Entwicklung einer semantischen Verknüpfungs- und Retrievalengine für IEEE-LOM-konforme Lernobjekte innerhalb des Hypermedia Learning Object Systems HyLOS

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker(FH)

an der
Technischen Fachhochschule Berlin

Fachbereich Informatik und Medien
Studiengang Medieninformatik

1. Betreuer: Prof. Dr. Petra Sauer
2. Betreuer: Prof. Dr. Thomas C. Schmidt

Eingereicht von: Dagmar Lange

Berlin, 1. Januar 2006

Inhaltsverzeichnis

Kapitel 1

Einleitung

Hypermediale Systeme werden immer häufiger in der Lehre eingesetzt und ergänzen das Lernen mit traditionellen Lehrbüchern. Lehrbücher besitzen eine lineare Struktur. Die einzelnen Kapitel bauen meist auf vorangegangenen Kapiteln auf; selbst bei fehlender inhaltlicher Abhängigkeit ist das Springen zwischen den Kapiteln eines Buches unpraktikabel. Hypermediale Systeme weisen keine lineare, sondern eine netzartige Struktur auf. Sie bieten die Möglichkeit, Wissensnetze – bestehend aus atomaren, in sich selbst konsistenten Wissenseinheiten – aufzubauen.

Für den Bereich E-Learning wurde der IEEE-LOM-Standard entwickelt, um solche atomaren, in sich selbst konsistenten Wissenseinheiten – genannt Lernobjekte – zu beschreiben. LOM ermöglicht das Verknüpfen von Lernobjekten mittels benannter Relationen und somit das Aufbauen eines semantischen Wissensnetzes. Ein derart eingewobenes Lernobjekt ist in eine Menge von Lernobjekten definiert eingeordnet; Zusammenhänge zwischen Lernobjekten können aufgrund der Verknüpfungen mittels benannter Relationen erkannt werden. Der Nutzen solch eines semantischen Netzes ist vielfältig. Beispielsweise können Lernende mit Hilfe der benannten Relationen ein Wissensnetz zielgerichtet nach Informationen durchsuchen.

Autoren von Lernobjekten stehen vor der Herausforderung, neue Lernobjekte in vorhandene semantische Wissensnetze einzuweben. Je mehr Lernobjekte vorhanden sind, desto aufwendiger ist es, alle Lernobjekte auf eine mögliche Beziehung zu einem neuerstellten Lernobjekt zu prüfen. Potentiell kann ein neues Lernobjekt zu jedem Lernobjekt innerhalb eines Netzes in Beziehung stehen. Es ist wünschenswert, das Identifizieren und Setzen von Lernobjektverknüpfungen zu automatisieren.

Das Hypermedia Learning Object System HyLOS, welches sowohl Lernobjekte präsentiert als auch ein Autorenwerkzeug zum Editieren von Lernobjektinhalten und -metadaten bietet, bildet den LOM-Standard konsequent ab. Im Rahmen der vorliegenden Arbeit wird eine semantische Verknüpfungs- und Retrievalengine für HyLOS entwickelt.

Die vorliegende Arbeit untersucht, welche Zusammenhänge zwischen Lernobjekten mittels benannter Relationen abgebildet werden sollen. Basierend auf den identifizierten Relationen werden Strategien entwickelt, diese automatisiert zu ermitteln. Infolge der Diskussion möglicher

Einsatzszenarien, notwendiger Funktionen und Anforderungen wird eine API entwickelt, welche anschließend implementiert wird.

Die Arbeit gliedert sich in 7 Abschnitte. Das zweite Kapitel erarbeitet die Grundlagen dieser Arbeit. Im dritten Kapitel wird die Problematik der Lernobjektverknüpfungen behandelt. Das vierte und fünfte Kapitel stellen Konzeption und Implementierung der Verknüpfungsengine vor. Im sechsten Kapitel wird ein Test ausgewertet. Siebentes und achtes Kapitel schließen diese Arbeit mit einer Zusammenfassung und einem Ausblick ab.

Kapitel 2

Grundlagen

2.1 Lerninhalte handhaben

Content Management Systeme für den Bereich E-Learning haben neben der Präsentation von Lerninhalten die Aufgabe, Autoren beim Erstellen und Verwalten von Lerninhalten zu unterstützen. Lerninhalte werden in kleine sinnvolle Lerneinheiten, so genannte Lernobjekte, unterteilt. Der Metadatenstandard LOM wurde entwickelt, um solche Lernobjekte zu beschreiben. Das Editieren dieser Metadaten wird durch geeignete Werkzeuge ermöglicht.

Dieser Abschnitt stellt sowohl den LOM-Standard als auch das Hypermedia Learning Object System HyLOS vor. HyLOS bildet LOM vollständig ab.

2.1.1 LOM – Ein Metadatenstandard zum Beschreiben von Lernobjekten

Learning Object Metadata (LOM) [LOM02] ist ein Metadatenstandard. Er wurde von dem Learning Technology Standards Committee, welches am Institute of Electrical and Electronics Engineers (IEEE) ansässig ist, entwickelt. Einfluss auf die Entwicklung von LOM hatten sowohl die Projekte ARIADNE[1] und IMS[2] als auch der Dublin-Core-Metadatenstandard[3].

Metadaten sind Daten, die Informationen über andere Daten enthalten. Oft werden Metadaten in Attribut-Wert-Paaren erfasst. Damit Metadaten von Nutzen sind, müssen sie standardisiert, das heißt, kontrollierte Vokabulare entwickelt werden. Ein kontrolliertes Vokabular bezeichnet eine Sammlung von Wörtern, welche eindeutig Konzepten zugeordnet sind. Diese Sammlung enthält keine Homonyme und in vielen Fällen ebenfalls keine Synonyme. Jedes Wort der Sammlung ist somit genau einem Konzept zugeordnet; jedes Konzept besitzt meistens nur eine präferierte Benennung. Neben kontrollierten Attributen definieren manche Standards auch kontrollierte Werte. Durch den Einsatz von Metadatenstandards werden Dokumente einheitlich

[1] http://www.ariadne-eu.org/
[2] http://www.imsproject.org/
[3] http://www.dublincore.org/

beschrieben. Die Verarbeitung – zum Beispiel das Auffinden von Dokumenten – ist einfacher als das Verarbeiten willkürlich beschriebener Dokumente.

Der LOM-Standard ist motiviert durch die steigende Anzahl von Lernobjekten, welche jedoch durch fehlende Beschreibungen schwer auffindbar, schwer automatisiert verarbeitbar und somit schwer nutzbar sind. Ein Lernobjekt ist eine in sich selbst konsistente, atomare Wissenseinheit, die sowohl den Inhalt als auch die Metadaten kapselt. Der Inhalt kann selbst aus Lernobjekten bestehen und somit eine selbstähnliche hierarchische Lernobjektstruktur aufbauen. LOM ist mit dem Ziel entwickelt worden, die Suche nach Lernobjekten, deren Bewertung, Erwerb und Gebrauch für Lernende, Lehrende und Softwareprogramme zu erleichtern.

LOM definiert die Struktur der Beschreibung eines Lernobjektes – im Weiteren Metadateninstanz genannt. Metadateninstanzen beschreiben die relevanten Eigenschaften eines Lernobjektes. Diese sind in neun Kategorien gruppiert. Pro Kategorie werden verwandte Metadatenelemente zusammengefasst. Die neun Kategorien sind:

- *General* enthält allgemeine Informationen, die das Lernobjekt als Ganzes beschreiben.
- *Lifecycle* gruppiert Angaben zur Historie, zum aktuellen Stand und zu Mitwirkenden an der Erstellung des Lernobjektes.
- *Meta-Metadata* enthält Informationen über die Metadateninstanz selbst anstatt Informationen über das Lernobjekt.
- *Technical* gruppiert technische Merkmale und technische Anforderungen des Lernobjektes.
- *Educational* enthält pädagogische Angaben zum Lernobjekt.
- *Rights* enthält Angaben zu Urheber- und Verwertungsrechten.
- *Relation* gruppiert Angaben über die Beziehungen des Lernobjektes zu anderen Lernobjekten.
- *Annotation* enthält Kommentare zum pädagogischen Nutzen des Lernobjektes und Informationen über den Autor und das Erstellungsdatum der Kommentare.
- *Classification* beschreibt das Lernobjekt in Beziehung zu einem Klassifikationssystem.

Die Daten innerhalb einer Kategorie sind hierarchisch strukturiert. Kategorien enthalten Datenelemente, die aus zusammengesetzten, hierarchisch organisierten Datenelementen bestehen oder konkrete Daten enthalten. Nur die Blätter der Hierarchie dürfen Daten aufnehmen.

Für einige der Datenelemente definiert der LOM-Standard kontrollierte Werte. Dabei handelt es sich um eine empfohlene Liste angebrachter Werte; sie darf erweitert werden. Um beispielsweise die semantische Bedeutung einer Beziehung zweier Lernobjekte innerhalb der Kategorie

Relation zu unterscheiden, steht ein Vokabular von zwölf vordefinierten Relationen zur Verfügung. Die aus dem Dublin-Core-Metadatenstandard übernommenen Relationen sind in sechs paarweise inversen Relationen geordnet.

isPartOf	isVersionOf	isFormatOf	references	isBasedOn	requires
hasPart	hasVersion	hasFormat	isReferencedBy	isBasisFor	isRequiredBy

Tabelle 2.1: vordefinierte LOM-Relationen

2.1.2 Einführung in HyLOS

Das Hypermedia Learning Object System HyLOS [HyL05] [EHLS05] [EHS05] ist ein lernobjektverarbeitendes Content Management System. Lernobjekte innerhalb des HyLOS-Systems kapseln sowohl Inhalt als auch Metadaten, welche LOM konform sind. Lernobjekte können Unterlernobjekte enthalten. HyLOS bietet neben einer Präsentationsschicht mit variablen Zugangsstrukturen eine Autorenumgebung zum Editieren von Inhalt und Metadaten. HyLOS basiert auf dem Media Information Repositoy MIR [FKRS01] [MIR05], einem anwendungsneutralen, hypermedialen Storage- und Laufzeitsystem zur Speicherung und Verarbeitung medialer Daten.

Eine Schwierigkeit des LOM-Standards ist die Reichhaltigkeit seiner Daten. LOM umfasst mehr als 60 Datenelemente, die teilweise durch Zuweisung verschiedener Werte mehrfach genutzt werden können. Autoren verzweifeln schnell, sollen sie den gesamten LOM-Datensatz per Hand eingeben. Die HyLOS-Autorenumgebung verfolgt den Ansatz, die Menge der notwendig manuell einzugebenden Daten zu reduzieren, damit Autoren sich auf die Inhaltseingabe konzentrieren können. Notwendig einzugebende Metadaten umfassen den pädagogischen Nutzen und allgemeine Informationen. Die Kategorien *General* und *Educational* des LOM-Standards enthalten im Wesentlichen diese Daten. Neben notwendig manuell einzugebenden Daten unterscheidet HyLOS automatisiert bestimmbare und optional manuell einzugebende Metadaten. Beispielsweise kann die Autorenumgebung den MIME-Type oder den Autor eines Lernobjektes automatisch ermitteln. Optional manuell einzugebende Daten sind zur Verarbeitung - zum Beispiel zur Darstellung - nicht zwingend erforderlich. Der gesamte Metadatensatz des LOM-Standards kann trotz der genannten Unterscheidungen vollständig manuell editiert werden.

Die Präsentationsschicht bietet drei verschiedene Sichten auf den Inhalt. Neben der hierarchischen Sicht, die sich aus der Lernobjektstruktur ableitet, existieren eine konstruktivistische und eine instruktivistische Sicht. Durch die Möglichkeit, Lernobjekten Unterlernobjekte zuzuweisen, lassen sich Lernobjekte strukturieren. Die hierarchische Sicht ermöglicht das Navigieren durch den so entstandenen Strukturbaum. Die konstruktivistische Sicht visualisiert Lernobjektbeziehungen, die in der Kategorie *Relation* des LOM-Standards abgelegt sind. Ausgehend von einem selektierten Lernobjekt kann durch den gerichteten Graphen der benannten Beziehungen navigiert werden. Zusätzlich zu den zwei genannten Sichten können Lehrende Instruktionspfade anlegen. Ein Instruktionspfad besteht aus einer beliebigen Menge von hierarchisierten Contai-

nerobjekten gliedernden Charakters - Lerneinheit, Kapitel, Unterkapitel, Seite -, die auf vorhandene Lernobjekte bzw. Lernobjektstrukturen verweisen. Der Lernende kann frei zwischen den verschiedenen Sichten wählen sowie jederzeit die Sicht wechseln. Abbildung 2.1 veranschaulicht diesen Sachverhalt.

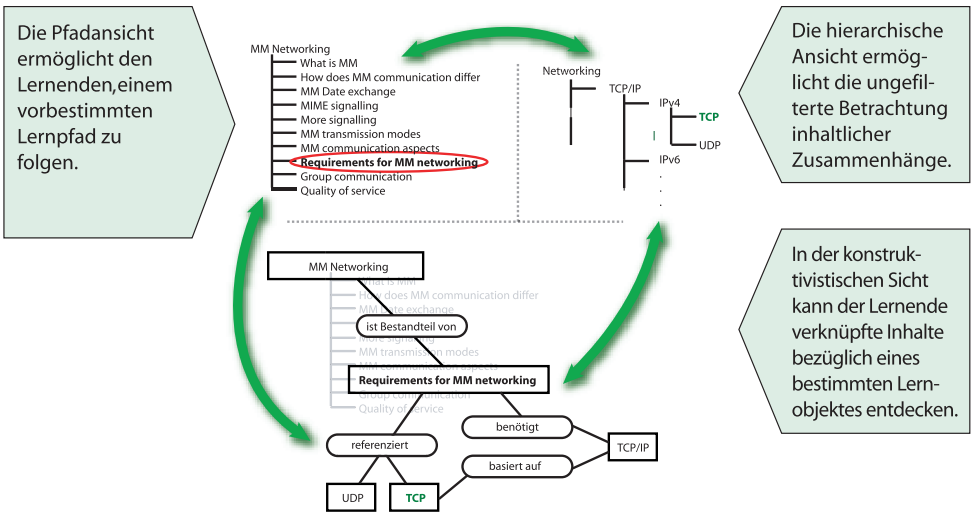

Abbildung 2.1: Die Sichten innerhalb des HyLOS-Systems

Eine vierte Navigationsmöglichkeit durch die Lerninhalte bilden Hyperreferenzen. HyLOS implementiert ein semantisches Verweismodell. In diesem Modell sind Anker nicht im eigentlichen Inhalt enthalten, sondern werden als separate Objekte gespeichert, die die Inhaltsobjekte oder Stellen innerhalb der Inhaltsobjekte referenzieren. Verweise zwischen den Ankern sind ebenfalls separat gespeicherte Objekte, ihnen können Kontexte zugewiesen werden. Da Verweise dynamisch ein- und ausgeblendet werden können, haben sowohl Lernende als auch Autoren die Möglichkeit, auf denselben Inhalten mit wechselnden Verweiskontexten zu interagieren [ES03].

Neben der manuellen Eingabe von Lernobjekten, werden Vorlesungen aufgezeichnet, um Lernobjekte automatisch zu erzeugen. Besonderes Augenmerk liegt hierbei auf den eingesetzten Lehrmaterialien und der gesprochenen Sprache. Die aufgezeichnete Vorlesung wird nachfolgend anhand geeigneter Kriterien wie zum Beispiel einem Folienwechsel in einzelne kleine Lernobjekte unterteilt und durch Aggregation der zusammengehörenden Teile in eine Lernobjektstruktur überführt und im HyLOS-System abgelegt. Die so entstandenen Lernobjekte besitzen keine für die semantische Auswertung notwendigen Metadaten. Deshalb werden die Lernobjekte mit Schlagwörtern angereichert und anschließend automatisch klassifiziert. Zur Klassifizierung wer-

den innerhalb des HyLOS-Systems weit verbreitete Taxonomien wie die Dewey Decimal Classification (DDC) [DDC05] und das ACM Computing Classification System (CCS) [ACM98] verwandt.

