite 20.

In beiden Fällen hat der Anwender aus seinem mentalen Modell ein formales Modell erstellt. Falls der gleiche Sachverhalt als RDF Modell und auch als SiRDF Modell modelliert wurde, so liegen zwei verschiedene Modelle der gleichen möglichen Welt vor. In beiden Modellen lassen sich Informationen darauf prüfen, ob sie in der formal beschriebenen, möglichen Welt war sind. Der Wahrheitswert einer Aussage oder eines Graph wird als seine Denotation bezeichnet. Um die Denotation einer Aussage zu bestimmen, muss die Interpretation der Teile der Aussage bestimmt werden.

Die Gleichheit der Denotationen eines RDF und SiRDF Graph, und die Möglichkeit in beiden Graphen Aussagen bezüglich der gleichen möglichen Welt zu machen, ist dann gegeben, wenn die Interpretation der jeweiligen Modelle die gleichen möglichen Welten beschreiben. Dazu müssen mehrere Teile einer SiRDF

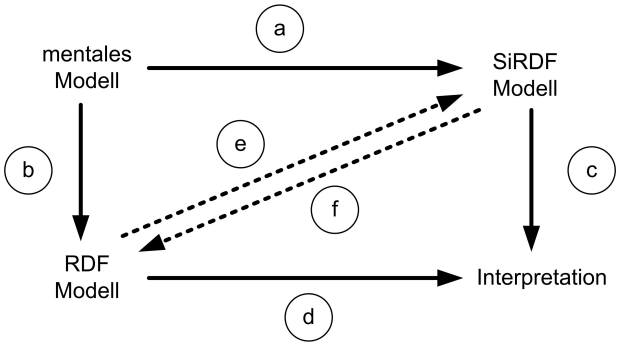

Abbildung 5.1: Transformation zwischen den Modellen

Interpretation und einer RDF Interpretation übereinstimmen.

In diesem Fall hat der Anwender sein mentales Modell sowohl in SiRDF *(Pfeil a)* als auch in RDF *(Pfeil b)* modelliert, und die Interpretation der gleichen Aussage ist dann sowohl bezüglich der SiRDF Interpretation *(Pfeil c)* als auch bezüglich der RDF Interpretation *(Pfeil d)* gleich.

In diesem Kapitel wird gezeigt werden, das jedes gegebene SiRDF Modell in ein RDF Modell transformiert werden kann *(Pfeil f)*. Dieses neue RDF Modell hat die gleiche Interpretation *(Pfeil d)* wie das SiRDF Modell *(Pfeil c)*. Umgekehrt lässt sich jedes RDF Modell in ein SiRDF Modell mit der gleichen Interpretation transformieren *(Pfeil e)*.

Das ist die formale Grundlage für eine automatische Transformation zwischen RDF und SiRDF Graphen. Das erlaubt es dem Anwender einen Sachverhalt in SiRDF zu modellieren, und danach automatisch den SiRDF Graph in einen semantisch äquivalenten RDF Graph umwandeln zu lassen. Der Anwender hat dann die Vorteile der Modellierung in SiRDF und gleichzeitig auch die Vorteile der Interoperabilität von RDF.

Für die Transformation zwischen SiRDF und RDF wird zunächst die formale semantische Grundlage beschrieben. Danach werden die Algorithmen zur Transformation zwischen SiRDF und RDF in beiden Richtungen beschrieben. Zu beiden Algorithmen wird der Pseudo-Code der Funktionen gegeben, sowie Erklärungen zur Funktionsweise der einzelnen Funktionen.

5.1 Formale semantische Grundlagen

Die semantische Grundlage der Gleichheit von Aussagen über die selbe mögliche Welt, erfordert die Gleichheit der Denotation von RDF und SiRDF Graphen mit den gleichen Informationen. Dies wiederrum erfordert die Gleichheit der Interpretation über beiden Graphen, obwohl sie in unterschiedlichen formalen Modellen vorliegen.

Deswegen wird zunächst, aufbauend auf den bisherigen Definitionen der formalen Semantik von RDF und SiRDF, spezifiziert wie RDF und SiRDF Graphen die gleiche Information ausdrücken können. Danach wird gezeigt, das unter diesen Bedingungen die Interpretation von Aussagen gleicher Bedeutung in beiden Graphen, und damit auch in beiden Modellen, eine äquivalente Denotation hat.

5.1.1 Äquivalenz der Modelle

Definition 5.1: semantische Äquivalenz zwischen RDF und SiRDF Modellen Gegeben sei eine RDF Interpretation I_R nach Definition 3.1: Sie ist auf dem Vokabular V, dem Universium IR und den Properties $IP \subset IR$ durch die Abbildungen $IExt()$, $IS()$ und $IL()$ definiert. Die Erweiterung von I um auch *Blank Nodes* zu behandeln ist $A()$. Die Denotation eines RDF Graphen ist für I_R nach Definition 3.2 und 3.3 definiert.

Weiter sei eine SiRDF Interpretation I_S nach Definitionen 4.2 gegeben. I_S ist auf dem gleichen Vokabular V und dem gleichen Universum IR wie I_R definiert. Auch die Abbildungen $IExt()$ und $IS()$ werden von I_R übernommen. Neu definiert wird $IL_S()$ und $SExt()$. Die Menge U sei die Menge der *URI Referenzen* in V, die Menge L sei die Menge der *Literale* aus V. Die Denotation eines SiRDF Graphen ist für I_S nach Definition 4.3 definiert.

Unter diesen Voraussetzungen ist die Interpretation I auf beiden Modellen gleich.

5.1.2 Zusammenfassung

In beiden Modellen müssen identisch sein:

- Das **Vokabular V** der Interpretation
- Das **Universum IR** der Interpretation
- Die **Abbildung IS()** von der Menge der Literale in V in das Universum IR
- Die **Abbildung IExt()**, welche binäre Relationen über IR mit einzelnen Elementen aus IR assoziiert

Unterschiede In I_R ist $IL_R()$ die Funktion, welche einem typisierten Literal ein Element aus dem Universum IR direkt zuordnet. In I_S ist der einzige Weg um vom Vokabular V in das Universum IR zu gelangen, die Abbildung durch $IS()$ von einer *URI Referenz* nach IR. Deswegen ist in I_S die Funktion $IL_S()$ definiert, als die Abbildung der Menge L aller Literale in die Menge U der *URI Referenzen*. Mit $IL_S()$ wird jedes Literal mit einer *URI Referenz* assoziert.

Die Abbildung SExt() muss nur für SiRDF Interpretationen definiert werden, da es kein Gegenstück dazu in der RDF Semantik gibt. SExt() bildet die *URI Referenz* eines Aussageknotens auf die *URI Referenz* des Subjekts, Prädikats und Objekts des Aussageknotens ab.

5.1.3 Beweis der Äquivalenz der Interpretationen

Die Bestandteiles eines RDF Graphen und eines SiRDF Graphen sind zwar teilweise unterschiedlich, das Modell einer möglichen Welt, welches in den Graphen enthalten ist, wird anhand des Universums IR und anhand der binären Prädikate,

welche über IExt() definiert sind, bestimmt. Der Beweis der Äquivalenz erfolgt in zwei Teilen, aufbauend auf Definition 5.1.

Beweis 5.2: Äquivalenz SiRDF $\Longrightarrow$ RDF Sei $I_S(A) = true$ für einen Aussageknoten aus einem SiRDF Graph und S,P,O seien über $SExt(A) =< S, P, O >$ definiert.
Dann ist $I_R(SPO) = true$ genau dann, wenn:

1. $S, P, O \in V$: Da $SExt()$ jeden Aussageknoten A auf ein Tripel aus der Menge U der *URI Referenzen* abbildet und $U \subset V$ ist die Bedingung erfüllt

2. $I_R(P) \in IP$: Da $P \in U$ ist, gilt nach Definition 5.1 $I_R(P) = I_S(P) = IS(P) \in IP \subset IR$

3. $< I_R(S), I_R(O) >\in IExt(I_R(P))$: Weiterhin sind $S, O \in U$, deswegen ist $< I_R(S), I_R(O) >=< IS(S), IS(O) >=< I_S(S), I_S(O) >$. Weiterhin ist $I_R(P) = I_S(P) = IS(P)$ und deswegen ist $< I_R(S), I_R(O) >\in I_S(P)$ und damit auch letzendlich $< I_R(S), I_R(O) >\in I_R(P)$

Erklärungen Die wichtigste Voraussetzung für die Äquivalenz von I_R und I_S ist die Gleichheit der Abbildung $IS()$ bei beiden Interpretationen. $IS()$ bildet *URI Referenzen* vom Vokabular V in das Universum IR ab und ist für eine SiRDF Interpretation die einzige Abbildung in das Universum der Interpretation. Die Gleichheit von $IS()$ bedeutet, das die Denotation von *URI Referenzen* in beiden Interpretationen gleich sind.