2.2 Semantik von Informationen im World Wide Web

Ontologien haben aufgrund der Vision des Semantic Webs – der Erweiterung des heutigen World Wide Webs, in der Informationen eine wohldefinierte Bedeutung besitzen [BLHL01] – an Bedeutung gewonnen. Der Begriff Ontologie wird in diesem Abschnitt erläutert. Anschließend werden zwei Ontologiesprachen – RDF und OWL –, die für den Einsatz im World Wide Web entwickelt worden sind, vorgestellt.

2.2.1 Ontologien

Der Begriff Ontologie entstammt ursprünglich der Philosophie. Als Ontologie wird hier die Lehre vom Sein und vom Seienden, die die formalen und materialen Prinzipien der Wirklichkeit begrifflich zu bestimmen sucht, bezeichnet. Zum einen befasst sich die Ontologie mit den obersten Strukturen und Gesetzlichkeiten des Seins. Zum anderen handelt die Ontologie von der inhaltlichen Gliederung des Seienden [phi04].

In der Informatik werden Ontologien zur Wissensrepräsentation eingesetzt. Es existieren viele Definitionen für den Begriff Ontologie; als Ontologien werden unterschiedliche – relativ einfache bis sehr komplexe – Strukturen bezeichnet, z.B. Metadatenstandards, Taxonomien oder logische Systeme [OWL04c]. Die Frage, ob die aufgeführten Strukturen Ontologien sind, wird im Anschluss an die Begriffsklärung beantwortet. Laut Gruber [Gru93] ist eine Ontologie eine explizite Spezifikation einer Konzeptualisierung. Unter Konzeptualisierung versteht Gruber eine abstrakte, vereinfachte Sicht auf die Welt, die für einen bestimmten Zweck abgebildet werden soll.

Ontologien beschreiben einen Bereich (zum Beispiel Medizin), namentlich die Konzepte des Bereiches, deren Eigenschaften und die Beziehungen zwischen den Konzepten. Sehr komplexe Ontologien enthalten zusätzlich Regeln. Daconta, Obrst und Smith [DOS03] unterscheiden drei Repräsentationslevels:

- Knowledge Representation Level
- Ontology Concept Level
- Ontology Instance Level

Das *Knowledge Representation Level* definiert die Sprachelemente, die zur Beschreibung genutzt werden können. Dazu gehören beispielsweise Sprachelemente für die Idee der Klasse oder

die Idee der Eigenschaft. Daneben können auch Sprachelemente existieren, mit denen sich Restriktionen für die Verwendung von Eigenschaften oder Beziehungen zwischen Klassen definieren lassen. Die Ausdruckskraft von Ontologiesprachen variiert stark. Die Sprachelemente der jeweiligen Ontologiesprache werden auf dem *Ontology Concept Level* genutzt, um die Konzepte eines Bereiches zu modellieren. Konkrete Klassen, die konkrete Eigenschaften besitzen, werden hier definiert. Soll der Bereich *Comics* in einer Ontologie abgebildet werden, so sind Klassen wie *Comicfigur*, *Zeichner* und die Eigenschaft *hatZeichner* für die Klasse *Comicfigur* denkbar. Das *Ontology Instance Level* enthält die eigentlichen Daten. Instanzen der auf dem *Ontology Concept Level* definierten Klassen werden erzeugt, deren Eigenschaften mit Werten gefüllt. *Calvin* und *Hobbes* sind Instanzen der Klasse *Comicfigur*. Die Eigenschaft *hatZeichner* besitzt für beide Instanzen den Wert *Bill Watterson*. *Bill Watterson* ist Instanz der Klasse *Zeichner*. Abbildung 2.2 visualisiert diese Beispielontologie.

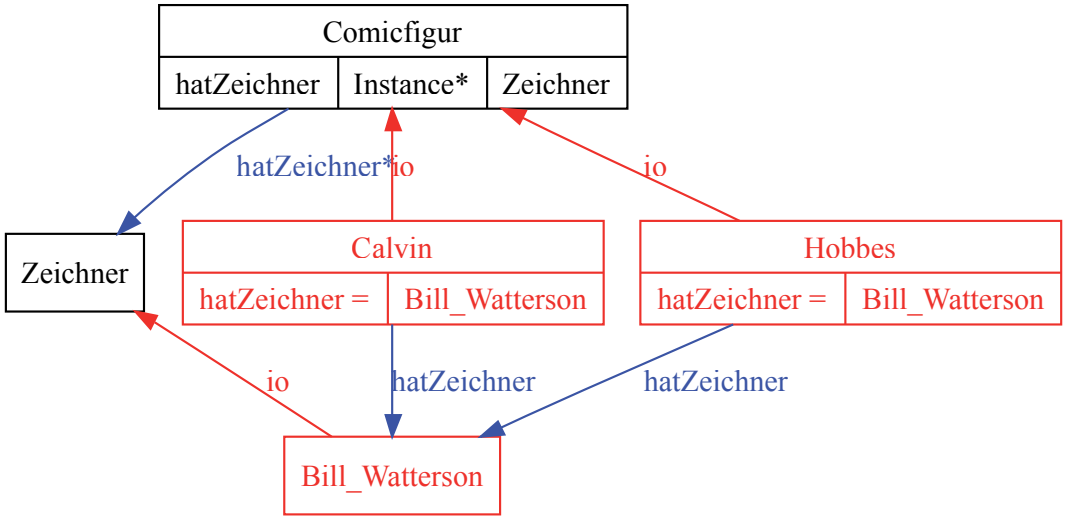

Abbildung 2.2: Graphische Darstellung einer Ontologie

Ontologien werden in logikbasierten Sprachen kodiert. Diese Sprachen sind unzweideutig; sie weisen ein Vokabular und eine wohldefinierte Syntax auf. Die Semantik einer Aussage kodiert in solch einer Sprache erschließt sich nicht durch subjektive Intuition oder Interpretation; stattdessen gibt es nur eine korrekte, definierte Bedeutung. Außerdem lassen sich in logikbasierten Sprachen Axiome und Regeln kodieren. Dadurch werden Ontologien für Maschinen nutzbar und können von Maschinen semantisch interpretiert werden. Unter semantischer Interpretation wird die Abbildung von Daten auf Konzepten innerhalb eines Modells verstanden. Maschinen können somit die Konsistenz von Ontologien, also die Konformität von Daten und Modell, prüfen. Werden zusätzlich zu dem Modell und den Daten Regeln definiert, können Maschinen automatisch schlussfolgern. Das heißt, sie erarbeiten neue Informationen durch das Anwenden von Regeln auf den gegebenen Daten.

Metadatenstandards, Taxonomien und logische Systeme – Strukturen unterschiedlicher Komplexität – werden oft als Ontologien bezeichnet. Ist das jedoch korrekt? In Abschnitt 2.1.1 auf Seite 3 wird erläutert, was ein Metadatenstandard ist. Metadatenstandards definieren kontrollierte Vokabulare, um Daten – zum Beispiel ein Textdokument – mittels Attribut-Wert-Paaren zu beschreiben. Aus dem Blickwinkel einer Ontologie ist das Textdokument ein Individuum, eine Instanz einer Klasse; die das Textdokument beschreibenden Attribut-Wert-Paare sind Eigenschaften oder Beziehungen, denen konkrete Werte zugewiesen sind. Das Beispiel zeigt, dass ein Metadatenstandard keine Ontologie ist. Ontologietypische Konzepte wie Klasse, Eigenschaft oder Individuum kennt ein Metadatenstandard nicht; er definiert im Gegensatz dazu ein Vokabular, um Individuen zu beschreiben.

Eine Taxonomie ist ein hierarchisches Klassifikationssystem, das oft durch einen Baum visualisiert wird. Ein Taxonomieknoten steht für ein Konzept. Normalerweise stehen die Knoten eines Taxonomiezweiges in einer Subklassen-, Teil- oder Spezialisierungsbeziehungen. Es sind jedoch auch Querverweise zwischen Knoten unterschiedlicher Taxonomiezweige möglich. Eine Taxonomie ist eine spezielle Ontologie. Ein Taxonomieknoten ist ein Individuum einer Klasse, die für das Konzept *Konzept* steht. Eigenschaften innerhalb einer Taxonomie können *istSpeziellerAls* oder *istAllgemeinerAls* sein. Taxonomien lassen sich mit Hilfe von SKOS[4] (Simple Knowledge Organisation System), einem Vokabular zum Beschreiben von Wissensorganisationssystemen, abbilden.

Logik beschäftigt sich im Allgemeinen mit dem Ziehen von Schlussfolgerungen aus gegebenen Aussagen, der Widerspruchsfreiheit von Aussagen und dem Ermitteln des Wahrheitsgehaltes von Aussagen. Prädikatenlogik eignet sich zum Abbilden von Ontologien, da sie zwischen Individuen und Eigenschaften, so genannten Prädikaten, unterscheidet. Zusätzlich können Quantoren – Allquantor („alle“) und Existenzquantor („mindestens einer“) – bei der Formulierung von Aussagen eingesetzt werden. Die Beispielontologie für den Bereich Comics kann folgendermaßen symbolisiert werden:

$C(x)$	x ist eine Comicfigur
$Z(x)$	x ist ein Zeichner
$HZ(x, y)$	x hat Zeichner y
$(\forall(x))(HZ(y, x) \wedge C(y) \rightarrow Z(x))$	Wenn eine Comicfigur y x als Zeichner hat, dann ist x ein Zeichner.
$Z(w)$	Bill Watterson ist ein Zeichner
$C(c) \wedge HZ(c, w)$	Calvin ist eine Comicfigur und hat den Zeichner Bill Watterson
$C(h) \wedge HZ(h, w)$	Hobbes ist eine Comicfigur und hat den Zeichner Bill Watterson

Das Beispiel zeigt, dass eine einfache Ontologie durch Prädikatenlogik darstellbar ist. Daraus jedoch zu schließen, ein logisches System ist eine Ontologien, ist falsch. Aussagenlogik unter-

[4] http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/

scheidet im Gegensatz zur Prädikatenlogik nicht zwischen Individuen und Eigenschaften; die Beispielontologie kann nicht durch Aussagenlogik abgebildet werden.

Durch den Einsatz von Ontologien lassen sich „intelligente“ Softwareprogramme entwickeln. Ein Beispiel dafür sind semantische Suchmaschinen. Heutige Suchmaschinen sind textbasiert; ist der Suchbegriff innerhalb des Textes oder der Schlüsselwörter des Dokumentes enthalten, so ist ein Suchergebnis gefunden. Im Gegensatz dazu „verstehen“ semantische Suchmaschinen sowohl den Suchbegriff als auch die Dokumente, die als potentielle Suchergebnisse in Frage kommen. Die Entwicklung von semantischen Suchmaschinen für das gesamte World Wide Web ist schwierig; die Entwicklung solch einer „intelligenten“ Suchmaschine für einen abgegrenzten Bereich ist eher denkbar. Der Grund dafür ist, dass eine Ontologie entwickelt werden muss, die das Suchgebiet umfassend abbildet. Auf die gesamte Welt bezogen ist das theoretisch machbar, jedoch praktisch zu umfangreich. Ein weiteres Problem stellt die Erkennung des Kontextes für eine Suchanfrage – vor allem bei Homonymen – dar. Der Suchbegriff *Heide* ist nicht eindeutig, solange der Kontext einer Suchanfrage nicht bekannt ist. *Heide* kann sowohl *Nichtchrist* als auch *unbebautes Land* bedeuten. Wird nach Dokumenten des Bereiches Landschaftspflege gesucht, ist klar, dass *unbebautes Land* gemeint ist. Die Dokumente, die durchsucht werden, müssen anhand der Ontologie, die den Bereich abbildet, beschrieben werden. Der Suchbegriff wird vor dem eigentlichen Suchvorgang verarbeitet; die Ontologie kann beispielsweise Synonyme oder verwandte Konzepte definieren, die für die Formulierung einer Suchanfrage innerhalb des Systems genutzt werden.

2.2.2 Resource Description Framework RDF

Das Resource Description Framework (RDF) [RDF04a] [RDF04c] ist eine Sprache, die vom World Wide Web Consortium entwickelt wurde, um Informationen über Ressourcen im World Wide Web für Maschinen verarbeitbar zu präsentieren. Der Begriff Ressource wird sehr allgemein aufgefasst; Ressource bezeichnet zum einen Webseiten, zum anderen jedoch ebenso Dinge, die im World Wide Web identifiziert werden können, auch wenn auf sie nicht direkt zugegriffen werden kann, wie zum Beispiel Waren eines Onlineshops. Zur Kennzeichnung von Ressourcen werden Uniform Resource Identifiers (URI) verwandt. Ein Uniform Resource Identifier ist eine geschlossene Zeichenkette zum Identifizieren von abstrakten oder physikalischen Ressourcen [URI05].

RDF basiert auf der Idee, dass über jede Ressource Aussagen in Form eines einfachen Satzes – Subjekt, Prädikat, Objekt – getroffen werden können. Die abstrakte Syntax von RDF spiegelt diese Idee wider. Eine Aussage über eine Ressource wird in RDF als Tripel dargestellt. Ein Tripel besteht aus drei Komponenten:

- *Subjekt* – Ressource
- *Prädikat* – Property der Ressource

- *Objekt* – Wert der Property

In RDF repräsentieren Properties entweder Eigenschaften einer Ressource oder Beziehungen zwischen zwei Ressourcen. Wegen dieser Doppelbedeutung wird nachfolgend das englische Wort Property oder dessen Pluralform Properties verwandt.

RDF modelliert Aussagen über Ressourcen als Knoten und Kanten eines Graphen. Das Subjekt und das Objekt einer Aussage bilden die Knoten des Graphen, das Prädikat ist eine gerichtete Kante vom Subjekt zum Objekt. Eine Menge solcher Tripel wird als RDF-Graph bezeichnet. Abbildung 2.3 veranschaulicht diesen Sachverhalt.

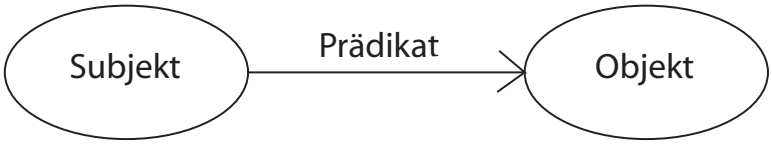

Abbildung 2.3: RDF-Graph

Tabelle 2.2 und Abbildung 2.4[5] zeigen, wie in RDF folgende Aussage modelliert wird: *"Die Comicfigur Calvin hat einen Zeichner namens Bill Watterson."*

Subjekt	Calvin
Prädikat	hatZeichner
Objekt	Bill Watterson

Tabelle 2.2: Beispiel einer RDF-Aussage

Abbildung 2.4: Beispiel eines RDF-Graphen

Das graphenbasierte Datenmodell von RDF ist unabhängig von der Syntax seiner Serialisierung. Die gebräuchlichsten Serialisierungssyntaxen sind RDF/XML[6], N-Triple[7]und N3[8]. Das aufgeführte Beispiel notiert in RDF/XML sieht folgendermaßen aus:

[5] In der graphischen Darstellung werden Knoten, die URIs enthalten, als Ellipsen und Knoten, die Literale enthalten, als Rechtecke symbolisiert.

[6] http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[7] http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/#ntriples

[8] http://www.w3.org/DesignIssues/Notation3.html

```
<rdf:RDF
   xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
   xmlns:comic="http://www.beispiel.de/comic/begriffe#"

   <rdf:Description
      rdf:about="http://www.beispiel.de/comic/figuren/calvin">
      <comic:hatZeichner>Bill Watterson</comic:hatZeichner>
   </rdf:Description>
</rdf:RDF>
```

Subjekte können entweder durch URIs oder leere Knoten, Prädikate nur durch URIs, Objekte entweder durch URIs oder Literale oder leere Knoten dargestellt werden. Deshalb wird in der RDF-XML-Darstellung des Beispiels die URI *http://www.beispiel.de/comic/figuren/calvin* anstatt *Calvin* und die URI *http://www.beispiel.de/comic/begriffe#hatZeichner* anstatt *hatZeichner* verwandt. [RDF04a] und [RDF04c] enthalten weiterführende Informationen dazu.

RDF spezifiziert eine abstrakte Syntax, um Ressourcen im World Wide Web zu beschreiben, jedoch nicht das Vokabular, welches zur Beschreibung eingesetzt wird. Das Vokabular wird in RDF Schema (RDFS) [RDF04b] definiert. In Abschnitt 2.2.1 wurden die drei Repräsentationslevels einer Ontologie vorgestellt. RDFS ist auf dem *Knowledge Representation Level* und dem *Ontology Concept Level* angesiedelt. RDFS definiert Sprachelemente, um Klassen und Properties, die innerhalb der RDF-Aussagen eingesetzt werden, zu beschreiben. Zusätzlich gibt RDFS an, wie diese Klassen und Properties zusammen genutzt werden. Die Möglichkeiten von RDFS gehen jedoch über die Definition von Klassen und Properties hinaus; beispielsweise besitzt RDFS zusätzlich ein Vokabular zur Definition von Containern oder Aussagen über RDF-Aussagen. Im Gegensatz zu RDFS trifft RDF Aussagen über Ressourcen, Instanzen der in RDFS definierten Klassen. RDF-Aussagen sind Angaben auf dem *Ontology Instance Level*. Das nachfolgende Beispiel zeigt die Funktionsweise von RDFS.

```
<rdf:RDF
   xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3c.org/2000/01/rdf-schema#"
   xml:base="http://www.beispiel.de/comic/begriffe#">

   <rdfs:Class rdf:ID="Mensch"/>

   <rdfs:Class rdf:ID="Zeichner">
      <rdfs:subClassOf rdf:resource="#Mensch"/>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Comicfigur"/>
```

```
   <rdfs:Property rdf:ID="hatZeichner">
      <rdfs:domain rdf:resource="#Comicfigur"/>
      <rdfs:range rdf:resource="#Zeichner"/>
   </rdfs:Property>
</rdf:RDF>
```

In diesem Beispiel werden zuerst die drei Klassen *Mensch*, *Zeichner* und *Comicfigur* definiert. Die Klasse *Zeichner* ist Unterklasse der Klasse *Mensch*. Die Property *hatZeichner* legt fest, dass sie auf Klassen des Typs *Comicfigur* angewandt werden darf und dass sie Individuen der Klasse *Zeichner* als Werte zulässt.

Das oben aufgeführte Beipiel zeigt eine Eigenschaft, die RDFS von anderen Typsystemen unterscheidet. Bei RDFS stehen die Properties einer Ressource im Zentrum und nicht deren Klasse. Das heißt, nicht Klassen, die Properties besitzen, werden definiert, sondern Properties definieren die Klassen, auf die sie angewandt werden dürfen.

Als eine weitere Eigenschaft sind die mächtigen Metamodellierungsmöglichkeiten von RDFS hervorzuheben. RDFS verlangt keine strikte Trennung von Klassen, Properties, Individuen und Datentypen; so kann beispielsweise eine Klasse Instanz von sich selbst sein. Die vorhergehenden Beispiele dieses Abschnittes reizen diese Möglichkeiten im Gegensatz zu dem folgenden nicht aus. Im folgenden Beispiel wird *Bill Watterson* als Wert der Property *hatZeichner* der Klasse *Comicfigur* – somit auch allen ihrer Instanzen – zugewiesen.

```
<rdf:RDF
   xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3c.org/2000/01/rdf-schema#"
   xmlns:comic="http://www.beispiel.de/comic/begriffe#"
   xml:base="http://www.beispiel.de/comic/begriffe#">

   <rdfs:Class rdf:ID="Comicfigur"/>

   <rdf:Description rdf:ID="Comicfigur">
      <comic:hatZeichner>Bill Watterson</comic:hatZeichner>
   </rdf:Description>
</rdf:RDF>
```

Der Grund für die Entwicklung von Ontologiesprachen für das World Wide Web ist in erster Linie der Wunsch, die in Dokumenten enthaltenen Informationen für Maschinen interpretierbar kodieren zu können. RDF ist ein Datenmodell zum Darstellen von Informationen als Aussagen über Ressourcen. RDFS definiert Klassen und Properties dieser Ressourcen. Ontologiesprachen sollen neben dem Kodieren von Informationen auch das Herleiten neuer Informationen durch

Maschinen ermöglichen. Die Semantik von RDF/RDFS erlaubt das Ziehen von Schlussfolgerungen, ist jedoch hauptsächlich beschränkt auf das Organisieren von Klassen und Properties in Subklassen- und Subpropertieshierarchien und dem Beschränken von Domäne und Wertebereich einer Property. RDF/RDFS kann folgende Beispiele nicht abbilden:

- Kardinalitätsbeschränkungen auf Properties (Eine Person hat genau einen biologischen Vater.)
- Spezifikation von transitiven, symmetrischen und inversen Properties
- Spezifikation, dass eine Property hinreichend für die Zugehörigkeit zu einer bestimmten Klasse ist
- Spezifikation, dass zwei Klassen identisch sind
- Spezifikation, dass zwei Objekte identisch sind
- Spezifikation von Beschränkungen auf dem Wertebereich oder der Kardinalität einer Property in Abhängigkeit von der Klasse, auf die die Property angewandt wird (Eine Fussballmannschaft hat genau elf Spieler, eine Volleyballmannschaft jedoch nur sechs.)
- Beschreibung neuer Klassen durch Kombination (Schnittmenge, Vereinigungsmenge, Differenzmenge) anderer Klassen
- Spezifikation disjunkter Klassen

Aufgrund dieser fehlenden Beschreibungsmöglichkeiten wurde OWL entwickelt.

2.2.3 Web Ontology Language OWL

OWL [OWL04a] [OWL04b] ist eine Ontologiesprache für das Semantic Web, die aus der Ontologiesprache DAML+OIL[9] abgeleitet wurde. Aufbauend auf RDF/RDFS besitzt OWL ein reichhaltigeres Vokabular zur Beschreibung von Klassen und Properties.

Die Entwicklung reichhaltigerer Ontologiesprachen führt zu einem Kompromiss zwischen Ausdruckskraft und Schlussfolgerungseffizienz. OWL trägt dem Rechnung, indem drei Untersprachen - OWL Full, OWL DL und OWL Lite - spezifiziert wurden, die sich durch ihre abnehmende Ausdruckskraft unterscheiden.

- *OWL Full* verfügt über die maximale Ausdruckskraft. Alle Sprachelemente von OWL und RDF/RDFS werden uneingeschränkt unterstützt. Wie RDFS verlangt OWL Full keine Trennung von Klassen, Properties, Individuen und Datentypen. Vollständiges Schlussfolgern kann für OWL Full nicht garantiert werden.

[9] http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218

- *OWL DL* unterstützt alle Sprachelemente von OWL, diese jedoch nur unter Einschränkungen. OWL DL verlangt die Trennung von Klassen, Properties, Individuen und Datentypen. Das ist der Grund, weshalb nur die Sprachelemente von RDF/RDFS verwandt werden dürfen, durch die sich Datentypen handhaben, Subklassen- beziehungsweise Subpropertieshierarchien aufbauen, Domäne beziehungsweise Wertebereich von Properties beschränken lassen. So ist zum Beispiel das Element *rdf:Property* nicht erlaubt. Da OWL DL strikt zwischen Individuen und Datentypen unterscheidet, sind Properties entweder vom Typ *owl:ObjectProperty* oder *owl:DatatypeProperty*. *owl:ObjectProperty* setzt Individuen in Beziehung zueinander, *owl:DatatypeProperty* weist einem Individuum eine Instanz eines Datentyps zu. Die Buchstaben DL stehen für Description Logic (Beschreibungslogik). Die Beschreibungslogik ist eine Untermenge der Prädikatenlogik erster Ordnung. Beschreibungslogik ist vollständig[10] und im Gegensatz zur Prädikatenlogik erster Ordnung in endlicher Zeit entscheidbar[11].

- *OWL Lite* ist eine Einschränkung von OWL DL. OWL Lite unterstützt nur eine Untermenge der Sprachelemente von OWL und dieselben RDF-RDFS-Sprachelemente wie OWL DL; die Verwendung einiger erlaubter Sprachelemente wird zusätzlich beschränkt. Die Idee hinter der schwachen Ausdruckskraft von OWL Lite ist, eine minimal brauchbare Untermenge der Sprachkonstrukte von OWL zur Verfügung zu stellen. Dadurch wird der Einstieg in das Arbeiten mit OWL erleichtert und die Entwicklung von Schlussfolgerungsmaschinenen vereinfacht. OWL Lite ist trotz schwacher Ausdruckskraft geeignet, Klassifikationshierarchien wie Taxonomien abzubilden. OWL Lite ist vollständig und in endlicher Zeit entscheidbar.

Aufgrund der unterschiedlich mächtigen Ausdruckskraft einer jeden der drei OWL-Sprachen sind folgende Beziehungen immer wahr:

- Jede gültige RDF-RDFS-Ontologie ist eine gültige OWL-Full-Ontologie.
- Jede gültige RDF-RDFS-Schlussfolgerung ist eine gültige OWL-Full-Schlussfolgerung.
- Jede gültige OWL-Lite-Ontologie ist eine gültige OWL-DL-Ontologie.
- Jede gültige OWL-Lite-Schlussfolgerung ist eine gültige OWL-DL-Schlussfolgerung.
- Jede gültige OWL-DL-Ontologie ist eine gültige OWL-Full-Ontologie.
- Jede gültige OWL-DL-Schlussfolgerung ist eine gültige OWL-Full-Schlussfolgerung.

[10] Ein formales System ist vollständig, wenn jeder gültige wohlgeformte Ausdruck durch Anwendung von Ableitungsregeln aus einer Menge von Axiomen ableitbar ist.

[11] Ein formales System ist entscheidbar, wenn es einen Algorithmus gibt, mit dem festgestellt werden kann, ob ein beliebiger wohlgeformter Ausdruck durch Ableitungsregeln aus einer Menge von Axiomem ableitbar ist oder nicht.

OWL basiert auf dem Datenmodell von RDF. Das bedeutet, eine OWL-Ontologie ist ein RDF-Graph, der wiederum aus einzelnen Tripeln besteht. Wie bei RDF sind unterschiedliche Formen der Serialisierung einer OWL-Ontologie denkbar. Die Bedeutung einer OWL-Ontologie wird nur von dem zugrunde liegenden RDF-Graphen und der Semantik der OWL-Sprachelemente bestimmt. Das folgende Beispiel nutzt die Syntax von RDF/XML.

```
<rdf:RDF
   xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3c.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3c.org/2002/07/owl#"
   xml:base="http://www.beispiel.de/comic/begriffe#">

   <owl:Class rdf:ID="Mensch"/>

   <owl:Class rdf:ID="Zeichner">
      <rdfs:subClassOf rdf:resource="#Mensch"/>
   </owl:Class>

   <owl:ObjectProperty rdf:ID="istZeichnerVon">
      <owl:inverseOf rdf:resource="#hatZeichner"/>
      <rdfs:domain rdf:resource="#Zeichner"/>
      <rdfs:range rdf:resource="#Comicfigur"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="hatZeichner">
      <owl:inverseOf ref:resource="istZeicherVon"/>
      <rdfs:domain rdf:resource="#Comicfigur"/>
      <rdfs:range rdf:resource="#Zeichner"/>
   </owl:ObjectProperty>

   <owl:Class rdf:ID="Comicfigur">
      <owl:disjointWith rdf:resource="#Zeichner"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#hatZeichner"/>
            <owl:minCardinality
               rdf:datatype="&xsd;nonNegativeInteger">
               1
            </owl:minCardinality>
```

```
        </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>
</rdf:RDF>
```

Das Beispiel definiert drei Klassen – *Mensch*, *Zeichner* und *Comicfigur*. Die Klasse *Zeichner* ist Unterklasse der Klasse *Mensch*. Die Klassen *Zeichner* und *Comicfigur* sind disjunkt. Des Weiteren werden die inversen Properties *istZeichnerVon* und *hatZeichner* definiert. Die Property *istZeichnerVon* darf auf Klassen des Typs *Zeichner* angewandt werden und lässt Individuen der Klasse *Comicfigur* als Werte zu. Die Domäne der Property *hatZeichner* ist die Klasse *Comicfigur*, ihr Wertebereich die Klasse *Zeichner*. Außerdem ist *Comicfigur* Unterklasse einer anonymen Klasse, die mit mindestens einem Individuum über *hatZeichner* in Beziehung steht. Das Beispiel ist OWL DL konform, da OWL Lite das Sprachelement *owl:disjointWith* verbietet.

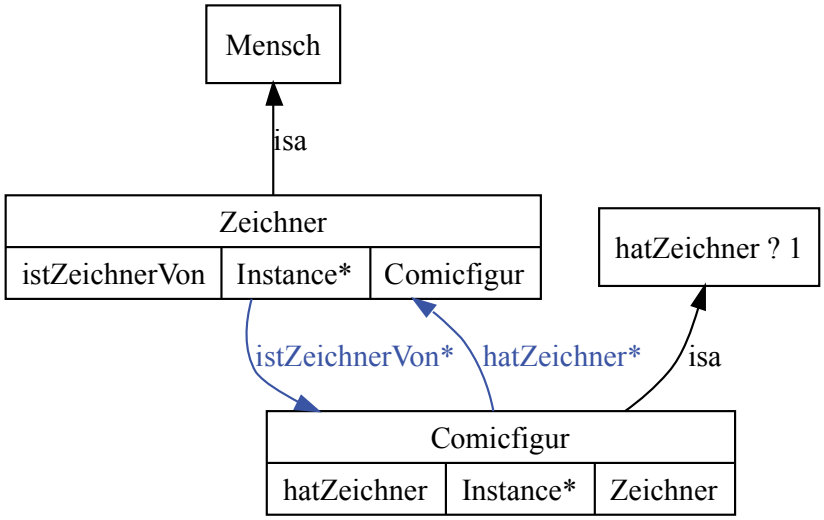

Abbildung 2.5: Graphische Darstellung des OWL-Beispiels

OWL ist eine Ontologiesprache für das World Wide Web. Dokumente unterschiedlichen Datentyps, die über HTML-Verweise miteinander verknüpft sind, bilden das World Wide Web. OWL ist entwickelt worden, diese Dokumente zu beschreiben und Schlussfolgerungen auf Basis mehrere Dokumente zu tätigen. OWL baut auf RDF/RDFS auf und nutzt somit einige Eigenschaften von RDF/RDFS, die eine Ontologiesprache für ein verteiltes System braucht:

- Die Verwendung von URIs als globale eindeutige Bezeichner erlaubt es, Informationen über Ressourcen dezentral zu verwalten.
- Das Konzept der klassenunabhängigen Properties steht im Gegensatz zu dem Konzept des Frames, der zusammengehörendes Wissen bündelt, indem er aus mehreren Attributfeldern (Slots) besteht. Durch den Einsatz von RDF/RDFS und OWL kann jeder die Beschreibung

einer Ressource erweitern. Zum Beispiel kann eine Property definiert werden und dieser fremde Klassen – eindeutig identifizierbar anhand ihrer URIs – als Domäne und Wertebereich zugewiesen werden. Für beliebige Ressourcen, die Instanzen der korrekten Klasse – Domäne der Property – sind, können Werte für diese Property angegeben werden.

Im Gegensatz zu OWL spezifiziert RDF/RDFS hauptsächlich, *wie* Ressourcen im World Wide Web beschrieben werden (graphenbasiertes Datenmodell). Bei der Entwicklung von OWL stand im Vordergrund, ein *Vokabular* zu definieren, das die in Dokumenten gebrauchte Terminologie beschreibt. In die Entwicklung flossen auch Anforderungen bestehender Webanwendungen und ontologiespezifische Erfordernisse (Erweiterung, Verteilung, Internationalisierung) ein. Eine ausführliche Diskussion dazu findet sich in [OWL04c].

2.3 WordNet – Ein semantisches Netz der englischen Sprache

Die im Rahmen dieser Arbeit zu entwickelnde Verknüpfungsengine soll das Weben eines semantischen Netzes innerhalb des HyLOS-Systems erleichtern. WordNet [Wor05], ein bekanntes semantisches Netz, wird zum besseren Verständnis des Konzeptes semantisches Netz in diesem Abschnitt vorgestellt.

Semantische Netze wurden ursprünglich in der Künstlichen Intelligenz zur Wissensrepräsentation eingesetzt. Ein semantisches Netz ist ein gerichteter, markierter Graph. Die Knoten des Graphen repräsentieren Konzepte, die Kanten Beziehungen zwischen den Konzepten.