Für den Übergang von SiRDF zu RDF brauchen wir nur *URI Referenzen* in das Universum IR abbilden, und dann die binären Relationen, welche auf den Prädikaten definiert sind, zu überprüfen.

Da in SiRDF jeder Wertknoten und jeder Aussageknoten mit einer *URI Referenz* assoziert werden kann, und weil $SExt()$ zu der *URI Referenz* von jedem Aussageknoten, die *URI Referenzen* der Elemente, die zum Aussageknoten gehören, finden kann, ist die Gleichheit von $IS()$ bei beiden Abbildungen das entscheidende Kriterium für den Übergang.

Beweis 5.3: Äquivalenz RDF $\Longrightarrow$ SiRDF Sei $I_R(SPO) = true$ für ein Tripel $< S, P, O >$ aus einem RDF Graph. Wenn im SiRDF Graph ein Aussageknoten A mit $SExt(A) =< S, P, O >$ existiert, dann ist $I_S(A) = true$ genau dann wenn:

$$IS(S), IS(P), IS(O) \in IR \wedge (< IS(S), IS(O) >\in IExt(IS(P)))$$

Da jedes Prädikat in einem RDF Tripel eine *URI Referenz* ist, ist in jedem Fall $IS(P) \in IR$. Für $E \in S, O$ muss nach dem Typ von E unterschieden werden.

Einfaches Literal Sei E ein einfaches Literal. Wähle $u \in U$ so, das $IL_S(P) = u$ und $IS(u) = P$. Dann ist $IS(E) = IS(U) = IS(IL_S(E)) \in IR$

Typisiertes Literal Sei E ein typisiertes Literal. Gegeben ist $IL_R(E) \in IR$. Wähle $u \in U$ so, das $IL_R(E) = IS(u)$ und $IL_S(E) = u$. Dann ist insgesamt $IL_R(E) = IS(u) = IS(IL_S(E)) \in IR$

Blank Node Sei E ein *Blank Node*. Dann gilt für I_R nach Definition 3.3: $[I_R + A](E) = A(E)$ und $A(E) \in IR$. Wähle $u \in U$ so, das $IS(u) = A(E)$. Damit ist $I_S(E) = IS(u) = A(E) \in IR$

Dann ist $(< IS(S), IS(O) > \in IExt(IS(P))$ genau dann,
wenn das Paar $< IS(S), IS(O) >$ in der binären Relation $IExt(IS(P))$ ist.

Erklärungen *Einfache Literale* sind schon im Universum IR enthalten. Für die SiRDF Interpretation eines einfachen Literals wählen wir eine *URI Referenz* U aus, welche wir mit dem Literal assoziieren. Dadurch ist die Denotation in I_S des einfachen Literals gleich wie in I_R definiert.

Typisierte Literale werden in der RDF Interpretation I_R durch $IL_R()$ vom Vokabular in das Universum IR abgebildet. Für I_S müssen wir einem typisierten Literal eine *URI Referenz* u zuordnen, danach ist die Denotation des typisierten Literals in I_S gleich wie in I_R definiert.

Blank Nodes werden in I_R von der Erweiterung $A()$ in das Universum IR abgebildet. Wir ordnen einem *Blank Node* b eine *URI Referenz* u zu, so das die Denotation von u und die Denotation des *Blank Nodes* b gleich sind, $IS(u) = A(b)$

Insgesamt lässt sich zusammenfassen sagen, das für alle Elemente aus dem Vokabular V, welche direkt von V in das Universum IR abgebildet werden, eine *URI Referenz* für diese Elemente erzeugt werden muss, und dann das Element aus V mit dieser neuen *URI Referenz* verknüpft wird. Diese Elemente aus V sind *Blank Nodes*, *einfache Literale* und *typisierte Literale*. Die Denotation diese Elemente wird so mit der Denotation der assozierten *URI Referenz* gleich gesetzt.

5.2 Algorithmus zur Transformation von RDF zu SiRDF

Im letzten Abschnitt wurde die formale Grundlage für die semantische Äquivalenz von RDF und SiRDF Modellen gezeigt. Aufbauend auf dieser formalen Grundlage wird nun ein Algorithmus beschrieben um aus einem RDF Graph einen SiRDF Graph mit der gleichen Bedeutung zu erzeugen.

5.2.1 Zuordnung von Literalen zu URI Referenzen

Obwohl ein Literal in RDF keine Adresse hat, gibt es einen formal korrekten Weg um einem Literal eine *URI Referenz* zu zuordnen. In [Hay04, Abschnitt

3.3.4] wird beschrieben, das `rdf:value` typischerweise zur Kennzeichnung der Haupteigenschaft einer Entität mit mehreren Eigenschaften verwendet wird. Das Tripel `ex:test rdf:value "Test"` ordnet also der *URI Referenz* `ex:test` das Literal `"Test"` zu. Dieser Umweg der Assoziation einer *URI Referenz* mit einem Literal wird bei der beiden Richtungen der Konvertierung von Literalen verwendet.

5.2.2 Algorithmus zur Transformation

Die Transformation wird in Pseudo-Code beschrieben, welcher eine Mischung aus Java und Pascal darstellt. Der Algorithmus wurde zum besseren Verständnis in Blöcke aufgeteilt, zu jedem Block wird die Funktionsweise erklärt. Der Pseudo-Code ist an manchen Stellen verkürzt worden, da er sonst den Rahmen der Arbeit sprengen würde.

Die Hauptfunktion Der Algorithmus verfolgt einen möglichst strikten top-down Ansatz. Gruppen von Aussagen im RDF Graph werden zuerst geschlossen verarbeitet, wenn die Gruppe als ganzes konvertiert werden muss. Diese „Gruppen“ sind die *hybriden Literale*, *adressierte Tripel* und *reifizierte Tipel*. Erst danach werden alle Aussagen verarbeitet, welche einzeln konvertiert werden können.

Hybride Literale sind Literale die mehr als einen Sprachbezeichner und/oder mehr als einen Datentyp haben, sie werden konvertiert von der Funktion `ConvertHybridLiterals()`. *Adressierte Tripel* werden von `ConvertAdressing()` konvertiert, *reifizierte Tripel* werden von `ConvertReification()` konvertiert.

Funktion 5.1 Die Hauptfunktion

```
Convert Graph from RDF to SiRDF
    ConvertHybridLiterals()
    ConvertAdressing()
    ConvertReification()
    for each (Statement S)
        if !(S.isConverted)
            ConvertStatement( S )
```

Jeder dieser Konvertierungsfunktionen behandelt eine ganze Gruppe von Aussagen und transformiert die Bedeutung der Gruppe als Ganzes, von der Form, welche diese Gruppe in RDF hat, in die Form, welche eine Gruppe mit der gleichen Bedeutung in SiRDF hat. Alle verbleibenden Tripel, welche nicht zu einer größeren Gruppe gehören, und deswegen noch nicht konvertiert wurden, werden, nach der Konvertierung der größeren Gruppen, einzelnen mit `ConvertStatement()` konvertiert.

Hybride Literale Ein hybrides Literal ist in SiRDF durch die abstrakte Syntax erlaubt, in RDF muss es über die formale Semantik aus einer Gruppe von mehreren Tripeln zusammengesetzt werden. Diese Tripel haben das Prädikat `rdf:value`, dieselbe *URI Referenz* im Subjekt und eine gleiches Literal mit verschiedenen Sprachbezeichnern und/oder Datentypen. Ein Beispiel:

```
ex:test rdf:value  "Teststring"^^DE
ex:test rdf:value  "Teststring"^^xsd:string
```

Hybride Literale werden in der Funktion `ConvertHybridLiterals()` konvertiert. Der RDF Graph wird auf unkonvertierte Gruppen, welche ein hybrides Literal enthalten, durchsucht. Alle Sprachbezeichner und Datentypen eines hybriden Literals werden in einem Array H gespeichert. Es wird ein neuer Wertknoten N mit dem Wert des Literals und mit der *URI Referenz*, welche über `rdf:value` mit dem hybriden Literal assoziiert ist, angelegt. Für jeden Sprachbezeichner und jeden Datentyp aus H wird eine Aussage über den Wertknoten N gemacht, in der der Sprachbezeichner oder der Datentyp gespeichert werden. Abschließend werden alle Aussagen, aus denen das hybride Literal besteht, als konvertiert markiert.