WordNet ist ein semantisches Netz, das die englische Sprache lexikalisch abbildet. Es wurde an der Princeton-Universität entwickelt. WordNet organisiert Substantive, Verben, Adjektive und Adverbien in Synonymgruppen. Eine Synonymgruppe enthält Wörter, die dasselbe Konzept bezeichnen. Benannte Beziehungen verknüpfen die einzelnen Synonymgruppen. Folgende der Sprachwissenschaft entnommene Begriffe sind Beispiele für benannte Beziehungen:

- *Hyperonym* – Oberbegriff (Säugetier ist Hyperonym von Hund)
- *Hyponym* – Unterbegriff (Hund ist Hyponym von Säugetier)
- *Meronym* – Begriff, der in einer Teil-von-Beziehung zu anderen Wörtern steht (Nase ist Meronym von Gesicht)
- *Holonym* – Begriff, der in Bezug auf andere Wörter deren Gesamtheit bezeichnet (Gesicht ist Holonym von Nase)

WordNet kann sowohl online genutzt als auch heruntergeladen und in externe Progamme integriert werden. Das Ergebnis einer WordNet-Suche gibt Auskunft über die Einordnung des gesuchten Wortes in den englischen Wortschatz. Abbildung 2.6 veranschaulicht WordNet anhand des Wortes *bird*.

Noun

- S: (n) bird (warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings)
  - *direct hyponym* / *full hyponym*
  - *direct part meronym* / *full part meronym*
  - *direct member holonym* / *inherited member holonym*
  - ***direct hypernym*** / *inherited hypernym* / *sister term*
    - S: (n) vertebrate, craniate (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
  - *domain term category*
  - *derivationally related form*
- S: (n) bird, fowl (the flesh of a bird or fowl (wild or domestic) used as food)
- S: (n) dame, doll, wench, skirt, chick, bird (informal terms for a (young) woman)
- S: (n) boo, hoot, Bronx cheer, hiss, raspberry, razzing, razz, snort, bird (a cry or noise made to express displeasure or contempt)
- S: (n) shuttlecock, bird, birdie, shuttle (badminton equipment consisting of a ball of cork or rubber with a crown of feathers)

Verb

- S: (v) bird, birdwatch (watch and study birds in their natural habitat)

Abbildung 2.6: Funktionsweise WordNets anhand des Wortes *bird*

Abbildung 2.6 zeigt, dass das Wort *bird* sowohl ein Substantiv als auch ein Verb ist. Jeder Aufzählungspunkt steht für genau ein Konzept. Sind mehrere Wörter für ein Konzept angegeben, so handelt es sich um Synonyme. Das Substantiv *bird* ist fünf verschiedenen Konzepten zugeordnet, das Verb *bird* nur einem. Der erste Aufzählungspunkt der Substantivliste steht für das Konzept *Vogel*. Die Liste der benannten Beziehungen wird für das Konzept *Vogel* angezeigt. Die Beziehung *direktes Hyperonym* (direct hypernym) ist geöffnet. Das Konzept *Wirbeltier* ist allgemeiner als das Konzept *Vogel*. Die Wörter *vertebrate* und *craniate* stehen für das Konzept *Wirbeltier*.

Kapitel 3

Lernobjektverknüpfung

3.1 Zielstellung

Der LOM-Standard ermöglicht das Verknüpfen von Lernobjekten mittels benannter Relationen und somit das Aufbauen eines semantischen Wissensnetzes. Solch ein semantisches Netz kann Lernenden dienen, selbstständigt neues Wissen zu erarbeiten. Die benannten Relationen können Wegweisern gleich Lernende zu neuartigen Lerninhalten – beispielsweise Grundlagen-, verwandtes oder vertiefendes Wissen – führen. Ebenso können Autoren und Suchmaschinen semantische Wissensnetze nutzen.

Autoren von Lernobjekten stehen vor der Herausforderung, ein semantisches Wissensnetz zu weben. Je mehr Lernobjekte in einem System vorhanden sind, desto aufwendiger ist es, Lernobjekte auf mögliche Beziehungen zu überprüfen. Von einem Autor kann nicht erwartet werden, einen vollständigen Überblick über den Lernobjektbestand zu besitzen. Er müsste demzufolge nach der Erstellung eines Lernobjektes nach passenden Lernobjekten in Bezug auf die möglichen benannten Relationen suchen. Wünschenswert ist es deshalb, Beziehungen zwischen Lernobjekten unter Beachtung der Relationssemantik automatisiert identifizieren und setzen zu können.

Im Rahmen dieser Diplomarbeit soll eine semantische Verknüpfungsengine für das Hypermedia Learning Object System HyLOS entwickelt werden. HyLOS bildet den LOM-Standard vollständig ab, somit sind die Ergebnisse dieser Arbeit bis auf die HyLOS-spezifische Implementierung allgemein verwendbar.

Bevor Strategien zum maschinellen Identifizieren von Lernobjektbeziehungen entwickelt werden können, muss die Semantik der LOM-Relationen auf ihren Nutzen im Lernobjektumfeld geprüft und gegebenenfalls präzisiert oder angepasst werden. Relationen, für die es in einer Lernumgebung keine Verwendung gibt, sind überflüssig. Das Vorhandensein überflüssiger Relationen lässt das Ziel, Relationen automatisch zu identifizieren, nicht näher kommen. Nur wenn die Semantik einer Relation genau definiert ist, kann das Identifizieren einer Relation in geeigneter Art und Weise formalisiert werden. Zu untersuchen ist außerdem, ob die Menge der Rela-

tionen aus didaktischer Sicht ausreicht oder eine Ergänzung notwendig ist. Der LOM-Standard erlaubt die Erweiterung seiner Vokabulare. Nachdem definiert wurde, welche Relationen zur Lernobjektverknüpfung genutzt werden dürfen, können Strategien zum automatischen Identifizieren entwickelt und umgesetzt werden. Der gesamte LOM-Metadatensatz eines Lernobjektes steht hierfür zur Verfügung. Welche Metadaten geeignet sind, wird während der folgenden Strategieentwicklung untersucht.

Als Ergebnis dieser Diplomarbeit sollen Relationen mit wohldefinierter Semantik, mittels derer sich Lernobjekte verknüpfen lassen, vorliegen. Die zu entwickelnde Verknüpfungsengine soll die Metadaten der Lernobjekte nutzen, um ein semantisches Netz aus Lernobjekten und benannten Relationen zu weben.

3.2 Relationen

3.2.1 Anforderungen an Relationen

Die Relationen sollen im Lernobjektumfeld eingesetzt werden. Deshalb muss ihre Semantik geeignet sein, Lernobjekte zu verknüpfen. Ihre Aussagekraft sollte den in Abschnitt 3.1 aufgezeigten Nutzen eines semantischen Netzes ermöglichen, indem eine Relation auf ein neuartiges Thema weist. Dabei muss die Art der Beziehung zwischen zwei Themen wohldefiniert sein. Allein die Semantik *neuartig* ist wenig aussagekräftig. Relationen müssen voneinander unterschieden werden können, sie dürfen nicht Synonymen gleich eingesetzt werden. Mit Blick auf das Ziel, Beziehungen zwischen Lernobjekten automatisch zu identifizieren, sollten Relationen und ihre Eigenschaften formal darstellbar sein.

3.2.2 Semantik der Dublin-Core-Relationen

Die Relationen, welche der LOM-Standard vordefiniert, wurden dem Dublin-Core-Metadatenstandard entnommen. Dublin Core definiert ein allgemeiner einsetzbares Vokabular zum Beschreiben von Dokumenten als LOM. Er orientiert sich vor allem an den Bedürfnissen von Bibliothekaren, Bestände zu katalogisieren. Tabelle 3.1 enthält die Dublin-Core-Relationen und ihre Bedeutungen [Dub05] [Dub98].

Die übernommenen Relationen sind für den Einsatz im Lernobjektumfeld ungeeignet. Sie werden den didaktischen Ansprüchen einer Lernplattform nicht gerecht. Zum einen bezieht sich die Semantik einiger Dublin-Core-Relationen nicht auf den Inhalt einer Ressource, sondern auf ihre technischen Merkmale. Zum anderen gibt es für die Relationen, deren Semantik inhaltsorientiert ist, im Kontext von Lernobjekten keine Verwendung.

Zu den Relationen, die Ressourcen aufgrund technischer Merkmale in Beziehung setzen, gehören *hasPart/isPartOf, hasVersion/isVersion, hasFormat/isFormatOf* und zum Teil auch *requires/isRequiredBy*. Die technischen Relationen führen nicht dazu, neue Inhalte – im Sinne von

Relationsname	Bedeutung
hasPart/isPartOf	Dieses Relationspaar drückt die Beziehung von Teilen eines Ganzen oder Gesamtwerkes zu dem Ganzen oder Gesamtwerk aus. Einbezogen werden sowohl physikalische als auch logische Teile.
hasVersion/isVersion	Die Relationen beschreiben Beziehungen zwischen verschiedenen Ausgaben bzw. historischen Ständen eines inhaltlich gleichen Werkes desselben Verfassers.
hasFormat/isFormatOf	Die Relationen beschreiben Beziehungen aufgrund eines Formatwechsels, wobei eine Ressource von einer anderen durch Reproduktion oder technologische Neuformatierung abgeleitet wurde. Es handelt sich nicht um eine Interpretation der ursprünglichen Ressource, sondern um eine weitere Darstellung.
references/isReferencedBy	Durch diese Relationen wird eine vom Autor behauptete inhaltliche oder zeitliche Verwandtschaft – Hinweise, Literaturzitate etc. – ausgedrückt. So können zum Beispiel einzelne Bilder eines Künstlers zu anderen Werken des Künstlers, seiner Biographie oder anderen Bildern derselben Epoche in Verbindung gestellt werden.
isBasedOn/isBasisFor	Dieses Relationspaar beschreibt eine schöpferische Beziehung. Damit ist eine Beziehung zwischen einer Ressource und ihrer künstlerischen oder übersetzungstechnischen Überarbeitung gemeint.
requires/isRequiredBy	Besteht eine Abhängigkeitsbeziehung zwischen zwei Ressourcen, so benötigt eine Ressource eine andere, um zu funktionieren, um die enthaltenen Informationen zu liefern oder damit ihr Inhalt genutzt werden kann. Sie kann nicht ohne die mit ihr in Beziehung stehende Ressource verwandt werden.

Tabelle 3.1: Dublin-Core-Relationen

zusätzlich und nicht im Sinne von *anders dargestellt* – zu entdecken. Neben dieser Unzulänglichkeit besitzt das Relationspaar *requires/isRequiredBy* eine weitere. Es drückt unter Umständen Informationen redundant aus. Die Relationen *requires/isRequiredBy* setzen eine Ressource in Beziehung zu einer benötigten Ressource. Software kann eine benötigte Ressource darstellen. Der LOM-Standard ermöglicht in der Kategorie *Technical* die Angabe des MIME-Types, welcher den Datentyp spezifiziert und dadurch der Indentifikation benötigter Verarbeitungssoftware dient. In derselben Kategorie können zusätzlich technische Anforderungen – Software inbegriffen – angegeben werden.

Aussagen über die inhaltliche Verwandtschaft zweier Ressourcen treffen die Relationen *references/isReferencedBy*, *isBasedOn/isBasisFor* und *requires/isRequiredBy*. Das Relationspaar *requires/isRequiredBy* ist nochmals aufgeführt, weil es neben einer technischen auch eine inhaltliche Abhängigkeit – eine Ressource baut inhaltlich auf einer anderen auf – ausdrücken kann. Die Beziehungen, die durch die inhaltsorientierten Relationen abgebildet werden, treten

zwischen Lernobjekten mit großer Wahrscheinlichkeit nicht auf. Das Relationspaar *isBasedOn/ isBasisFor* drückt beispielsweise eine schöpferische Beziehung aus. Dabei kann es sich um die Beziehung eines Filmes zu einem Buch, das künstlerisch in diesem Film überarbeitet wurde, handeln. Zum Verknüpfen von Lernobjekten können *isBasedOn/isBasisFor* aufgrund ihrer Semantik nicht genutzt werden.

Es wurde gezeigt, dass die möglichen Relationen nicht didaktisch zweckmäßig sind und deshalb die Notwendikeit besteht, ihre Semantik zu konkretisieren und interpretieren. Der LOM-Standard erlaubt das Erweitern seiner Vokabulare, somit können zusätzlich die Relationen ergänzt werden.

Sowohl Konkretisierung und Interpretation als auch Erweiterung schränken die semantische Interoperabilität der Lernobjekte ein. Alle Lernplattformen, die die ergänzenden Relationen nicht übernehmen, kennen deren Semantik nicht. Diese Lernplattformen können nur die Information, dass zwei Lernobjekte in Beziehung zueinander stehen, verarbeiten. Das stellt eine Einschränkung dar, ist jedoch nicht falsch. Anders verhält es sich mit den konkretisierten und interpretierten Relationen. Lernplattformen, die auf der ursprünglichen Semantik der Relationen aufbauen, verarbeiten die in ihrer Semantik veränderten Relationen problemlos; die daraus folgenden Ergebnisse können jedoch falsch sein. Werden Lernobjekte nur präsentiert, ist die negative Auswirkung gering. Ein Anwender kann unter Umständen eine ihm präsentierte Lernobjektbeziehung nicht nachvollziehen, da er eine andere Auffassung von der Bedeutung einer Relation hat. Werden anhand der Relationen Schlüsse gezogen, hat eine nicht einheitliche Relationssemantik negative Folgen. Falsche Informationen werden erarbeitet.

Aufgrund der aufgezeigten Unzulänglichkeiten der Dublin-Core-Relationen, müssen sie trotz der daraus folgenden Probleme an die Anforderungen einer Lernplattform angepasst werden. LOM sollte diesbezüglich überarbeitet werden. Die im Rahmen der Diplomarbeit vorgenommenen Anpassungen könnten als Anregung dafür dienen.

3.2.3 Konkretisierung und Interpretation der Dublin-Core-Relationen

In einem ersten Schritt wurden die Relationen ausgewählt, deren Semantik nicht geändert, sondern konkretisiert werden musste, um im Kontext von Lernobjekten einsetzbar zu sein (Tabelle 3.2). Zum einen wurden die Anforderungen für den Gebrauch der Relationen gelockert, zum anderen beschränkt. Das Relationspaar *hasVersion/isVersionOf* darf entgegen der ursprünglichen Semantik auch bei unterschiedlichen Autoren gesetzt werden. Von den inversen Relationen *hasFormat/isFormatOf* wurde nur *isFormatOf* als symmetrische Relation beibehalten, da nicht zwischen Ursprungs- und umformatiertem Lernobjekt unterschieden werden soll. Die Relation *isFormatOf* soll Lernobjekte desselben pädagogischen Niveaus verknüpfen, deshalb ist die Semantik in Bezug auf die Daten der Kategorie *Educational* des LOM-Standards konkretisiert worden. Es ist nicht sinnvoll, ein Lernobjekt über Braunbären, das für Grundschulkinder gedacht

ist, mittels *isFormatOf* in Beziehung zu einem weiteren Lernobjekt zum Thema Braunbären auf Oberschulniveau zu setzen.

Relationsname	Bedeutung
hasPart/isPart	Mittels der Relationen hasPart und isPartOf werden Lernobjekte in einer hierarchischen Struktur geordnet. Alle Lernobjekte solch einer Struktur bilden zusammen ein Sinngefüge. Ein Elternlernobjekt ist inhaltlich als Summe aller seiner direkten und indirekten Kindlernobjekte aufzufassen. Ein Kindlernobjekt befindet sich immer in demselben Kontext wie alle seine Elternlernobjekte, oft ist es eine Spezialisierung. Die Relationen hasPart und isPartOf drücken jedoch primär eine Strukturbeziehung aus und spezialisieren nicht die Art und Weise der inhaltlichen Beziehung. Die Relationen hasPart und isPartOf sind gleichwertig zu den Beziehungen verschachtelter Lernobjekte untereinander. isPartOf ermöglicht einem Autor, eine Strukturbeziehung auszudrücken, ohne Schreibrechte auf das Elternlernobjekt zu besitzen, indem er auf dem Kindlernobjekt die Relation isPartOf zum Elternlernobjekt definiert.
hasVersion/isVersion	Das Relationspaar hasVersion/isVersionOf verknüpft ein Lernobjekt mit seiner Version und umgekehrt. Eine Version entsteht, indem ein Lernobjekt überarbeitet, fortentwickelt, aktualisiert oder erweitert wird. Der Autor einer Version muss nicht zwingend auch Autor des ursprünglichen Lernobjektes sein. Eine Version ist dem ursprünglichen Lernobjekt ähnlich; der Inhalt einer Version ändert sich nicht grundlegend; das Format wird beibehalten.
isFormatOf	Die Relation isFormatOf drückt aus, dass zwei Lernobjekte im Wesentlichen denselben Inhalt präsentieren, jedoch ein unterschiedliches Format besitzen. Sie bedeutet nicht zwingend, dass zwei Lernobjekte alternativ einsetzbar sind. Die Angaben der LOM-Datenelemente 5.6 (Context), 5.7 (Typical Age Range), 5.8 (Difficulty) weichen nicht groß voneinander ab (siehe Tabelle 3.7 Seite 33).