Funktion 5.2 Konvertierung hybrider Literale: `ConvertHybridLiterals()`

```
for each (hybrid Literal HT)
    if !(HT.isConverted)
        Array H = Get all Tripels of the Hybrid Literal;
        N=new ValueNode(URI=Random_URI or Associated_URI,
                        Value=HybridLiteral.String )
        for each DataType D in H do
            SiRDFGraph.add(Random_URI, < N, sirdf:DataType,
                           (URI=Random_URI, Value=D) > )
        for each LanguageTag L in H do
            SiRDFGraph.add(Random_URI, < N, sirdf:LanguageTag, L > )

        MarkAsConverted(All Tripels in H)
```

Das hybride Literal aus dem Beispiel wird in SiRDF wie folgt notiert:

```
(ex:s1 ->> (ex:l1 "Teststring") (sirdf:LanguageTag) (ex:l2 "DE"))
(ex:s2 ->> (ex:l1 "Teststring") (sirdf:Datatype) (xsd:string))
```

Adressierung von Aussagen Zur Adressierung von Aussagen in RDF wird die Empfehlung der RDF Spezifikation befolgt. Ein *Reification* Tripel und ein echtes Tripel, werden miteinander assoziiert, wenn sie den gleichen Inhalt haben. Dies ist nach [Hay04, Abschnitt 3.3.1] zwar nicht die formale Semantik einer Adressierung

aber die „intendierte Semantik“, welche von einer Implementierung ausdrücklich angewandt werden darf. Die Form eines RDF *Reification* Tripels wird in [Hay04, Abschnitt 3.3.1] beschrieben.

Funktion 5.3 Konvertierung von adressierten Tripeln: `ConvertAdressing()`

```
for each (real Tripel T and Reification-Tripel R)
    if (T.equals(R) & !T.isConverted & !R.isConverted)
        ConvertStatement(R.URI or Random_URI, < T.S, T.P, T.O > )
        MarkAsConverted(R and T)
```

Nach dem ein *Reification* Tripel R und ein echtes Tripel T gleichen Inhalts identifiziert wurden, wird ein Aussageknoten erzeugt. Dieser Aussageknoten hat als Adresse die *URI Referenz* von R, und den Inhalt von T. Falls R nicht mit einer *URI Referenz* sondern mit einem *Blank Node* assoziert ist, wird eine neue *URI Referenz* für den Aussageknoten erzeugt. Der Aussageknoten muss dann noch durch `ConvertStatement()` konvertiert werden, da z.B. sein Prädikat noch eine Bedeutung haben kann, die konvertiert werden muss. Danach werden R und T als konvertiert markiert. Das Beispiel aus Abschnitt 3.1.4 nach der Konvertierung:
`(ex:RandomURI ->> (ex:a) (ex:b) (ex:c))`

Funktion 5.4 Konvertierung von Reification Tripeln: `ConvertReification()`

```
for each (Reification-Tripel R)
    if (!R.isConverted & RDF contains no equal Tripel)
        SiRDFGraph.add(Random_URI, < R.URI, sirdf:subject, R.S > )
        SiRDFGraph.add(Random_URI, < R.URI, sirdf:predicate, R.P > )
        SiRDFGraph.add(Random_URI, < R.URI, sirdf:object, R.O > )
        SiRDFGraph.add(Random_URI, < R.URI, sirdf:type, sirdf:Reification > )
        MarkAsConverted(R)
```

Reification Tripel Falls es kein echtes Tripel als Gegenstück zum *Reification* Tripel gibt, so kann Adressierung nicht die intendierte Bedeutung sein. Das RDF *Reification* Tripel wird in ein SiRDF *Reification* Tripel konvertiert. Ein Beispiel einer SiRDF *Reification* ist in Abschnitt 4.2.5 zu sehen.

Die vier RDF Aussagen, aus denen ein RDF *Reification* Tripel besteht, werden in vier SiRDF Aussagen umgeschrieben. Dabei werden die Prädikate aus dem RDF Namespace Vokabular zur Beschreibung des Subjekts, Prädikats und Objekts durch das Vokabular aus dem SiRDF Namespace mit der gleichen Bedeutung ausgetauscht.

Zuerst werden alle Aussagen-Gruppen, welche ein RDF *Reification* Tripel darstellen identifiziert. Dann werden Subjekt, Prädikat und Objekt in einem SiRDF

Reification Tripel gespeichert. Durch den Aussageknoten mit der *URI Referenz* des *Reification* Tripels als Subjekt, dem Prädikat `sirdf:type` und dem Objekt `sirdf:Reification` wird das *Reification* Tripel mit der *URI Referenz* assoziert.

Konvertierung von Aussagen `ConvertStatement()` konvertiert eine einzelne Aussage, ohne Abhängigkeiten zu anderen Aussagen zu beachten. Das heisst das Abhängigkeiten einer Aussage zu einer Gruppe von anderen Aussagen schon behandelt sein müssen, wenn `ConvertStatement()` aufgerufen wird.
`ConvertStatement()` ruft `ConvertEntity()` für alle Teile eines Tripels auf, und erzeugt aus diesen Teilen ein neues Statement. Für das erzeugte Statement kann eine Adresse angegeben werden, oder es wird eine neue *URI Referenz* erezugt, welche eindeutig im neuen SiRDF Graph ist.

Funktion 5.5 Konvertierung einer Aussage: `ConvertStatement(Statement S, Address A)`

```
if (S.P == rdf:value)
    if (RDF Graph contains <x,?,?> or <?,x,?> or <?,?,x>)
        return null;
if A=null then A=Random_URI
SiRDFGraph.add(A, < ConvertEntity(S.S), ConvertEntity(S.P),
                    ConvertEntity(S.O) > )
MarkAsConverted(S)
```

Zu Beachten ist, das das Statement S in genau einem Fall nicht konvertiert werden darf: Ist `rdf:value` das Prädikat von S, so darf S nicht konvertiert werden, falls das Objekt von S ein Literal ist. `rdf:value` kann nach [Hay04, 3.3.4] auf viele verschiedene Arten verwendet werden. Der Konvertierungsalgorithmus hat nur für Literal-Objekte eine Bedeutung definiert: S ist in diesem Fall eine Aussage, die ein Literal mit einer *URI Referenz* assoziiert. Diese Art von Aussagen werde nicht direkt konvertiert, sondern nur im Rahmen von `ConvertEntity()` behandelt, wenn dort die *URI Referenz* konvertiert werden soll. Ein Beispiel:
`ex:c rdf:value "Sashimi"` wird nicht von `ConvertStatement()` konvertiert.
`ex:a ex:b ex:c` wird von `ConvertStatement()` konvertiert, und zwar durch Aufruf von `ConvertEntity(ex:a)`, `ConvertEntity(ex:b)` und `ConvertEntity(ex:c)`. Letzterer Aufruf bemerkt das `ex:c` mit dem Literal `"Sashimi"` assoziiert ist, und so ergibt sich im SiRDF Graph:
`(ex:s1 ->> (ex:a) (ex:b) (ex:c "Sashimi"))`

Konvertierung von Entitäten `ConvertEntity()` ist die Funktion, welche die atomaren Bestandteile eines RDF Graph konvertiert, die RDF Entitäten. Dies sind *URI Referenzen*, *Blank Nodes*, einfache und typisierte Literale.

Funktion 5.6 Konvertierung einer Entität: `ConvertEntity(E)`

```
if (E allready converted)
    return lookup(E)
if E is URI
    if (E has associated Literal L)
        return ValueNode(URI=E, Value=L)
    else
        return ValueNode(URI=E)
if E is BlankNode
    Skolemize E with Random URI U
    if (E has associated Literal L)
        V = ValueNode(URI=U, Value=L)
    else
        V = ValueNode(URI=U)
    SiRDFGraph.add(Random_URI, < V, sirdf:value,
                                 sirdf:BlankNode > )
    return V
if E is Literal
    V = valueNode(URI=Random_URI, Value=E)
    if E is Plain Literal
        return V
    if E is Literal with Datatype or Language Tag
        Put V with its Datatype or Language Tag into SiRDF Graph
        return V
```

`ConvertEntity()` bekommt die zu konvertierende Entität E als Argument. Zuerst wird geprüft ob E schon einmal konvertiert wurde, und wenn ja, wird die SiRDF Entität, welche mit E durch die Konvertierung assoziert wurde, zurückgegeben. Dadurch wird verhindert, das eine RDF Entität in zwei verschiedene SiRDF Entitäten konvertiert wird.

Falls E eine *URI Referenz* ist, muss geprüft werden ob über `rdf:value` ein Literal mit dieser *URI Referenz* verknüpft ist. Dann wird aus der *URI Referenz* und dem Literal, falls vorhanden, ein neuer Wertknoten gebildet. Dieser wird an `ConvertStatement()` zurückgegeben (diese Funktion ruft als einzige `ConvertEntity()` auf.

Analog werden *Blank Nodes* konvertiert. Da ein *Blank Node* keine *URI Referenz* hat, muss eine neue eindeutige *URI Referenz* erzeugt werden. Dieser Vorgang der Umbenennung des *Blank Nodes* wird Skolemisierung genannt. Die neue *URI Referenz* und das eventuell vorhandene Literal, welches durch `rdf:value` mit dem *Blank Node* assoziert sein kann, werden in einem neuen Wertknoten gespeichert. Dieser wird von der Funktion zurückgegeben.