Tabelle 3.2: Konkretisierung der Dublin-Core-Relationen

In einem zweiten Schritt wurde die Semantik der verbleibenden Relationen mit dem Ziel, Lernobjekte verknüpfen zu können, interpretiert (Tabelle 3.3). Die Relationspaare *references/isReferencedBy, isBasedOn/isBasisFor* und *requires/isRequiredBy* wurden dabei in ihrer Bedeutung stark verändert. Die drei Relationspaare stellen thematische Abhängigkeiten zunehmender Stärke dar. Im Gegensatz zu den Relationen des ersten Schrittes erfüllen diese die Anforderung, auf neuartige Lerninhalte zu verweisen.

3.2.4 Erweiterung der Dublin-Core-Relationen

Neben der Konkretisierung und Interpretation wurde in einem dritten Schritt die Menge Relationen erweitert. Welche Relationen sind aus didaktischer Sicht sinnvoll? Welche Informationen

Relationsname	Bedeutung
references/isReferencedBy	Das Relationspaar references/isReferencedBy setzt ein Lernobjekt in Beziehung zu einem zweiten Lernobjekt, das Zusatzinformationen, die nicht notwenig für das Verstehen des Lernobjektes sind, enthält. Diese Zusatzinformationen sind für interessierte Lernende gedacht. Sie sind Informationen zu verwandten Themen, weiterführenden Themen, Themen, die im Zusammenhang mit dem Lernobjekt von Interesse sind, Themen, die das Lernobjekt am Rande anspricht.
isBasedOn/isBasisFor	Das Relationspaar isBasedOn/isBasisFor drückt aus, dass ein Lernobjekt die inhaltliche Grundlage eines anderen Lernobjektes bildet. Das Lernobjekt, welches die inhaltliche Grundlage bildet, ist zum Verstehen nicht notwendig; es erleichtert das Verstehen jedoch.
requires/isRequiredBy	Das Relationspaar requires/isRequiredBy drückt eine Abhängigkeitsbeziehung aus. Ein Lernobjekt braucht ein anderes Lernobjekt zwingend, um verstanden zu werden. Falls gewährleistet ist, dass der Inhalt eines benötigten Lernobjektes bekannt ist, kann ein Lernobjekt ohne das benötigte Lernobjekt genutzt werden.

Tabelle 3.3: Interpretation der Dublin-Core-Relationen

sind bereits in anderen Kategorien des LOM-Standards kodiert? Die Antworten auf die Fragen, flossen in den Prozess der Relationserweiterung ein. Nachfolgend sind die Anforderungen an die Relationsergänzungen aufgelistet.

- Die ergänzenden Relationen sollen Aussagen über die inhaltliche Verwandtschaft zweier Lernobjekte treffen.
- Die ergänzenden Relationen sollen Lernende beim selbstständigen Lernen unterstützen.
- Die ergänzenden Relationen sollen keine seltenen Spezialfälle abdecken, sondern ihr Einsatz innerhalb Lernplattformen häufig möglich sein.
- Die Eigenschaft der paarweise inversen Relationen soll möglichst beibehalten werden.
- Die Menge Relationen soll um so wenig Relationen/Relationspaare wie möglich ergänzt werden.

Mit der Erweiterung der Relationen haben sich bereits andere Autoren beschäftigt. Der Fokus von [KS04] liegt auf Metadaten, die Zugangsmöglichkeiten zu Lerninhalten beschreiben. Erweiterungsvorschläge sind *hasVisualAlternative, hasTextAlternative* und *hasAuditoryAlternative*. Die Virtual-Campus-Plattform [tea03] nimmt *isAlternativeTo* und *requiredOnFailure* in ihre Menge an Relationen auf. Sie dienen einer optimierten Kursmodellierung. Die Relation *requiredOnFailure* verknüpft ein Lernobjekt, das Lernstoff abfragt, mit einem Lernobjekt, welches

vom Lernenden durchgearbeitet werden muss, sollte der Lernende den Test nicht bestehen. Die Relation *isAlternativeTo* setzt zwei Lernobjekte, die in ihrer didaktischen Funktion gleichwertig sind, in Beziehung zueinander. [SSFS99] definiert didaktische Relationen, die Spezialfälle abbilden, wie *example* und *illustrates*. Die Relation *example* verknüpft ein Lernobjekt[1] mit einem Beispiel. Ein Lernobjekt wird mit seiner graphischen Darstellung mittels *illustrates* in Beziehung gesetzt.

Relationsname	Bedeutung
isNarrowerThan/ isBroaderThan	Eine Spezialisierungs- bzw. Verallgemeinerungsbeziehung liegt vor, wenn ein Lernobjekt ein Wissensgebiet beschreibt, das eine Spezialisierung bzw. Verallgemeinerung des Wissensgebietes, welches ein anderes Lernobjekt beschreibt, darstellt. Taxonomien werden hauptsächlich durch diese Beziehungen aufgebaut.
isAlternativeTo	Zwei Lernobjekte stellen Alternativen zueinander dar, wenn sie inhaltlich, pädagogisch und strukturell gleichwertig und somit untereinander austauschbar sind. Zwei Lernobjekte sind strukturell gleichwertig, wenn sie innerhalb eines Lernobjektgefüges dieselbe Position einnehmen können; so sind zum Beispiel zwei Einleitungen strukturell gleichwertig. Alternative Lernobjekte können unterschiedliche Formate besitzen.
illustrates/IsIllustratedBy	Diese Relation drückt aus, dass ein Lernobjekt ein anderes Lernobjekt veranschaulicht oder erläutert. Das kann durch konkrete Beispiele, Diagramme, Tabellen, Schaubilder, Graphiken, erklärende Texte etc. geschehen.
isLessSpecificThan/ isMoreSpecificThan	Das Relationspaar isLessSpecificThan/isMoreSpecificThan verknüpft zwei Lernobjekte, die im Wesentlichen denselben Inhalt darstellen, eines der Lernobjekte diesen jedoch ausführlicher/detaillierter behandelt. Zum Beispiel ist die Einleitung in ein Thema weniger detailreich als dessen ausführliche Darstellung. Eine andere Situation, in der dieses Relationspaar verwandt werden kann, liegt in der thematischen Überlappung von Lernobjekten. Das heißt, zwei Lernobjekte behandeln denselben Themenkomplex; ein Lernobjekt ist dann präziser als ein zweites Lernobjekt, wenn es den Inhalt des zweiten Lernobjektes im Wesentlichen ebenfalls behandelt, jedoch zusätzlich den Lernstoff vertieft.

Tabelle 3.4: Erweiterungen der Dublin-Core-Relationen

Tabelle 3.4 enthält die im Rahmen dieser Arbeit entwickelten Erweiterungen. Das Relationspaar *isNarrowerThan/isBroaderThan* wurde unter anderem in die Menge der Relationen aufgenommen, weil eine Spezialisierungs- beziehungsweise Verallgemeinerungsbeziehung direkt

[1] In [SSFS99] ist anstatt von Lernobjekten von Media Bricks die Rede. Ein Media Brick ist eine Entität, die entweder Text oder Multimediaelemente wie Bilder, Videos und Audiodateien enthält. Um Verwirrung zu vermeiden, wird im Haupttext das Wort Lernobjekt verwandt.

aus der Taxonomie, anhand der Lernobjekte klassifiziert werden, ablesbar ist. Das Ziel ist es, eine Relation zwischen zwei Lernobjekten automatisch zu identifizeren. Dafür bilden *isNarrowerThan/isBroaderThan* einen Einstiegspunkt. Unabhängig davon sind diese Relationen im Lernobjektumfeld wertvoll; sie weisen auf speziellere beziehungsweise allgemeinere Themen. Einige der aufgeführten Erweiterungen anderer Autoren dienten als Anregungen für die Relationen *isAlternativeTo* und *illustrates/isIllustratedBy*. Die Definition des Relationspaares *isLessSpecificThan/isMoreSpecificThan* war ein mehrstufiger Prozess. Im Interesse, die Relationserweiterungen zu beschränken, sollte das gesuchte Paar inverser Relationen mehrere Spezialfälle wie Einleitung, Überblick, Zusammenfassung auf der einen Seite und Vertiefung auf der anderen Seite abdecken. Eine erste Idee war das Relationspaar *isMoreDetailedThan/isLessDetailedThan*. Sowohl eine Einleitung als auch ein Überblick oder eine Zusammenfassung ist weniger detailreich als eine ausführliche Darstellung eines Lerninhaltes. Der Spezialfall der Vertiefung lässt sich jedoch hierdurch nicht abbilden. Vertiefende Inhalte sind keine ausführlicheren Darstellungen, sondern zusätzliche Inhalte. Zum Beispiel kann das Thema Namensräume in XML als Vertiefung von XML im Allgemeinen bezeichnet werden. Deshalb wurde überlegt, ob stattdessen von einer Anreicherung oder Aufwertung (*enhances/isEnhancedBy*) eines Lernobjektes gesprochen werden kann. Somit wäre der Fall Vertiefung mit einbegriffen. Ein Lernobjekt wird durch ein vertiefendes Lernobjekt angereichert/aufgewertet. Allerdings erschien die Semantik der Relation zu allumfassend, da auch Lernobjekte von Beispielen oder Zusatzinformationen angereichert werden. Diese Fälle werden von vorhandenen Relationen jedoch abgedeckt. Letztendlich ist das Relationspaar *isLessSpecificThan/isMoreSpecificThan* gewählt worden. Einleitungen, Überblicke und Zusammenfassungen sind weniger präzise als ausführliche Darstellungen eines Themas. Das gefundene Relationspaar kann bei Themenüberlappung ebenfalls Vertiefungen darstellen (siehe nachfolgendes Beispiel); in einigen Fällen kann auch *isNarrowerThan/isBroaderThan* dafür genutzt werden.

Abbildung 3.1 zeigt ein Netz von Lernobjekten, die unter Verwendung der in diesem Abschnitt definierten Relationen miteinander verknüpft sind. Das Beispiel soll vor allem die Semantik der Relationen verdeutlichen. Es enthält nicht die Relationen *hasVersion/isVersionOf*, *isFormatOf* und *isAlternativeTo*, weil sich diese Relationen nicht gut zwischen Lernobjekten, von denen nur der Titel bekannt ist, darstellen lassen. Jede in dem Beispiel enthaltene Relation wird anhand eines Vorkommens erklärt: Zwei Lernobjekte, die konkrete Taxonomien – ACM Computing Classification System und Dewey Decimal Classification – behandeln, veranschaulichen oder sind Beispiele für (*illustrates*) ein Lernobjekt, das Taxonomien im Allgemeinen zum Thema hat. Ein Lernobjekt kann über mehr als eine Relation in Beziehung zu einem anderen Lernobjekt stehen. Das Lernobjekt zum Thema Markup-Sprachen bildet mit den Lernobjekten zu den Themen XML und HTML ein Sinngefüge, indem es diese Lernobjekte als Unterlernobjekte besitzt (*hasPart*). Außerdem sind XML und HTML Spezialisierungen (*isNarrowerThan*) des Themas Markup-Sprachen. Das Lernobjekt, das XML und XLINK behandelt, ist über die

Relation *isMoreSpecificThan* mit dem Lernobjekt zum Thema XML verknüpft, weil es zusätzlich zu XML auch XLINK thematisiert. Hier liegt eine thematische Überlappung vor. Zwingend erforderlich (*isRequiredBy*) ist das Wissen um eine Ontologie, um sich mit der Ontologiesprache OWL zu beschäftigen. Eine schwächere thematische Abhängigkeit liegt zwischen den Themen Ontologien und RDF vor. Das Wissen über Ontologien erleichtert (*isBasisFor*) das Verstehen von RDF. RDF und OWL sind Zusatzinformationen (*isReferencedBy*) zu dem Thema Semantic Web, jedoch keine thematischen Voraussetzungen.

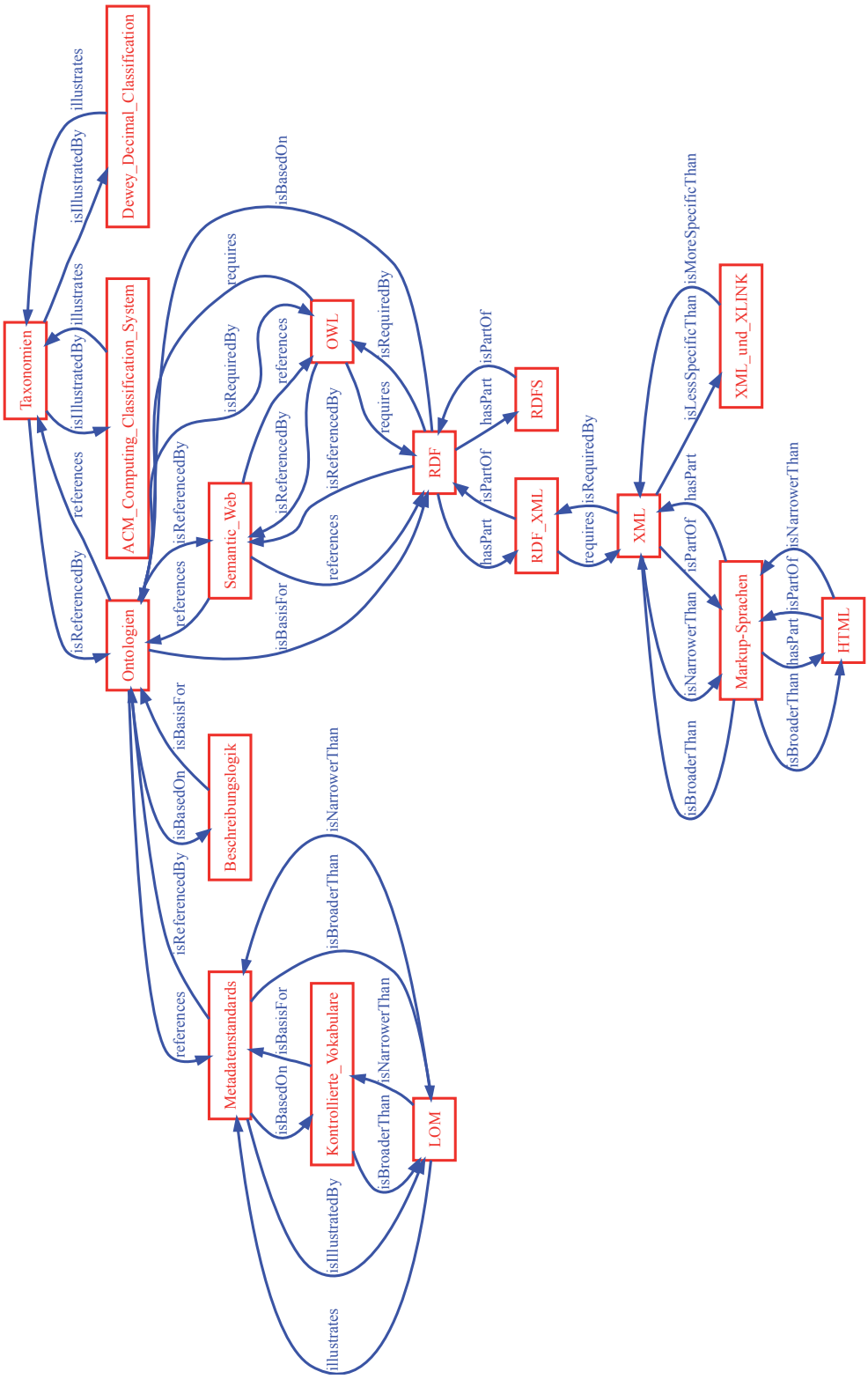

Abbildung 3.1: Veranschaulichung der Relationssemantik

3.3 Automatisches Identifizieren von Relationen

Im Rahmen dieser Arbeit soll eine semantische Verknüpfungsengine für das HyLOS-System entwickelt werden. Es wurden zehn mögliche Relationspaare beziehungsweise Relationen benannt und ihre Semantik definiert. Wird ein neues Lernobjekt dem HyLOS-System hinzugefügt, so kann es potentiell mit jedem Lernobjekt innerhalb des Systems in Beziehung stehen. Von einem Autor kann nicht erwartet werden, alle Lernobjekte auf eine mögliche Beziehung zu dem neuerstellten Lernobjekt zu prüfen. Das Problem, welches es zu lösen gilt, liegt darin, eine Maschine in die Lage zu versetzen, Lernobjekte korrekt, unter Beachtung der Relationssemantik zu verknüpfen. Die Strategien zur automatischen Relationsfindung lassen sich in zwei Gruppen unterteilen:

- Strategien der ersten Gruppe finden Relationen anhand der in den Metadaten von Lernobjekten enthaltenen Informationen.
- Strategien der zweiten Gruppe nutzen vorhandene Lernobjektverknüpfungen, um daraus neue Relationen zu erschließen.