Für Literale wird eine neue eindeutige *URI Referenz* erzeugt, so das ein gültiger Wertknoten aus dem Literal und der *URI Referenz* gebildet werden kann. Falls

ein Sprachbezeichner oder ein Datentyp vorhanden ist, wird dies in einer Aussage über den Wertknoten gespeichert.

5.3 Algorithmus zur Transformation von SiRDF zu RDF

Im letzten Abschnitt wurde ein Algorithmus zur Transformation eines RDF Graph in einen semantisch äquivalenten SiRDF Graph beschrieben. Aufbauend auf der selben formalen Grundlage wird nun ein Algorithmus für die Rückrichtung beschrieben, um aus einem SiRDF Graph einen RDF Graph mit der gleichen Bedeutung zu erzeugen.

Die Hauptfunktion Der Algorithmus zur Transformation von SiRDF zu RDF hat die selbe Struktur wie der Algorithmus zur Transformation von RDF zu SiRDF.

Funktion 5.7 Die Hauptfunktion

```
Convert Graph from SiRDF to RDF
    ConvertHybridLiterals()
    ConvertAdressing()
    ConvertReification()
    for each (Statementnode S)
        if !(S.isConverted)
            ConvertStatement(S)
    for each (Valuenode V)
        if !(V.isConverted)
            ConvertEntity(V)
```

Zuerst werden Funktionen aufgerufen um Gruppen von Aussageknoten zu konvertieren, in denen hybride Literale, adressierte Aussageknoten und *Reification* Aussagen enthalten sind. Hybride Literale werden von `ConvertHybridLiterals()` konvertiert. Adressierte Aussageknoten werden von `ConvertAdressing()` konvertiert. *Reification* Aussagen werden von `ConvertReification()` konvertiert.

Jede dieser Funktionen konvertiert eine ganze Gruppe in die entsprechende Gruppe von RDF Aussagen. Nachdem alle zusammengehörigen Gruppen von Aussageknoten konvertiert wurden, werden alle übrig gebliebenen Aussageknoten von `ConvertStatement()` konvertiert. Ein SiRDF Graph darf auch Wertknoten enthalten, welche mit keiner Aussage verbunden sind, diese werden zum Schluss mit `ConvertEntity()` konvertiert.

Hybride Literale Die hybriden Literale müssen zuerst im SiRDF Graph identifiziert werden. Dazu werden Gruppen von SiRDF Aussageknoten gesucht, bei

denen der gleiche Wertknoten das Subjekt ist, und bei denen das Prädikat die *URI Referenz* `sirdf:DataType` oder `sirdf:LanguageTag` hat.

Funktion 5.8 Konvertierung hybrider Literale: `ConvertHybridLiterals()`

```
for each (hybrid Literal H)
    if !(H.isConverted)
    Array H = Get all Tripels of the Hybrid Literal;
    for each DataType D in H do
        RDFGraph.add( < Hybrid-Literal.URI, rdf:value,
                        Hybrid-Literal.Value^^D > )
    for each LanguageTag L in H do
        RDFGraph.add( < Hybrid-Literal.URI, rdf:value,
                        Hybrid-Literal.Value^^L > )
    MarkAsConverted(All Tripels in H)
```

Alle Aussagen einer gefundenen Gruppe werden in das Array H gelegt. Zur Konvertierung des hybriden Literals werden mehrere gleiche Literale mit verschiedenen Sprachbezeichnern und Datentypen über `rdf:value` mit einer einzelnen *URI Referenz* in Verbindung gebracht. Abschließend werden alle Aussageknoten, aus denen das hybride Literal besteht, als konvertiert markiert.

Adressierung von Aussageknoten Gibt es in einem SiRDF Graph einen Aussageknoten, welcher einen anderen Aussageknoten referenziert, so wird zur Kennzeichnung des zweiten Aussageknotens seine *URI Referenz* verwendet. Bei der Konvertierung in einen RDF Graph muss nicht nur der adressierte Aussageknoten konvertiert werden, es muss auch ein *Reification* Tripel mit dem gleichen Inhalt erzeugt werden. Dieses *Reification* Tripel hat eine Adresse. Wenn das echte Tripel und das inhaltsgleiche *Reification* Tripel wie in [Hay04, Abschnitt 3.3.1] beschrieben, miteinander assoziert werden, dann kann ein RDF Tripel adressiert werden.

Funktion 5.9 Konvertierung von adressierten Aussageknoten: `ConvertAdressing()`

```
for each (Statementnode S)
    if !(S.isConverted) & S.isAddressedByOtherStatement
        NewStatement = ConvertStatement(S)
        MarkAsConverted(S)
        RDFGraph.add( < S.URI, rdf:subject, NewStatement.S > )
        RDFGraph.add( < S.URI, rdf:predicate, NewStatement.P > )
        RDFGraph.add( < S.URI, rdf:object, NewStatement.O > )
        RDFGraph.add( < S.URI, rdf:type, rdf:Statement > )
```

Bevor die vier Aussagen des *Reification* Tripel zum RDF Graph hinzugefügt werden können, muss zuerst der Statementknoten S konvertiert werden. Nachdem S konvertiert und zum RDF Graph hinzugefügt wurde, wird S als konvertiert markiert. Danach können Subjekt, Prädikat und Objekt vom konvertierten S, in den vier Aussagen des *Reification* Tripel gespeichert werden.

Reification Enthält der Graph eine Gruppe von SiRDF *Reification* Aussageknoten, so muss diese Gruppe für den RDF Graph umgeschrieben werden. Statt der Prädikate `sirdf:subject`, `sirdf:predicate` und `sirdf:object` aus dem SiRDF Namespace, werden nun die Prädikate `rdf:subject`, `rdf:predicate` und `rdf:object` aus dem RDF Namespace verwendet. Und statt des Typs `sirdf:type sirdf:Reification` wird nun `rdf:type rdf:Statement` verwendet.

Funktion 5.10 Konvertierung von reifizierten Aussageknoten: `ConvertAdressing()`

```
for each (Reification R)
    if !(R.isConverted)
        RDFGraph.add( < R.URI, rdf:subject, R.S > )
        RDFGraph.add( < S.URI, rdf:predicate, R.P > )
        RDFGraph.add( < S.URI, rdf:object, O=R.O > )
        RDFGraph.add( < S.URI, rdf:type, rdf:Statement > )
        MarkAsConverted(All Statementnodes of R)
```

Zuerst muss der SiRDF Graph auf noch unkonvertierte Gruppen von Aussageknoten, in welchen eine SiRDF *Reification* dargestellt wird, durchsucht werden. Jede gefundene Gruppe wird in die entsprechende Gruppe von RDF Aussagen umgeschrieben, und dann werden die SiRDF Aussageknoten im Graph als konvertiert markiert.

Funktion 5.11 Konvertierung eines Aussageknotens: `ConvertStatement(Statementnode S)`

```
RDFGraph.add( < ConvertEntity(S.S), ConvertEntity(S.P),
                ConvertEntity(S.O) > )
MarkAsConverted(S)
```

Konvertierung von Aussageknoten Die Funktion `ConvertStatement()` wird von Funktionen aufgerufen, die eine Gruppe von Aussagen behandeln. Nachdem die Bedeutung der Gruppe als ganzes konvertiert wurde, sind abhängig vom Typ der Gruppe, noch einzelne Aussageknoten übrig, welche ohne die Kenntnis

der ganzen Gruppe konvertiert werden können. Aussageknoten, welche zu keiner Gruppe gehören, werden auch von `ConvertStatement()` konvertiert.

Die Bestandteile des Statementknotens S werden jeweils einzelnen konvertiert. Dann wird ein neues RDF Statement aus den konvertierten Bestandteilen von S gebildet, und zum RDF Graph hinzugefügt. Danach wird S als konvertiert markiert.

Funktion 5.12 Konvertierung eines Wertknotens: `ConvertEntity(E)`

```
if (E allready converted)
    return lookup(E)
if (E has sirdf:type BlankNode)
    R = new RDF Blank Node
    MarkAsConverted(Statementnode with sirdf:type BlankNode)
if (E has sirdf:DataType)
    R = E.URI
    RDFGraph.add( < E.URI, rdf:value,
                    datatyped Literal(E.Value, Datatype) > )
    MarkAsConverted(Statementnode with Datatype)
if (E has sirdf:LanguageTag)
    R = E.URI
    RDFGraph.add( < E.URI, rdf:value,
          Literal with LanguageTag(E.Value, LanguageTag) > )
    MarkAsConverted(Statementnode with LanguageTag)
if (E has non empty Value)
    R = E.URI
    RDFGraph.add( < E.URI, rdf:value,
                    plain Literal(E.Value) > )
else
    R = E.URI
MarkAsConverted(E)
return R;
```

Konvertierung eines Wertknotens `ConvertEntity()` konvertiert einen einzelnen Wertknoten E. Zuerst wird überprüft ob der Wertknoten E schon konvertiert wurde. Wenn ja, wird die RDF Entität in die E konvertiert wurde zurückgegeben. Das verhindert, das eine SiRDF Entitäten in zwei verschiedene RDF Entitäten konvertiert wurde.

Gibt es einen Aussageknoten über E, mit dem Prädikat `sirdf:type` und dem Objekt `sirdf:BlankNode`, dann wird ein neuer BlankNode erzeugt. BlankNodes müssen für das „round tripping“ zwischen RDF und SiRDF konvertiert werden, ohne das sie verloren gehen, obwohl es in SiRDF keine *Blank Nodes* gibt.

Falls ein Datentyp über einen Aussageknoten mit dem Wertknoten E assozi-

iert ist, wird ein typisiertes Literal erzeugt. Falls ein Sprachbezeichner über einen Aussageknoten mit dem Wertknoten E assoziiert ist, wird ein Literal mit Sprachbezeichner erzeugt. Der Aussageknoten in dem der Datentyp oder der Sprachbezeichner gespeichert wurden kann nun als konvertiert markiert werden.

Falls kein Sprachbezeichner oder Datentyp mit E assoziert ist, aber der Wert des Wertknotens E nicht leer ist, so wird ein einfaches Literal erzeugt. Zu beachten ist, das alle Literale durch `rdf:value` mit der *URI Referenz* des Wertknotens, aus dem sie erzeugt wurden, assoziert werden. Dies ermöglicht die Adressierung aller konvertierten Literale.

Falls der Wert eines Wertknotens leer ist, wird seine *URI Referenz* gespeichert. Nach der Fallunterscheidung über die Art der zu erzeugenden RDF Entität, wird die neu erzeugte RDF Entität als konvertiert markiert und an den Aufrufer von `ConvertEntity()` zurückgegeben.

6 Implementierung

6.1 Aufbau der Implementierung

Die gesamte Implementierung wurde in Java realisiert. Java ist eine Programmiersprache, welche für die Forschung im Bereich des Semantic Web sehr weit verbreitet ist. Bibliotheken aus diesem Bereich sind deswegen in Java in der gleichen oder sogar besserer Qualität zu finden, als für andere Programmiersprachen. Die Implementierung steht unter `http://ontoware.org/projects/sirdf/` zum Download bereit. Der Quell-Code steht unter der GNU Lesser General Public License (LPGL).

Die SiRDF Implementierung besteht aus einem *Application Program Interface* (API), welches dem Anwender den vollen Funktionsumfang von SiRDF zur Verfügung stellt und einer Implementierung des Transformationsalgorithmus, welcher jeden SiRDF Graph in seinen semantisch äquivalenten RDF Graph und wieder zurück umwandeln kann. Die Objekte der API sind in Abbildung 6.1 zu sehen.

6.2 Die API von SiRDF

Mit der API von SiRDF wird es dem Anwender ermöglicht von allen Eigenschaften des SiRDF Modells bei der Modellierung eines Sachverhalts zu profitieren. Die wichtigste Priorität beim Implementieren der API war es, die Einfachheit des

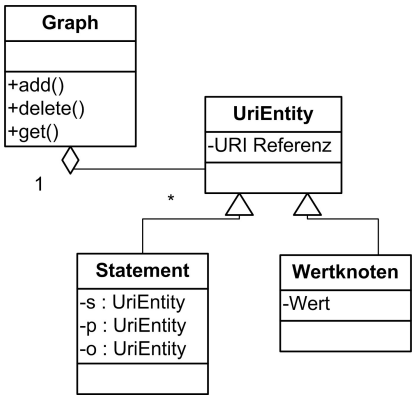

Abbildung 6.1: UML Diagramm der SiRDF API Objekte

SiRDF Modells nicht durch eine komplexe API zu negieren.

Die SiRDF API befindet sich im Java Paket `org.ontoware.sirdf`. Wie in Kapitel 4 ausgeführt, besteht das Datenmodell von SiRDF nur aus drei Arten von Entitäten: dem SiRDF Graph, den Aussageknoten und den Wertknoten.

Erzeugung eines SiRDF Graph Jeder SiRDF Graph ist eine Instanz der Klasse `Graph`. Eine Instanz eines neuen SiRDF Graph wird mit dem Konstruktor ohne Argumente erzeugt: `Graph newGraph = new Graph();`

Erzeugen eines Wertknotens Alle Wertknoten sind Instanzen der Klasse `Node`. Wird dem Konstruktor ein String und ein URI Objekt übergeben, so wir ein Wertknoten mit dem Wert des Strings und der gegebenen *URI Referenz* erzeugt. Der Konstruktor lässt sich auch ohne einen String aufrufen, dann wird nur die *URI Referenz* initialisiert. Alternativ lässt sich der Konstruktor nur mit einem String aufrufen, dann wird die *URI Referenz* mit einer zufällig für den Graph erzeugten und eindeutigen URI initialisiert. Der Wertknoten kann direkt zum Graph hinzugefügt werden mit `Graph.add(Node)`.

Die abstrakte Oberklasse UriEntity Da es viele funktionale Überschneidungen im Code von Wertknoten und Aussageknoten gibt, wurde dieser Code in die abstrakte Oberklasse `UriEntity` ausgelagert. `Node` und `Statement` erben beide von dieser Klasse.

Erzeugen eines Aussageknotens Aussageknoten sind Instanzen der Klasse `Statement`. Ein Aussageknoten wird aus einer *URI Referenz*, und drei Instanzen der `UriEntity` Klasse konstruiert. Jede der drei Rollen Subjekt, Prädikat und Objekt können jweils von einem Aussageknoten oder einem Wertknoten eingenommen werden. Die *URI Referenz* für den Aussageknoten kann auch leer gelassen werden, dann wird eine neue eindeutige *URI Referenz* für den Aussageknoten erzeugt.
`Statement s = new Statement(node1, node2, s2);`
Danach kann ein Aussageknoten zum Graph dazugefügt werden mit
`Graph.add(Statement)`. Dadurch werden auch alle anderen Entitäten, welche mit dem Aussageknoten s verbunden sind, in den Graph eingefügt.

6.3 Die Infrastruktur-Schicht

Die Implementierung des Transformationsalgorithmus von SiRDF zu RDF und zurück, basiert auf RDF2Go. RDF2Go ist eine Java API, welche eine Abstraktionsschicht für die Verwendung mehrerer RDF Bibliotheken bietet. RDF2Go definiert eine Minimal-API als den kleinsten gemeinsamen Nenner, der unterstützen

RDF Implementierungen. Dies sind zur Zeit: Jena [McB01], YARS [Dec05], NG4J [BCW05] und Sesame [BKvH02]. Die Implementierungsziele von RDF2Go sind ein hohes Maß an Portabilität, Performanz und eine einfache API. Diese einfache API wird für die SiRDF Implementierung ausgenutzt. Alle Teile des Transformationsalgorithmus, welche einen RDF Graph manipulieren müssen, verwenden dafür die RDF2Go API. RDF2Go wird ausführlich in [VEK+05, Kapitel 2] beschrieben.

6.4 Der Transformations-Algorithmus

Die Implementierung der Transformation befindet sich im Java Paket `org.ontoware.sirdf.transformer`. Eine neue Instanz wird mit `new Transformer()` erzeugt.

Dann kann der SiRDF Graph mit `Transformer.getSiRDFGraph()` manipuliert werden, und der RDF Graph mit `Transformer.getRDFGraph()`.

Die Transformation von SiRDF zu RDF wird mit `Transformer.SiRDF2RDF()` gestartet, von RDF zu SiRDF mit `Transformer.RDF2SiRDF()`.

6.4.1 Beispiel

Hier der Java Code für die Transformation eines sehr kleinen SiRDF Graphen, in dem eine Aussage eine andere Aussage adressiert.

Java Beispiel 6.1 Adressierung von Aussagen

```
public void testDemo() throws Exception {
    Transformer t = new Transformer();
    Graph myGraph = t.getSiRDFGraph();

    Node a = new Node(new URI("t:a"));
    Node b = new Node(new URI("t:b"));
    Node c = new Node(new URI("t:c"));

    Node n2 = new Node("Sashimi");
    Node n3 = new Node(URIUtils.createURI("ex:Klaus"), "Klaus");

    Statement s = new Statement(a, b, c);

    myGraph.add(s);
    myGraph.add(s, n2, n3);

    t.SiRDF2RDF();
}
```

Und hier ist zuerst der auf so konstruierte SiRDF Graph und der daraus erzeugte RDF Graph.