3.3.1 Identifizieren von Relationen anhand von Metadaten

3.3.1.1 Aus Taxonomien ablesbare Relationen

Die Relationen *isNarrowerThan/isBroaderThan* können direkt automatisch identifiziert werden. Lernobjekte werden anhand von Taxonomien klassifiziert. Diese Information wird in der LOM-Kategorie *Classification* abgelegt. Ein Lernobjekt ist eine Spezialisierung eines anderen Lernobjektes, wenn es innerhalb desselben Taxonomiezweiges eingeordnet, jedoch einem tiefer angesiedelten Taxonomieknoten zugeordnet wurde. Bei einer Verallgemeinerungsbeziehung gilt die Umkehrung. Es wurde darauf hingewiesen, dass Taxonomien nicht ausschließlich hierarchisch aufgebaut sind. Das innerhalb des HyLOS-Systems eingesetzte ACM Computing Classification System enthält Querverweise. Der Taxonomieknoten *D.3.3 Language Constructs and Features* (D. Software/D.3 Programming Languages) verweist auf *E.2 Data Storage Representations* (E. Data). Querverweise können auf den Relationen *references/isReferencedBy* abgebildet werden. Ein Lernobjekt, das dem Taxonomieknoten D.3.3 zugeordenet ist, referenziert somit ein Lernobjekt des Taxonomieknotens E.2.

3.3.1.2 Aus Lernobjektstrukturen ablesbare Relationen

Die Relationen *hasPart/isPartOf* sind gleichwertig zu den Beziehungen verschachtelter Lernobjekte untereinander und können demzufolge aus Lernobjektstrukturen übernommen werden. Ein Lernobjekt steht zu seinen Unterlernobjekten mittels *hasPart* in Beziehung. Ein Unterlernobjekt ist mit seinem übergeordneten Lernobjekt durch *isPartOf* verknüpft.

3.3.1.3 Heuristische Verfahren

Neben den genannten Strategien, die zu logisch korrekten Lernobjektverknüpfungen führen, können heuristische Verfahren zur Identifikation von Beziehungen zwischen Lernobjekten angewandt werden. Ein heuristisches Verfahren ist eine Faustregel; Lernobjektmetadaten und -strukturen werden untersucht, um Vermutungen über mögliche Beziehungen anzustellen.

Die Relationen *isFormatOf*, *isAlternativeTo* und *isLessSpecificThan/isMoreSpecificThan* sind mit hoher Wahrscheinlichkeit durch Vergleiche bestimmter LOM-Datenelemente und der Lernobjektstrukturen identifizierbar. Diese Relationen haben gemeinsam, dass sie Lernobjekte verknüpfen, die im Wesentlichen dasselbe Thema behandeln. Tabelle 3.5 zeigt LOM-Datenelemente, die Hinweise auf inhaltliche Gleichwertigkeit geben.

LOM-Datenelement	Erläuterungen zur inhaltlichen Übereinstimmung
1.3 Language	Zwei Lernobjekte, die miteinander verknüpft sind, sollten dieselbe Sprache besitzen. Ein Sprachwechsel kann problematisch sein.
1.5 Keywords	Die Schlüsselwortlisten müssen nicht hundertprozentig, jedoch mehrheitlich übereinstimmen. Synonyme sind erlaubt.
1.6 Coverage	Zwei Lernobjekte sind derselben Zeit, Kultur, Geographie, Region zugeordnet. LOM definiert für dieses Datenelement kein kontrolliertes Vokabular. Somit ist maschinell schwer überprüfbar, ob die Zuordnung zweier Lernobjekte gleich ist.
7 Relation	Zwei Lernobjekte stehen zu denselben Lernobjekten in derselben benannten Relation.
9.2 Taxon Path	Zwei Lernobjekte sind anhand desselben Klassifikationsschemas unterhalb desselben Knotens eingeordnet.

Tabelle 3.5: Hinweise auf inhaltliche Gleichwertigkeit

Die Relation *isFormatOf* verlangt zusätzlich unterschiedliche Formate und gleichwertige Angaben für die LOM-Datenelemente 5.6 (Context), 5.7 (Typical Age Range) und 5.8 (Difficulty). Gleichwertig bedeutet in diesem Zusammenhang, dass zwei Angaben in ihrer Bedeutung nicht stark voneinander abweichen. Sofern gleiche Angaben verlangt würden, wäre die Wahrscheinlichkeit, eine Beziehung mittels der aufgezeigten heuristischen Verfahren zu identifizieren, gering. Außerdem können Autoren unterschiedliche Auffassungen von der Bedeutung einzelner Elemente eines kontrollierten Vokabulars haben und nutzen demzufolge unterschiedliche Elemente trotz gleicher beabsichtigter Aussage. Tabelle 3.7 informiert über gleichwertige Angaben. Die Unterteilung des menschlichen Altersspektrums in Altersstufen beruht auf der Annahme, dass die geistige Entwicklung in jeder – außer der ersten – Stufe ähnlich ist. Die erste Stufe umfasst das Vorschulalter. Diese Unterteilung muss mit einem Entwicklungspsychologen diskutiert werden. Das LOM-Datenelement 4.1 (Format) enthält Angaben über das Format eines Lernobjektes.

Die Relation *isAlternativeTo* verlangt neben inhaltlicher auch pädagogische und strukturelle Gleichwertigkeit. Tabelle 3.6 listet LOM-Datenelemente auf, die Hinweise auf eine pädagogische Gleichwertigkeit zweier Lernobjekte geben. Um strukturelle Gleichwertigkeit zweier Lernobjekte zu identifizieren, müssen die Lernobjektgefüge, in die sie eingebunden sind, verglichen werden. Nehmen zwei Lernobjekte ähnliche Positionen innerhalb ihrer Lernobjektgefüge an, so sind sie strukturell gleichwertig. Eine Lernobjektstruktur ist ein Baum. Obwohl die Strukturtiefe nicht begrenzt ist, wird die Struktur eines Lernobjektes weit weniger als zehn Ebenen besitzen. Die momentan im HyLOS-System vorhandenen Lernobjekte bestätigen diese Annahme. Deshalb besitzt ein Lernobjekt eine ähnliche Position, wenn es vom Wurzellernobjekt aus betrachtet höchstens eine Ebene höher oder tiefer als ein anderes Lernobjekt angesiedelt ist.

LOM-Datenelement	Art der Übereinstimmung
5.1 Interactivity Type	gleich
5.2 Learning Resource Type	gleich
5.3 Interactivity Level	gleichwertig
5.4 Semantic Density	gleichwertig
5.5 Intended End User Role	gleich
5.6 Context	gleich
5.7 Typical Age Range	gleichwertig
5.8 Difficulty	gleich
5.11 Language	gleich

Tabelle 3.6: Hinweise auf pädagogische Gleichwertigkeit

Das Relationspaar *isLessSpecificThan/isMoreSpecificThan* verknüpft dann zwei inhaltlich gleichwertige Lernobjekte, wenn eines der Lernobjekte detaillierter ist oder das andere Lernobjekt thematisch überlappt. Das LOM-Datenelement 5.4 (Semantic Density) gibt das Verhältnis des Informationsgehaltes eines Lernobjektes zu seiner Größe an. Wenn ein Lernobjekt A ein Thema kürzer und knapper darstellt – eine höhere semantische Dichte besitzt – als ein inhaltlich gleichwertiges Lernobjekt B, dann ist die Wahrscheinlichkeit hoch, dass A *isLessSpecificThan* B gilt.

Die in Abschnitt 3.3.2 aufgezeigten Strategien verarbeiten vorhandene Lernobjektverknüpfungen. Sind keine Lernobjektverknüpfungen vorhanden, so können diese Strategien nicht angewandt werden. Als Einstieg dienen neben manuell gesetzten Relationen die in Abschnitt 3.3.1 vorgestellten Strategien. Bei den aufgezeigten heuristischen Verfahren zur Identifikation von Lernobjektverknüpfungen handelt es sich um Faustregeln, die nicht in jedem Fall zu einer korrekten Verknüpfung führen müssen. Da aufgrund einer durch heuristische Verfahren identifizierten Verküpfung die Schlussfolgerungskette angestoßen werden kann und sich mögliche Fehler somit fortsetzen, ist Vorsicht geboten. Die identifizierten Beziehungen sollten vorerst nicht automatisch gesetzt werden, sondern einem Autor präsentiert und von ihm bestätigt oder abgelehnt werden. Sollte sich herausstellen, dass die heuristischen Verfahren eine geringe Fehlerquote auf-

LOM-Datenelement	gleichwertige Angaben	
5.3 Interactivity Level	very low	low
5.4 Semantic Density	low	very low, medium
	medium	low, high
	high	medium, very high
	very high	high
5.6 Context	school	-
	higher education	training
	training	higher education
	other	-
5.7 Typical Age Range	0-6	
	7-9	
	10-13	
	14-17	
	18-24	
	>=25	
5.8 Difficulty	very easy	easy
	easy	very easy, medium
	medium	easy, difficult
	difficult	medium, very difficult
	very difficult	difficult

Tabelle 3.7: gleichwertige Angaben

weisen, können sie automatisch gesetzt werden. Unkorrekte Lernobjektverknüpfung können gelöscht werden; Abschnitt 3.5 diskutiert diese Problematik.

3.3.2 Identifizieren von Relationen anhand von Lernobjektverknüpfungen

3.3.2.1 Erweiterung der Relationseigenschaften

Mit Ausnahme der Relationen *isFormatOf* und *isAlternativeTo* liegen inverse Relationspaare vor. Weitere mögliche Relationseigenschaften sind Transitivität und Symmetrie. Für das Identifizieren neuer Lernobjektbeziehungen ist die Transitivität wertvoll. Inversität und Symmetrie führen auch zu neuen Beziehungen, jedoch sind diese redundant. Alle Relationen wurden auf die genannten Eigenschaften untersucht. Tabelle 3.8 enthält das Ergebnis. Die drei genannten Relationseigenschaften lassen sich in OWL abbilden.

Die Erweiterung der Relationseigenschaften bedarf einiger Erklärungen: Die Entwicklung eines Lernobjektes soll nachvollziehbar bleiben; zwischen direkter und in mehreren Schritten erarbeiteten Version soll unterschieden werden können. Deshalb sind *hasVersion/isVersionOf* keine transitiven Relationen. Da nicht zwischen Ursprungs- und umformatiertem Lernobjekt unterschieden wird, ersetzt die symmetrische Relation *isFormatOf* das inverse Relationspaar *hasFormat/isFormatOf*. Damit Lernobjekte gleichen Formats nicht mittels *isFormatOf* verknüpft werden, ist diese Relation nicht transitiv. Folgende Konstellation hätte eine unkorrekte Ver-

Relation	invers zu	transitiv	symmetrisch
hasPart	isPartOf	ja	
isPartOf	hasPart	ja	
hasVersion	isVersionOf		
isVersionOf	hasVersion		
isFormatOf			ja
references	isReferencedBy		
isReferencedBy	references		
isBasedOn	isBasisFor		
isBasisFor	isBasedOn		
requires	isRequiredBy	ja	
isRequiredBy	requires	ja	
isNarrowerThan	isBroaderThan	ja	
isBroaderThan	isNarrowerThan	ja	
isAlternativeTo		ja	ja
illustrates	isIllustratedBy	ja	
isIllustratedBy	illustrates	ja	
isLessSpecificThan	isMoreSpecificThan	ja	
isMoreSpecificThan	isLessSpecificThan	ja	

Tabelle 3.8: Relationseigenschaften

knüpfung zur Folge: Ein Lernobjekt A, welches nur Text enthält, ist mittels *isFormatOf* mit einem Lernobjekt B bestehend aus Text und einem Video verknüpft. Lernobjekt B ist außerdem mit einem Lernobjekt C, welches ebenfalls nur Text enthält, mittels *isFormatOf* verknüpft. Die Relationen *references/isReferencedBy* stellen eine schwache thematische Abhängigkeit dar. Zusatzinformationen werden zum einen nicht für das Verstehen des Lernstoffes gebraucht, zum anderen können sie thematisch weit entfernt sein. Um Verknüpfungen zu verhindern, bei denen eine Zusatzinformation nicht mehr sinnvoll ist, sind diese Relationen nicht transitiv. Ein ähnlicher Grund liegt bei den Relationen *isBasedOn/isBasisFor* vor. Eine inhaltliche Grundlage erleichtert das Verstehen, ist jedoch nicht zwingend erforderlich. Damit thematisch zu weit entfernte „Grundlagen“ nicht verknüpft werden, sind diese Relationen nicht transitiv.

3.3.2.2 Schlussfolgerungsregeln

Bestehende Lernobjektbeziehungen lassen auf neue Beziehungen schließen. Es wurden 39 Zwei-Schritt-Regeln und 13 Regel, die aus mehr als zwei Schritten bestehen, identifiziert. Anhang A enthält eine Übersicht aller Regeln. Bei Zwei-Schritt-Regeln ist eine der beiden in den zwei Schritten enthaltenen Relationen dominant und spezifiziert die Art und Weise der neuen Lernobjektverknüpfung. Es wurde keine Zwei-Schritt-Regel identifiziert, die auf eine dritte Relation schließt. Zwei-Schritt-Regeln haben die Form:

$$\text{A in Relation } R_1 \text{ zu B} \wedge \text{B in Relation } R_2 \text{ zu C} \rightarrow \text{A in Relation } R_1 \text{ oder } R_2 \text{ zu C}$$

Zum Identifizieren der Zwei-Schritt-Regeln wurde jede mögliche Kombination zweier Relationen überprüft. So konnte beim Identifizieren der N-Schritt-Regeln nicht vorgegangen werden. Stattdessen wurden Zweierkombinationen, die zu keiner Schlussfolgerung führen, dahin gehend untersucht, ob durch zusätzliche Information eine Schlussfolgerung zulässig ist. Drei ausgewählte N-Schritt-Regeln werden anschließend erklärt (Nummerierung gemäß Tabelle A.11).

N-Schritt-Regel 1: A *isAlternativeTo* B $\wedge$ B *isFormatOf* C lässt keine Schlussfolgerung zu. Alternativen sind inhaltlich gleichwertig, jedoch dürfen sie unterschiedliche Formate besitzen. Die Schlussfolgerung A *isFormatOf* C könnte falsch sein, weil nicht ausgeschlossen ist, dass A und C dasselbe Format besitzen. Die zusätzliche Information (A *isVersionOf* B $\vee$ A *hasVersion* B) bedeutet, A und B haben das gleiche Format, da Versionen sich nicht im Format unterscheiden. Weil B sich in seinem Format von C unterscheidet, ist es zulässig, A *isFormatOf* C zu schließen.