SiRDF Graph 6.1 Adressierung von Aussagen

```
( sirdf://ID/0 ->> (t:a) (t:b) (t:c) )
( sirdf://ID/1 ->> sirdf://ID/0 (sirdf://ID/2 "Sashimi")
                                (ex:Klaus "Klaus") )
```

RDF Graph 6.1 Adressierung von Aussagen

t:a	t:b	t:c
sirdf://ID/0	rdf:object	t:c
sirdf://ID/0	rdf:predicate	t:b
sirdf://ID/0	rdf:subject	t:a
sirdf://ID/0	rdf:type	rdf:Statement
sirdf://ID/0	sirdf://ID/2	ex:Klaus
sirdf://ID/2	rdf:value	Sashimi
ex:Klaus	rdf:value	Klaus

7 Evaluation

In den vorherigen Kapiteln wurden die verschiedenen Aspekte von SiRDF beschrieben. Es wurde das Datenmodell definiert, eine darauf aufbauende abstrakte Syntax, welche die Form eines SiRDF Graph beschreibt, und die darauf aufbauende formale Semantik, welche es erlaubt Information bezüglich eines Modells einer möglichen Welt auszuwerten.

Die Motivation für das Design von SiRDF wurde in Kapitel 2 behandelt. In Kapitel 3 wurden die formalen Hintergründe von RDF analysiert, welche dazu führen, das beim Transfer eines mentalen Modells in das Modell von RDF Schwierigkeiten auftreten.

Deswegen wird in diesem Kapitel untersucht, wie sich die Wahl von SiRDF als Zielmodell auf den Modellierungsprozess für den Anwender auswirkt. Dazu wird für verschiedene Anwendungsfälle das mentale Modell des Anwenders beschrieben. Dann wird das mentale Modell in die Modelle von SiRDF und RDF übertragen. Für beide Modelle wird eine textuelle Notation der jeweiligen Graph gegeben, sowie eine graphische Notation. Danach wird überprüft, in wie fern die Modellierung mit SiRDF die Schwierigkeiten, welche in der Motivation beschrieben wurden, verringern kann.

7.1 Adressierung von Aussagen

Beschreibung In einem Satz der Form „Peter weiß, Klaus ist arbeitslos.“ geht es um die Verknüpfung von zwei verschiedenen Aussagen. Es soll beschrieben werden, das der Wahrheitswert einer Aussage bekannt ist.

mentales Modell Es gibt zwei Personen, Peter und Klaus. Über Klaus ist die Information bekannt, das er arbeitslos ist. Über Peter ist die Information bekannt, das er die Information, das Klaus arbeitslos ist, kennt.

Modellierung in RDF In RDF lässt sich diese Verknüpfung von zwei Aussagen wegen der nicht definierten formalen Semantik der *Reification* nicht ausdrücken. Es lässt sich aber wie in [Hay04, Abschnitt 3.3.1] ausgeführt, die intendierte Semantik der *Reification* ausnutzen. Peter und Klaus werden als *URI Referenzen* `ex:Klaus` und `ex:Peter` modelliert. Die Verben „weiß“ und „ist“ werden als Prädikate modelliert, deswegen müssen *URI Referenzen* für die Verben definiert werden (`ex:weiß` und `ex:ist`). „arbeitslos“ kann als *URI Referenz* oder als Literal modelliert werden, da es Objekt eines Tripels ist. Der geringere Aufwand ist

mit der Modellierung als Literal `"arbeitslos"` verbunden. Die Aussage „Klaus ist arbeitslos“ muss als echtes Tripel modelliert werden. Das erlaubt es eine Anfrage über Klaus zu machen ohne etwas über Peter wissen zu müssen. Damit die Aussage adressierbar wird, muss sie auch als *Reification* Tripel mit der Adresse `t:1` modelliert werden.

RDF Graph 7.1 Adressierung von Aussagen

```
ex:Klaus ex:ist "arbeitslos"
t:1 rdf:type rdf:Statement
t:1 rdf:subject ex:Klaus
t:1 rdf:predicate ex:ist
t:1 rdf:object "arbeitslos"
ex:Peter ex:weiß t:1
```

Modellierung in SiRDF In SiRDF modellieren wir Peter und Klaus am einfachsten als Wertknoten mit den Werten „Klaus“ und „Peter“ und jeweils zufällig erzeugten *URI Referenzen*. Genauso simpel lassen sich auch „ist´´, „weiß´´ und „arbeitslos´´ modellieren. Der Aussage „Klaus ist arbeitslos´´ geben wir die Adresse `ex:Wichtig`.

SiRDF Graph 7.1 Adressierung von Aussagen

```
(ex:Wichtig ->> (t:rnd1 "Klaus") (t:rnd2 "ist") (t:rnd3 "arbeitslos"))
(t:rnd4 ->> (t:rnd5 "Peter") (t:rnd6 "sagt") ex:Wichtig)
```

Vergleich der Modellierung Der SiRDF Graph, gibt den Sachverhalt der Verknüpfung von zwei Aussagen durch Adressierung deutlicher wieder. Der RDF Graph wirft für den ungeschulten Betrachter die Frage nach der Redundanz des *Reification* Tripels auf. Für den SiRDF Graph müssen sehr viele *URI Referenzen* definiert werden, oder es muss die Verwendung von zufällig erzeugten *URI Referenzen* in Kauf genommen werden. Dies führt aber bei größeren Graphen dazu, das jedes Element des Graphen explizit adressierbar ist, und die Interoperabilität zwischen verschiedenen SiRDF Graphen enorm erleichtert wird, da jede *URI Referenz* global gültig ist. Trotzdem ist der SiRDF Graph schlanker als der RDF Graph.

7.2 Indirekte Rede und der Konjunktiv

Beschreibung Peter sagt, das Klaus krank sei. Er ist sich aber nicht sicher, ob Klaus das wirklich ist.

mentales Modell Der Information „Klaus ist krank“ kann kein gesicherter Wahrheitswert zugeordnet werden. Der Konjunktiv „sei“ muss in der Verknüpfung der beiden Aussagen deutlich werden. Für das Vokabular der Aussagen selbst gilt, alles was auch in Abschnitt 7.1 gesagt wurde.

Modellierung in RDF Der Konjunktiv kann formal semantisch nicht ausgedrückt werden. Nach [Hay04, Abschnitt 3.3.1] lässt sich aber das Fehlen eines gesicherten Wahrheitswerts für die Information „Klaus ist krank“ lässt sich dadurch ausdrücken, das keine echte Aussage mit dieser Information im RDF Graph vorkommt, sondern nur ein *Reification* Tripel mit diesem Inhalt. Dieses *Reification* Tripel kann dann auch zur Adressierung verwendet werden.

RDF Graph 7.2 Indirekte Rede mit der RDF Reification

```
t:2 rdf:type rdf:Statement
t:2 rdf:subject ex:Klaus
t:2 rdf:predicate ex:ist
t:2 rdf:object "krank"
ex:Peter ex:sagt t:2
```

Modellierung in SiRDF Entscheidend für die Modellierung in SiRDF ist, das der Wahrheitswert der Information „Klaus ist krank“ unbekannt ist. Deswegen darf auch im SiRDF Graph diese Information nicht als vollwertige Aussage modelliert werden. Es bietet sich deswegen die Modellierung als SiRDF *Reification* an. Dazu wird das SiRDF Vokabular `sirdf:Reification`, `sirdf:subject`, `sirdf:predicate` und `sirdf:object` verwendet.

SiRDF Graph 7.2 Indirekte Rede mit der SiRDF Reification

```
(t:rnd1 ->> (ex:Unklar) (sirdf:type) (sirdf:Reification))
(t:rnd2 ->> (ex:Unklar) (sirdf:subject) (t:rnd3 "Klaus"))
(t:rnd4 ->> (ex:Unklar) (sirdf:predicate) (t:rnd5 "ist"))
(t:rnd5 ->> (ex:Unklar) (sirdf:object) (t:rnd6 "krank"))
(t:rnd7 ->> (t:rnd8 "Peter") (t:rnd9 "sagt") (ex:Unklar))
```

Vergleich der Modellierung Der Konjunktiv der indirekten Rede und die Unklarheit über den Wahrheitsgehalt der Aussage müssen in beiden Graphen fast

gleich umständlich modelliert werden. In SiRDF ist aber eine klare Semantik für Informationen, welche mit keinem Wahrheitswert assoziert sind definiert. Diese Information sind als SiRDF *Reification* Tripel zu modellieren. Dies ist aber der sehr viel seltenere Anwendungsfall. Alle anderen Fälle, in denen gültige Aussage adressiert werden muss, können direkt als Adressierung modelliert werden.

7.3 Blank Node oder Literal?

Beschreibung Das Dokument D hat einen Autor, von dem nur der Name „Klaus“ bekannt ist. Es ist keine Homepage und auch sonst keine Netz-Präsenz von Klaus bekannt. Es soll aber zusätzlich über Klaus gespeichert werden, das er in Karlsruhe lebt.

mentales Modell Es muss die Information gespeichert werden, daß Dokument D einen Autor mit Namen Klaus hat. Die zweite Information ist, daß der Autor von Dokument D in Karlsruhe lebt. Für Dokument D ist eine *URI Referenz* bekannt. Für den Autor nicht.

Modellierung in RDF Es bietet sich an den Autor als *Blank Node* zu modellieren. Er kann nicht als Literal `"Klaus"` modelliert werden, da eine weitere Aussage über Klaus gemacht werden muss. Der Autor soll auch nicht als *URI Referenz* modelliert werden, da im kein global im Internet gültiges Attribut zugeordnet werden kann. Für die hat-Autor Beziehung wird das Prädikat `dc:creator` verwendet.

RDF Graph 7.3 Der Autor als Blank Node modelliert

```
ex:DokumentD dc:creator _:1
_:1 ex:Vorname "Klaus"
_:1 ex:Wohnort "Karlsruhe"
```

Modellierung in SiRDF Da es in SiRDF keine *Blank Nodes* gibt, und da Wertknoten auch ohne die explizite Angabe einer *URI Referenz* erzeugt werden können, lassen sich die beiden zu speichernden Informationen, direkt als zwei Aussageknoten speichern. Die Entscheidung zwischen der Modellierung der einzelnen Entitäten als *URI Referenz*, *Blank Node* oder Literal stellt sich nicht. SiRDF kennt nur den Wertknoten, und mit ihm lassen sich alle notwendigen Entitäten modellieren.

Vergleich der Modellierung Die Modellierung in SiRDF ist direkter, und stärker an das mentale Modell angelehnt. Zwei Informationen im mentalen Modell

SiRDF Graph 7.3 Der Autor als Wertknoten modelliert

```
(t:rnd1 ->> (ex:DokumentD) (dc:creator) (t:rnd2 "Klaus"))
(t:rnd3 ->> (t:rnd4 "Klaus") (ex:Wohnort) (t:rnd5 "Karlsruhe))
```

können direkt durch zwei Aussageknoten modelliert werden. Falls es notwendig ist den Autor des Dokuments D von einem anderen Graph aus zu referenzieren, so kann dies auch getan werden. Im Gegensatz dazu, lässt sich der Autor im RDF Graph nicht adressieren, da er durch einen *Blank Node* modelliert wurde.

7.4 Versionierung von Datentypen

Beschreibung Hans hatte Glück. Bei der Euro Umstellung am 1.1.2002 wurde sein Gehalt gleichzeitig verdoppelt. Sein aktuelles Gehalt (2000 Euro) soll gespeichert werden, zusammen mit der Information, das das Gehalt ab dem 1.1.2002 in Euro gezahlt wird.

mentales Modell Die erste Information ist, das Hans ein Gehalt von 2000 Euro hat. Hans ist die Person und das Subjekt der Aussage. Der Wert 2000 ist nur zusammen mit der Währung interessant. Die zweite Information ist, das sich die Währung am 1.1.2002 in Euro geändert hat. Das Subjekt dieser Aussage ist die Währung, das Prädikat ist „geändert-am´´, das Objekt ist das Datum „1.1.2002“.

Modellierung in RDF Hans ist ein Subjekt, muss deswegen also als *URI Referenz* modelliert werden, z.B. `ex:Hans`. „geändert-am“ und „Gehalt“ sind Prädikate und mussen deswegen auch als *URI Referenzen* modelliert werden: `ex:geändert-am` und `ex:Gehalt`.

Das Gehalt muss zusammen mit der Währung gespeichert werden, also als Literal mit Datentyp: `"2000"^^Währung:Euro`. Aber wir wollen eine Aussage über die Währung des Gehalts machen. Das geht nicht, da der Datentyp eines Literals nicht unabhängig vom Literal existiert. Also müssen wir die Aussage über das Literal selbst machen. Das geht auch nicht, da ein Literal kein Subjekt einer Aussage sein darf. Deswegen müssen wir einen *Blank Node* einführen, den wir mit dem Literal assozieren. Über diesen *Blank Node* machen wir dann die Aussage, das er sich am 1.1.2002 geändert hat.

RDF Graph 7.4 Versionierung der Währung in RDF

```
ex:Klaus ex:Gehalt _:1
_:1 rdf:value "2000"^^Währung:Euro
_:1 ex:geändert-am "1.1.2002"
```

Modellierung in SiRDF Falls nicht aufgrund externer Bedingungen *URI Referenz* für Klaus und die Prädikate bekannt sind, können sie einfach durch Wertknoten mit zufälligen *URI Referenzen* modelliert werden. Der Wert des Gehalts wird auch als Wertknoten modelliert. Die Währung des Gehalts wird als Aussage über den Wert des Gehalts festgehalten. Und über die Einheit der Währung lässt sich sehr einfach festhalten, das diese geändert wurde, da sie als Wertknoten gespeichert wurde, und deswegen auch eine *URI Referenz* besitzt.

SiRDF Graph 7.4 Versionierung der Währung in SiRDF

```
(t:rnd1 ->> (t:rnd2 "Klaus")
               (t:rnd3 "Gehalt") (t:rnd4 "2000"))
(t:rnd5 ->> (t:rnd4 "2000")
               (rdf:DataType) (t:rnd6 "Währung:Euro"))
(t:rnd7 ->> (t:rnd6 "Währung:Euro")
               (t:rnd8 "geändert-Am") (t:rnd9 "1.1.2002"))
```

Vergleich der Modellierung Der RDF Graph sieht auf den ersten Blick kleiner, schlanker und einfacher aus. Aber für die Modellierung des Wertes vom Gehalt musste aber der Umweg über einen *Blank Node* gewählt werden. Das eigentlich gewünschte Modell lässt sich mit RDF nicht modellieren.

Der SiRDF Graph ist etwas größer geraten, weil er jeder Entität eine eigene *URI Referenz* zuordnet. Dafür lässt sich dann aber der Wert des Gehalts als Objekt und Subjekt verwenden. Die Währungseinheit des Gehalts lässt sich, weil sie durch einen Wertknoten modelliert wurde, auch adressieren. Deswegen kann sehr präzise genau über die Währungseinheit des Gehalts, die Aussage gemacht werden, das sie sich am 1.1.2002 geändert hat. Und für diese präzise Modellierung mussten fast keine Regeln beachtet werden, sie kommt dem Ideal eines direkten Transfers vom mentalen Modell zum formalen Modell sehr nah.

7.5 Zusammenfassung des Vergleichs

Nachdem wir nun mehrere unterschiedliche Anwendungsfälle betrachtet haben, und zu jedem Anwendungsfall das mentale Modell in ein SiRDF Modell und in ein RDF Modell transferiert haben, lassen sich ein paar klare Aussagen über den Vergleich der Modelle machen.

Adressierbarkeit der Entitäten Da sich alle Entitäten in einem SiRDF Graph direkt über ihre *URI Referenz* adressieren lassen, wird die Modellierung aller Anwendungsfälle, bei denen dieser Aspekt eine große Rolle spielt, sehr stark vereinfacht. Das resultierende Modell ist ein direkteres Abbild des mentalen Modells.

Speziell für die Problematik der Versionierung bietet RDF nicht das notwendige Instrumentarium, da nicht alle Elemente eines RDF Graph adressierbar sind. Nicht adressierbare Entitäten lassen sich auch nicht Versionieren.

Umfang der Repräsentation RDF Graphen haben meistens eine etwas schlankere Repräsentation, dafür enthalten sie meistens sehr viele einzelne Aussagen. SiRDF Graphen speichern zu jeder Entität auch eine *URI Referenz*, deswegen ist jede der wenigen Aussagen in einem SiRDF Graph etwas umfangreicher.

Lesbarkeit des Graphen Alle Fälle, in denen die Adressierung eines Elements in einem RDF Graph modelliert werden müssen, führen dazu, das der RDF Graph unübersichtlicher wird, da RDF die Adressierung mancher Elemente des Graphen nur über Umwege erlaubt. Dies ist im speziellen bei den *Reification* Tripeln zu sehen. Wegen der einheitlichen Behandlung aller Wertknoten und Aussageknoten ist deswegen in den meisten Fällen ein SiRDF Graph besser lesbar.

Abstraktion vom Text Weil alle Elemente eines SiRDF Graph sehr einheitlich modelliert werden können, ist der Transferprozess von einer textuellen Beschreibung eines Sachverhalts zum mentalen Modell, und von dort in das SiRDF Modell sehr viel leichter und direkter. In der Regel lässt sich aus einer wichtigen Information, mit Subjekt, Prädikat und Objekt, auch eine einzelne Aussage erzeugen. Deswegen lassen sich einfache und komplexe Sachverhalte leichter in einem SiRDF Graph modellieren.