N-Schritt-Regel 6: Aus A *isBasedOn* B $\wedge$ B *hasPart* C lässt sich kein Schluss ziehen. B hat mindestens ein Kindlernobjekt C. B ist inhaltlich als Summe aller seiner Kindlernobjekte aufzufassen. B bildet die inhaltliche Grundlage von A. Daraus zu schließen, dass auch das Kindlernobjekt C die inhaltliche Grundlage bildet ist falsch, da nicht der gesamte Inhalt von B die inhaltliche Grundlagen von A bilden muss. Wenn aber zusätzlich A *isNarrowerThan* C gilt, also A eine Spezialisierung von C ist, kann auf A *isBasedOn* C geschlossen werden. Abbildung 3.2 veranschaulicht das anhand eines Beispiels.

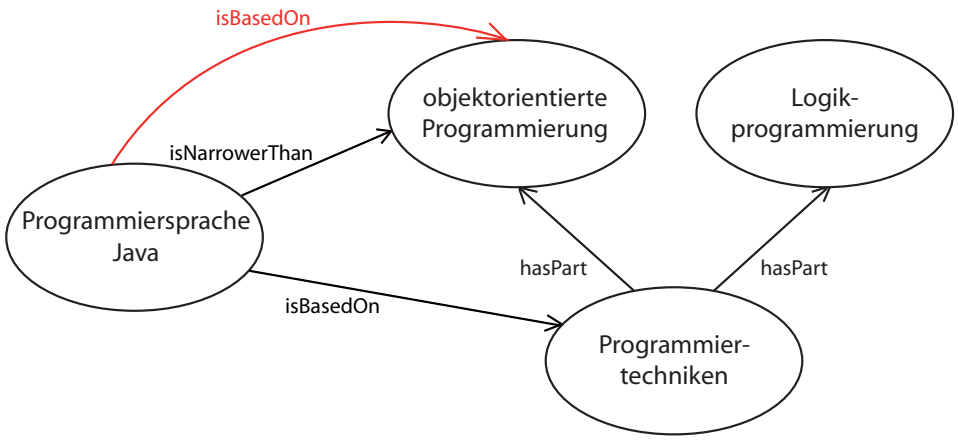

Abbildung 3.2: Beispiel für Anwendung der 6. N-Schritt-Regel

N-Schritt-Regel 11: A *illustrates* B $\wedge$ B *isBasedOn* C lässt keine Schlussfolgerung zu, da ein Beispiel einem anderen Themengebiet entnommen sein kann. Die Blocksicherung der Bahn[2] ist zum Beispiel ein themengebietfremdes Beispiel für Diskretisierung. Die zusätzliche Information

[2]Das Schienennetz der Bahn wird in Abschnitte (Blöcke) unterteilt. Innerhalb eines Abschnittes darf sich nur ein Zug befinden. Beim Einfahren in einen Abschnitt schaltet ein Zug das hinter ihm liegende Signal auf Rot, welches beim Verlassen des Abschnittes wieder auf Grün gestellt wird.

(A *isNarrowerThan* B ∨ A *isBroaderThan* B) bedeutet, dass A und B demselben Taxonomiezweig zugeordnet sind und somit demselben Themengebiet entstammen. Deshalb ist der Schluss A *isBasedOn* C zulässig. Abbildung 3.3 zeigt ein Beispiel dazu.

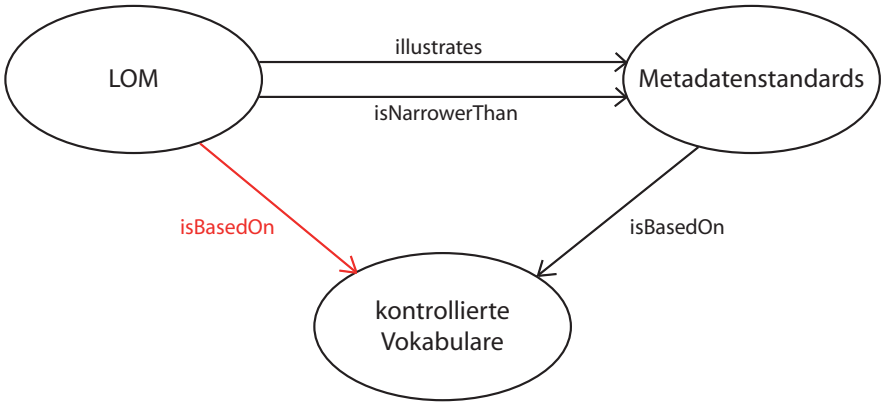

Abbildung 3.3: Beispiel für Anwendung der 11. N-Schritt-Regel

3.4 Relationen im Widerspruch

Durch manuelles Setzen von Relationen können logisch unkorrekte Lernobjektverknüpfungen entstehen. Mit logisch unkorrekt ist nicht gemeint, dass eine gesetzte Relation bezogen auf die inhaltliche Verwandtschaft zweier verknüpfter Lernobjekte falsch ist. Logisch unkorrekt bedeutet, dass eine Relation im Widerspruch zu ihrer Semantik steht. Bis auf *references/isReferencedBy* dürfen alle anderen Relationen, die eine inverse Relation besitzen, nicht wie eine symmetrische Relation benutzt werden. Ein Lernobjekt kann nicht Teil eines anderen Lernobjektes sein und dieses Lernobjekt als Unterlernobjekt besitzen. Genauso wenig kann ein Lernobjekt von einem anderen Lernobjekt gebraucht werden und dieses selber benötigen. Das Relationspaar *references/isReferencedBy* bildet aufgrund seiner Semantik eine Ausnahme unter den Paaren inverser Relationen. Es ist nicht falsch, wenn ein Lernobjekt auf ein anderes Lernobjekt, von dem es referenziert wird, als Zusatzinformation verweist. Abbildung 3.4 zeigt das anhand eines Beispiels.

Taxonomien repräsentieren wie Ontologien Wissen. Taxonomien als die schwächeren Wissensrepräsentationssysteme sind verwandt mit Ontologien; das Thema Ontologien führt das Thema Taxonomien weiter. Trotz der gegenseitigen inhaltlichen Nähe liegt eine schwache thematische Abhängigkeit vor. Das Wissen, was eine Taxonomie ist, ist nicht erforderlich, um sich mit Ontologien auseinander zu setzen und umgekehrt. Ontologien sind die Basis für das Semantic Web. Die Themen Ontolgien und Semantic Web können als gegenseitig weiterführende oder verwandte Themen angesehen werden, ebenso die Themen Semantic Web und OWL.

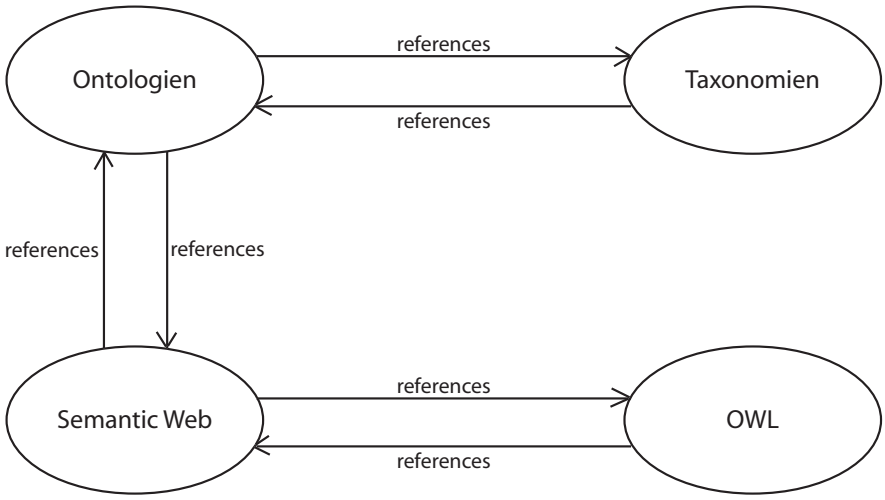

Abbildung 3.4: Beispiel für Verwendung von references

Wie Abbildung 3.5 zeigt, können neben unmittelbar falsch gesetzten Relationen Lernobjekt durch Kreisbildung logisch unkorrekt verknüpft werden.

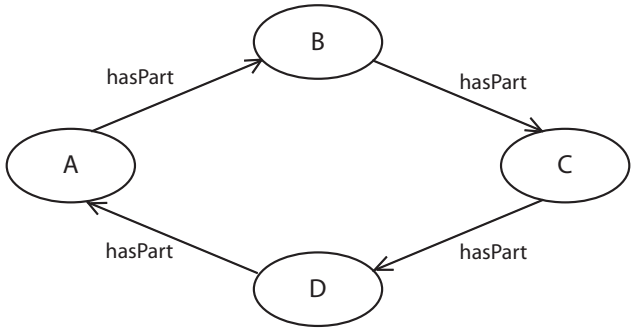

Abbildung 3.5: Kreisbildung durch unkorrekte Relationssetzung

Die Eigenschaft der Inversität kann in OWL durch das Sprachelement *owl:inverseOf* ausgedrückt werden. Die Semantik dieses Sprachelementes schränkt die Benutzung einer Property nicht ein. Stattdessen ist *owl:inverseOf* eine zusätzliche Information, die zum Schlussfolgern genutzt wird. Das bedeutet, eine logisch unkorrekte Lernobjektverknüpfung ist aus Sicht von OWL nicht falsch. OWL bietet keine Möglichkeiten, die in diesem Abschnitt aufgezeigten Beschränkungen auszudrücken. Es lassen sich jedoch Regeln definieren, die logisch unkorrekte Lernobjektverknüpfungen identifizieren (siehe Abschnitt 4.2.3).

3.5 Löschen von Relationen

Bisher sind nur Strategien zum Identifizieren von Lernobjektverknüpfungen erörtert worden. Es muss ebenfalls darüber nachgedacht werden, wie sich das bisher entstandene semantische Netz beim Verschieben und Löschen von Lernobjekten und beim Löschen von Relationen – Verknüpfungen zweier Lernobjekte – verhalten soll, weil vorhandene Lernobjektverknüpfungen eine Basis für das automatische Identifizieren bilden.

Sowohl das Verschieben als auch das Löschen eines Lernobjektes lässt sich in das Problem des Löschens einer Relation überführen. Das Verschieben eines Lernobjektes verändert die Lernobjektstruktur, die entweder durch Verschachtelung von Lernobjekten oder mittels der Relationen *hasPart/isPartOf* aufgebaut wird. (Es können Lernobjekte existieren, die in keiner Strukturbeziehung stehen. Diese können durch Verschieben nur in vorhandene Strukturen eingebunden werden.) Die Relationen *hasPart/isPartOf* sind zum einen invers zueinander und transitiv, zum anderen Glieder innerhalb einiger Schlussfolgerungsregeln. Durch Anwenden der Relationseigenschaften und der Schlussfolgerungsregeln werden neue Lernobjektverknüpfungen identifiziert. Wenn ein Lernobjekt, das mittels der Relationen *hasPart/isPartOf* in einer Strukturbeziehung steht, verschoben wird, so müssen alle Verknüpfungen, die aufgrund der wegfallenden Verknüpfung geschlussfolgert wurden, zurückgesetzt werden. Wird ein Lernobjekt verschoben, das nicht in einer Strukturbeziehung mittels *hasPart/isPartOf* steht, muss das HyLOS-System darauf nicht reagieren. Das Verschieben eines Lernobjektes wird genauso wie das Löschen einer Lernobjektverschnüpfungen mittels *hasPart/isPartOf* behandelt.

Das Löschen eines Lernobjektes zieht das Löschen aller seiner Verknüpfungen nach sich. Stellt eine der zu löschenden Verknüpfungen eine Strukturbeziehung mittels *hasPart/isPartOf* dar, müssen aus den genannten Gründen alle Verknüpfungen, die aufgrund dieser Verknüpfung geschlussfolgert wurden, zurückgesetzt werden. Sofern alle anderen Verknüpfungen des zu löschenden Lernobjektes korrekt sind, gibt es keinen Grund, die daraus geschlussfolgerten Verknüpfungen zu löschen, da diese immer noch korrekt sind. Im Gegensatz zu dem Relationspaar *hasPart/isPartOf*, das nur eine Strukturbeziehung darstellt, treffen alle anderen Relationen Aussagen über die Art und Weise der inhaltlichen Verwandtschaft zweier Lernobjekte. Wenn eine inhaltliche Beziehung zwischen zwei Lernobjekten A und B richtig ist und daraus neue Beziehungen hergeleitet werden, dann bleiben diese neue Beziehungen richtig, auch wenn Lernobjekt A oder B im Nachhinein gelöscht wird.

Beim Löschen eines Lernobjektes ist demzufolge wichtig, ob seine Verknüpfungen – alle außer denen mittels *hasPart/isPartOf* – korrekt sind oder nicht. Im Gegensatz zum ersten Fall müssen im zweiten alle aufgrund von unkorrekten Verknüpfungen geschlussfolgerten Verknüpfungen zurückgesetzt werden. Beim Löschen einer Relation wird genauso verfahren.

Da Lernobjektverknüpfungen auch anhand anderer Daten identifiziert werden, müssen nach dem Löschen dieser Daten ebenfalls alle daraus geschlussfolgerten Lernobjektverknüpfungen gelöscht werden.

Kapitel 4

Konzeption

4.1 Einsatz der Verknüpfungsengine

Es gibt mehrere Möglichkeiten, die Verknüpfungsengine in einem Lernobjekt-Managementsystem funktionell einzugliedern. Zum einen kann sie in das Autorenwerkzeug integriert, zum anderen als eigenständige Applikation mit oder ohne Nutzerinteraktion implementiert werden.

Eine Integration in das Autorenwerkzeug kann auf zwei Arten erfolgen. Die Verknüpfungsengine kann ein Bestandteil des Autorenwerkzeuges sein oder als Server fungieren und von dem Autorenwerkzeug bei Aktionen von Belang benachrichtigt werden.

Eine eigenständige interaktionslose Applikation kann sowohl manuell als auch anderweitig, zum Beispiel vom Betriebssystem, gestartet werden. Eine derart aktivierte Verknüpfungsengine verarbeitet entweder den gesamten Datenbestand oder nur die nach dem letzten Durchlauf hinzugefügten oder gelöschten Daten. Neben dieser Einsatzmöglichkeit kann die Verknüpfungsengine ebenso als Server fungieren, wobei die Datenbank, in der Lernobjekte gespeichert sind, oder die Middleware, die die Datenzugriffslogik implementiert, sie bei relevanten Datenbankaktionen benachrichtigt. Das derzeitig im HyLOS-System eingesetzte Datenbanksystem Sybase kennt eine Extended Stored Procedure, die es ermöglicht, ein Kommando im laufenden Betriebssystem abzusetzen. Ein Datenbanktrigger kann diese Extended Stored Procedure aufrufen.

Eine eigenständige Applikation mit Nutzerinteraktion kann zum einen anhand des Client-Server-Modells implementiert werden. Dabei kommuniziert der Client mit der Verknüpfungsengine und zeigt deren Ergebnisse auf einer graphischen Benutzeroberfläche an. Zum anderen kann die Verknüpfungsengine Bestandteil einer graphischen Benutzeroberfläche sein. Letzteres hat zur Folge, dass pro Client eine Instanz der Verknüpfungsengine gestartet wird. Gleichzeitig agierende Verknüpfungsengines, die dieselben Aktionen auf denselben Daten ausführen, sind nicht gewinnbringend. Diese Einsatzmöglichkeit wird nicht weiter betrachtet.

Alle Möglichkeiten weisen Vor- und Nachteile auf. Die folgenden vier Unterabschnitte erläutern diese.

4.1.1 Integration in das Autorenwerkzeug

Vorteile einer in das Autorenwerkzeug integrierten Verknüpfungsengine sind:

- Die Verknüpfungsengine kann direkt auf relevante Aktionen innerhalb des Autorenwerkzeuges reagieren. Logisch unkorrekte und aufgrund heuristischer Verfahren identifizierte Lernobjektverknüpfungen können einem Autor sofort präsentiert werden und müssen deshalb nicht zur nachträglichen Verarbeitung gespeichert werden. Außerdem muss das Autorenwerkzeug nicht derart erweitert werden, dass es relevante Löschvorgänge speichert.