8 Zusammenfassung

In dieser Studienarbeit wurde ein neues Datenmodell auf der Basis von RDF beschrieben, welches einige der Schwierigkeiten beim Transfer eines mentalen Modells in ein formales Modell verringert. Dieses neue Datenmodell wird Simple RDF (SiRDF) genannt. Jeder SiRDF Graph kann dann automatisch in einen RDF Graph und wieder zurück konvertiert werden, da SiRDF und RDF semantisch äquivalent sind. Das erlaubt es dem Anwender einen Sachverhalt möglichst direkt im formalen Modell von SiRDF spezifizieren zu können, ohne das die Kompatibilität an die Technologie des Semantic Web verloren geht.

Die Schwierigkeiten beim Modellieren in RDF wurden in Kapitel 2, ab Seite 6, untersucht. Nicht jede RDF Entität kann jede Rolle in einem Tripel einnehmen, so das bei der Modellierung die Art einer Entität entsprechend der Rolle der Entität gewählt werden muss. Literale lassen sich nicht adressieren, genausowenig wie Sprachbezeichner und Datentypen von Literalen. Die Adressierung einer Aussage durch eine andere Aussage gestaltet sich unnötig komplex. Insgesamt kann ein signifkanter Aufwand beim Transfer eines mentalen Modells in ein RDF Modell verzeichnet werden.

Die Begründung dieser Schwierigkeiten durch die abstrakte Syntax und die formale Semantik von RDF wurde in Kapitel 3, ab Seite 12, untersucht. Verschiedene Teile eines RDF Graphen können nicht von anderen Teilen des selben Graphen aus adressiert werden. Dies sind die Literale, Sprachbezeichner, Datentypen und Aussagen selbst. Der Grund liegt in den Regeln der abstrakten Syntax für das Konstruieren eines RDF Graphen, und in der Interpretation dieser Entitäten im Rahmen der formalen Semantik von RDF. *Blank Nodes* sind nur lokal innerhalb eines Graphen adressierbar, da für sie keine global eindeutige *URI Referenz* zur Adressierung verwendet wird. Die Semantik der *Reification* wird in der RDF Spezifikation nicht explizit ausgeführt.

Vereinfacht gesagt, lassen sich die meisten Probleme beim Modellieren in RDF darauf zurückführen, das es Teile des Graphen gibt, für die keine formale Semantik der Adressierung oder keine praktische Möglichkeit der Adressierung spezifiziert wurde. Durch das Einführen einer expliziten Möglichkeit zur Adressierung aller Teile eines Graphen können die identifizierten Probleme verringert werden.

Im SiRDF Modell lassen sich alle Bestandteile eines Graphen adressieren. Ein SiRDF Graph besteht nur aus Aussageknoten und Wertknoten. Ein Aussageknoten besteht aus einem Subjekt, einem Prädikat und einem Objekt. Jede dieser Rollen können von einem anderen Aussageknoten oder einem Wertknoten ein-

genommen werden. Beide Arten von Knoten haben eine *URI Referenz*, über die sie lokal und global eindeutig adressiert werden können. Wertknoten besitzen zusätzlich noch einen Wert. Die ausführliche Spezifikation findet sich in Kapitel 4, ab Seite 20. Dort wird das Datenmodell, die abstrakte Syntax und die formale Semantik beschrieben.

Um den Bezug zwischen SiRDF und RDF herzustellen, wird in Kapitel 5, ab Seite 29, zunächst untersucht, welche formalen Voraussetzungen erfüllt werden müssen, damit ein SiRDF Modell und ein RDF Modell die gleiche mögliche Welt beschreiben, und damit Information in SiRDF und RDF Graphen auf die gleiche Weise bezüglich dieser Welt interpretiert werden können. Wenn beide Modelle auf dem gleichen Vokabular und dem gleichen Universum definiert sind, und die Teilmenge der *URI Referenzen* aus dem Vokabular in beiden Modellen auf die gleiche Weise in das Universum abgebildet werden, und wenn die Prädikate auf dem Universium gleich definiert sind, dann ist eine Interpretation auf beiden Modellen gleich. Auf dieser Grundlage werden die Algorithmen zur Transformation von RDF zu SiRDF und umgekehrt in Form von Pseudo-Code präsentiert.

Anschliessend wird in Kapitel 6, ab Seite 44, die Implementierung der SiRDF API beschrieben. Die SiRDF API erlaubt es dem Anwender die Vorteile von SiRDF für das Modellieren in Java zu verwenden.

Abschliessend wurde der Aufwand der Modellierung in SiRDF und RDF anhand von mehreren kurzen Anwendungsfällen in Kapitel 7, ab Seite 48, verglichen. RDF Graphen sind meistens etwas schlanker in der Repräsentation der einzelnen Tripel, sie bestehen aber meistens aus mehr Tripeln als der äquivalente SiRDF Graph. SiRDF Graphen bestehen dagegen aus weniger Tripeln, in denen aber zu jedem Element auch eine *URI Referenz* als Adresse vermerkt ist. Speziell bei der Adressierung von Aussagen und Literalen, sowie ihren Bestandteilen, führt dies zu einer direkteren und einfachereren Modellierung. Die Abstraktion von einer textuellen Beschreibung eines Sachverhalts zu einem formalen Modell fällt in SiRDF leichter, da es nur einen Wertknoten Typ gibt, und sich die Frage nach der Art der Modellierung einzelner Entitäten nicht stellt. Durch die Adressierbarkeit aller Teile des Graphen lässt sich speziell die Versionierung in einer feineren Granularität realisieren, als dies bei einem RDF Graph möglich ist. Zusammengefasst ergeben sich durch die Modellierung eines Sachverhalts in SiRDF viele echte Vorteile und Vereinfachungen gegebenüber dem Modellieren in RDF.

Literaturverzeichnis

[Alv01] H. Alvestrand. *Tags for the Identification of Languages*. Network Working Group, 2001. Available from: `http://www.isi.edu/in-notes/rfc3066.txt`.

[BCW05] Chris Bizer, Richard Cyganiak, and E. Rowland Watkins. *NG4J - Named Graphs API for Jena*, 2005. Available from: `http://www.wiwiss.fu-berlin.de/suhl/bizer/pub/Bizer-NG4J-ESWC2005.pdf`.

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema, 2002.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. Network Working Group, 1998. Available from: `http://www.isi.edu/in-notes/rfc2396.txt`.

[Con03] The Unicode Consortium. *The Unicode Standard, Version 4.0*. Addison-Wesley, 2003. Available from: `http://www.unicode.org/versions/Unicode4.1.0/`.

[CP05] Jeremy J. Carroll and Addison Phillips. Multilingual RDF and OWL. In *Lecture Notes in Computer Science*, volume 3532, pages 108 – 122, Jan 2005.

[Dec05] Andreas Harth Stefan Decker. *Optimized Index Structures for Querying RDF from the Web*, 2005. Available from: `http://sw.deri.org/2005/02/dexa/yars.pdf`.

[DFZD] Alexandre Delteil, Catherinee Faron-Zucker, and Rose Dieng. Extension of rdf(s) with contextual and definitional knowledge. Available from: `citeseer.ist.psu.edu/453445.html`.

[Dub05] Dublin Core Metadata Initiative. *DCMI Metadata Terms*, 2005. Available from: `http://dublincore.org/documents/dcmi-terms/`.

[Hay04] Patrick Hayes. *RDF Semantics*. World Wide Web Consortium, 10 february 2004 edition, 2004. Available from: `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium, 10 february 2004 edition, 2004. Available from: `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[McB01] Brian McBride. *Jena: Implementing the RDF Model and Syntax Specification*, 2001. Available from: `http://www.hpl.hp.com/personal/bwm/papers/20001221-paper/`.

[MM04] Frank Manola and Eric Miller. *RDF Primer*. World Wide Web Consortium, 10 february 2004 edition, 2004. Available from: `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[Pat] Jeremy Carroll Patrick. Trix : Rdf triples in xml. Available from: `citeseer.ist.psu.edu/699443.html`.

[Sch02] W. Schnotz. *Wissenserwerb mit Texten, Bildern und Diagrammen*. Belz, PVU, Weinheim, third, completely revised edition, 2002.

[URI01] URI Planning Interest Group, W3C/IETF. *URIs, URLs, and URNs: Clarifications and Recommendations 1.0*, 2001. Available from: `http://www.w3.org/TR/uri-clarification/`.

[VEK+05] Max Völkel, Carlos F. Enguix, Sebastian Ryszard Kruk, Anna V. Zhdanova, Robert Stevens, and York Sure. Semversion - versioning rdf and ontologies. KnowledgeWeb Deliverable D2.3.3.v1, Institute AIFB, University of Karlsruhe, JUN 2005. Available from: `http://semversion.ontoware.org/kwebd233a.pdf`.

[YK02] Guizhen Yang and Michael Kifer. On the semantics of anonymous identity and reification. Technical report, Department of Computer Science Stony Brook University, 2002.