Nachteile einer in das Autorenwerkzeug integrierten Verknüpfungsengine sind:

- Direktes Reagieren auf Aktionen innerhalb des Autorenwerkzeuges blockiert dieses, sofern der Verknüpfungsmechanismus nicht nebenläufig arbeitet, da das automatische Identifizieren ein rechenintensiver Prozess ist. Bei Nebenläufigkeit muss ein vorzeitiges Beenden des Autorenwerkzeuges verhindert werden.

- Die Verknüpfungsengine muss alle relevanten Daten aus der Datenbank lesen, um diese verarbeiten zu können. Wenn mehrere Autorenwerkzeuge gleichzeitig die Daten innerhalb der Datenbank ändern und die Verknüpfungsengine ein Bestandteil des Autorenwerkzeuges ist, werden mehrere Instanzen der Verknüpfungsengine gestartet. Trotzdem die Verknüpfungsengine nur auf die auslösende Aktion reagiert, kann es zu inkonsistenten Zuständen kommen. Das Problem besteht nicht beim Hinzufügen eines relevanten Datums, sondern beim Löschen. Wenn während des Ausführens einer Verknüpfungsengine ein Datum gelöscht wird, welches diese zum Schlussfolgern verwendet, können die aufgrund des gelöschten Datums geschlussfolgerten Lernobjektverknüpfungen nicht mehr rückgängig gemacht werden, da Informationen über Löschvorgänge nicht gespeichert werden. Die Verknüpfungsengine muss benachrichtigt werden, wenn sich die Daten, die sie zum Schlussfolgern nutzt, durch parallel agierende Autorenwerkzeuge ändern.

4.1.2 Interaktionslose zu einem definierten Zeitpunkt gestartete Applikation

Vorteile einer interaktionslosen zu einem definierten Zeitpunkt gestarteten Applikation sind:

- Eine Synchronisation zwischen Datenbank und Verknüpfungsengine muss nicht vorgenommen werden. Es wird nur auf den zum Ausführungszeitpunk in der Datenbank vorhandenen Daten gearbeitet. Der Ausführungszeitpunkt ist gleichgültig. Die Verknüpfungsengine verarbeitet alle Daten, die während oder nach dem Ausführen hinzugefügt oder gelöscht werden, beim nächsten Durchlauf.

- Die Verknüpfungsengine arbeitet selbstheilend. Werden während eines Durchlaufs relevante Daten hinzugefügt, so identifiziert die Verknüpfungsengine die daraus resultierenden Verknüpfungen beim nächsten Durchlauf. Selbst wenn Daten, die während eines

Durchlaufs gelöscht werden, zum Identifizieren genutzt werden, kann dieser inkonsistente Zustand behoben werden. Da relevante Löschvorgänge und alle Schlussfolgerungsschritte gespeichert werden, können alle aufgrund gelöschter Daten geschlussfolgerten Lernobjektverknüpfungen beim nächsten Durchlauf zurückgesetzt werden. Ebenso werden Zustände korrigiert, die entstehen, wenn aufgrund einer Sperre Daten nicht gelesen oder identifizierte Lernobjektverknüpfungen nicht geschrieben werden können.

- Die Dauer der Ausführung ist gleichgültig, da keine Anwendung auf das Beenden der Verknüpfungsengine wartet.

Nachteile einer interaktionslosen zu einem definierten Zeitpunkt gestarteten Applikation sind:

- Es wird nicht direkt auf Datenänderungen reagiert. Das Hauptproblem besteht beim Löschen von Daten, da im Nachhinein keine Informationen über erfolgte Löschvorgänge zu erhalten sind. Das Autorenwerkzeug muss dahingehend erweitert werden, dass alle relevanten Löschvorgänge festgehalten werden. Im HyLOS-System besteht dieses Problem beim Hinzufügen von relevanten Daten nicht, da alle Lernobjekte Properties, die Erstellungs- beziehungsweise Änderungszeitpunkt angeben, besitzen.

- Aufgrund fehlender Interaktionsmöglichkeiten kann ein Autor nicht direkt auf manuell gesetzte Lernobjektverknüpfungen, die logisch unkorrekt sind, reagieren. Die Verknüpfungsengine kann eine logisch unkorrekte Lernobjektverknüpfung identifizieren, jedoch nicht entscheiden, welche der beteiligten Verknüpfungen falsch ist. Ebenso kann ein Autor nicht direkt auf Lernobjektverknüpfungen, die durch heuristische Verfahren identifiziert wurden, reagieren.

4.1.3 Interaktionslose datenbankgesteuerte Applikation

Vorteile einer interaktionslosen datenbankgesteuerten Applikation sind:

- Eine Synchronisation zwischen Datenbank und Verknüpfungsengine muss nicht vorgenommen werden. Die Verknüpfungsengine fungiert als Server und arbeitet alle Nachrichten über relevante Datenbankaktion, die möglicherweise von parallel agierenden Autorenwerkzeugen ausgelöst wurden, nacheinander ab.

- Auf das Löschen eines relevanten Datums kann direkt reagiert werden, dadurch muss das Autorenwerkzeug relevante Löschvorgänge nicht speichern.

- Die Dauer der Ausführungszeit ist gleichgültig.

Nachteile einer interaktionslosen datenbankgesteuerten Applikation sind:

- Logisch unkorrekte und durch heuristische Verfahren identifizierte Lernobjektverknüpfungen können nicht sofort einem Autor präsentiert werden.

4.1.4 Eigenständige Applikation mit Nutzerinteraktion

Vorteile einer eigenständigen Applikation mit Nutzerinteraktion sind:

- Logisch unkorrekte und durch heuristische Verfahren identifizierte Lernobjektverknüpfungen können sofort einem Autor präsentiert werden.
- Eine Synchronisation zwischen Datenbank und Verknüpfungsengine muss nicht vorgenommen werden.
- Die Verknüpfungsengine arbeitet selbstheilend.

Nachteile einer eigenständigen Applikation mit Nutzerinteraktion sind:

- Die Dauer der Ausführung ist nicht gleichgültig. Der Autor muss auf das Ende der Verknüpfungsengine warten, um auf deren Ergebnisse zu reagieren oder die Anwendung zu beenden.
- Das Autorenwerkzeug muss derart erweitert werden, dass es relevante Löschvorgänge festhält, um von der Verknüpfungsengine verarbeitet zu werden.

4.1.5 Lernobjektverknüpfungen zwischen Datenbanken

Eine weitere interessante Einsatzmöglichkeit ist das Setzen von Relationen über Datenbankgrenzen hinweg. Es können ausschließlich Datenbanken genutzt werden, die einen Zugriff auf alle relevanten Metadaten eines Lernobjektes bieten. Die Verknüpfungsengine spricht anstatt einer solchen Datenbank mehrere Datenbanken an, um die Informationen, die zum Schließen notwendig sind, zu lesen.

Die Implementierung der Verknüpfungsengine muss flexibel in Bezug auf die genutzten Datenbanken sein. Das kann derart realisiert werden, dass ein Datenbank-Interface notwendige Methoden definiert und dieses Interface für jede Datenbank implementiert werden muss. Zusätzlich muss ein Interface oder eine Klasse definiert werden, welches/welche als Rückgabewert der Datenbank-Interface-Methoden dient. Damit ist der Zugriff auf die Rückgabewerte einheitlich für alle Datenbanken (siehe Abschnitt 4.4).

4.2 Funktionen der Verknüpfungsengine

4.2.1 Lernobjektverknüpfungen identifizieren und setzen

Bis auf die heuristischen Verfahren führen alle in Abschnitt 3.3 entwickelten Strategien zu logisch korrekten Lernobjektverknüpfungen und können ohne Bestätigung eines Autors gesetzt werden.

Abschnitt 2.2.1 stellte Ontologien als eine Möglichkeit, Daten für Maschinen verarbeitbar zu präsentieren, vor. Sowohl die Sprachelemente von RDFS als auch die Sprachelemente von OWL enthalten implizit Regeln, die angewandt werden können, um neue Daten – Lernobjektverknüpfungen – zu erarbeiten. Deshalb werden fast alle von der Verknüpfungsengine zu verarbeitenden Daten mittels RDF/RDFS oder OWL beschrieben.

Um Lernobjektverknüpfungen aus Taxonomien abzulesen, müssen die Taxonomien und die Klassifikationsinformationen eines Lernobjektes verarbeitet werden. Die Taxonomien werden mit Hilfe von SKOS (siehe Abschnitt 2.2.1 Seite 9) beschrieben. Das SKOS-Vokabular ist in RDF-RDFS-Properties und -Klassen organisiert. Die Metamodellierungsmöglichkeiten von RDF/RDFS (siehe Abschnitt 2.2.2 Seite 13) werden nicht genutzt, somit ist SKOS in endlicher Zeit entscheidbar. Die Klassifikationsinformationen eines Lernobjektes werden innerhalb einer OWL-Lite-Ontologie als Tripel dargestellt. Jedes Tripel steht für eine Aussage der Form: Lernobjekt A ist dem Taxonomieknoten X zugeordnet. Die Ausdruckskraft von OWL Lite ist ausreichend. OWL Lite ist in endlicher Zeit entscheidbar. Ein Schlussfolgerer kann diese Daten zur Identifikation neuer Lernobjektverknüpfungen nutzen, indem er die in Abschnitt 3.3.1.1 aufgestellten Regeln anwendet. Anstatt einen Schlussfolgerer zu benutzen, können ausgehend von einem Lernobjekt alle Lernobjekte, die anhand dieser Regeln verknüpft werden, mit Hilfe einer entsprechenden Datenbankanfrage identifiziert werden. Das sind beispielsweise alle Lernobjekte, die in demselben Taxonomiezweig eingeordnet, jedoch einem tiefer angesiedelten Taxonomieknoten zugeordnet sind.

Das Datenbank-Interface, welches für jede Datenbank, deren Lernobjekte verknüpft werden sollen, implementiert werden muss, stellt Methoden zur Verfügung, mittels derer die Lernobjektstruktur, in der ein Lernobjekt eingebunden ist, untersucht werden kann.

Um die erweiterten Relationseigenschaften zu verarbeiten, werden zum einen die Relationen mit ihren Eigenschaften als auch die Lernobjektmetadaten, die Informationen über die Beziehungen eines Lernobjektes enthalten, in einer OWL-Lite-Ontologie abgebildet. Wegen der Relationseigenschaften – Inversität, Transitivität und Symmetrie –, die eine Grundlage für das Identifizieren neuer Lernobjektverknüpfungen bilden, muss OWL als Ontologiesprache eingesetzt werden. Jede Information über eine Beziehung eines Lernobjektes ist ein Tripel der Form: Lernobjekt A steht in Beziehung X zu Lernobjekt B.

Die identifizierten Schlussfolgerungsregeln arbeiten ebenfalls auf vorhandenen Lernobjektverknüpfungen. Zur Verarbeitung der Ontologien und der Schlussfolgerungsregeln wird eine Schlussfolgerungsmaschine benötigt, die neben RDF/RDFS und OWL eigene Regeln verarbeiten kann.

4.2.2 Vermutungen über Lernobjektverknüpfungen aufstellen

Abschnitt 3.3.1.3 identifizierte LOM-Datenelemente, die bei gleich oder gleichwertigen Angaben Hinweise auf weitere mögliche Lernobjektverknüpfungen geben.

Eine Möglichkeit, diese heuristischen Verfahren umzusetzen, ist die folgende: Die im vorherigen Abschnitt erwähnte Ontologie der Lernobjektmetadaten wird um die Angaben der relevanten LOM-Datenelemente erweitert. Zusätzlich werden Regeln, die diese verarbeiten, erstellt. Abschnitt 3.3.1.3 benannte unter anderem LOM-Datenelemente, die auf inhaltliche oder pädagogische Gleichwertigkeit schließen lassen. Die Regel zur Identifikation zweier inhaltlich gleichwertiger Lernobjekte besagt: Wenn zwei Lernobjekte dieselbe Sprache besitzen und ihre Schlüsselwörter mehrheitlich übereinstimmen und sie derselben Zeit, Kultur, Geographie oder Region zugeordnet sind und sie zu denselben Lernobjekten mittels derselben benannten Relation in Beziehung stehen und sie anhand desselben Klassifikationsschemas unterhalb desselben Taxonomieknotens eingeordnet sind, dann sind sie wahrscheinlich inhaltlich gleichwertig. Diese Regel kann um die Bedingungen, dass beide Lernobjekte gleichwertigen Lernkontexten, Altersgruppen und Schwierigkeitsgraden zugeordnet sind und dasselbe Format besitzen, erweitert werden, um auf die Relation *isFormatOf* zu schließen. Die Lernobjektontologie wird jedoch durch diese Vorgehensweise aufgebläht. Das wirkt sich nachteilig auf die Ausführungszeit der Schlussfolgerungsmaschine aus. Ein weiterer Mangel ist, dass die Schlussfolgerungsmaschine viele Daten, die zu keinem Ergebnis führen, verarbeiten muss.

Sinnvoller erscheint, bezogen auf ein zu verknüpfendes Lernobjekt Datenbankanfragen zu formulieren und somit nur Lernobjekte, deren Metadaten die Anforderungen erfüllen, aus der Datenbank zu lesen. Die durch diese sehr komplexen Datenbankanfragen zurückgelieferten Lernobjekte müssen nicht mehr von einer Schlussfolgerungsmaschine verarbeitet werden.

Abzuwägen ist zwischen Laufzeitflexibilität und -komplexität: Datenbankanfragen können nur mit dem Wissen über das jeweilige Datenbankschema formuliert werden. Im Gegensatz dazu ist die Ontologie der Lernobjektmetadaten unabhängig von der Organisation der Daten in einer Datenbank; ihre Verarbeitung durch eine geeignete Schlussfolgerungsmaschine jedoch komplexer. Trotzdem darf die Ontologie nicht losgelöst von dem Problem der Datenorganisation betrachtet werden, denn für ihre Erstellung werden Datenbanken angesprochen. Das Datenbank-Interface bildet dafür eine einheitliche Schnittstelle, sodass die implementierenden Klassen einheitliche in datenbankspezifische Anfragen übersetzen müssen.

Neben ausgewählten LOM-Datenelementen werden auch die Lernobjektstrukturen untersucht. Die Relation *isAlternativeTo* verlangt strukturelle Gleichwertigkeit. Zum Vergleich der Lernobjektstrukturen zweier Lernobjekte definiert das Datenbank-Interface Methoden.

4.2.3 Logisch unkorrekte Lernobjektverknüpfungen identifizieren

Die Verknüpfungsengine muss überprüfen können, ob eine Lernobjektverknüpfung logisch korrekt ist. Mindestens zwei Lernobjektverknüpfungen sind die Ursache einer logisch unkorrekten Lernobjektverknüpfung (siehe Abschnitt 3.4). Welche der beteiligten Verknüpfungen falsch ist, kann die Verknüpfungsengine nicht entscheiden. Das muss von einem Autor getan werden.

Um logisch unkorrekte Lernobjektverknüpfungen zu identifizieren, kann folgendermaßen vorgegangen werden: Für jedes Paar inverser Relationen, mittels dessen sich Lernobjekte logisch unkorrekt verknüpfen lassen, wird eine zusätzliche Relation definiert. Für das Paar *hasPart/isPartOf* ist das zum Beispiel die Relation *incorrectIsPartOf*. Ausserdem wird für jede Relation, die eine logisch unkorrekte Lernobjektverknüpfung bezeichnet, eine Regel zu deren Identifikation erstellt. Diese Regeln sind ähnlich den Schlussfolgerungsregeln. Die Regel zur Identifikation der Relation *incorrectIsPartOf* ist:

$$\text{A } \textit{isPartOf} \text{ B} \wedge \text{B } \textit{isPartOf} \text{ A} \rightarrow \text{A } \textit{incorrectIsPartOf} \text{ B}$$

Die identifizierten logisch unkorrekten Lernobjektverknüpfungen werden je nach Einsatz der Verknüpfungsengine sofort einem Autor präsentiert oder zur nachträglichen Bearbeitung gespeichert.

4.2.4 Schlussfolgerungsketten merken

Damit nach dem Löschen eines Datums, das zum Identifizieren von Lernobjektverknüpfungen relevant ist, alle daraus geschlussfolgerten Lernobjektverknüpfungen zurückgesetzt werden können, muss zusätzlich zum Identifizieren und Setzen einer Lernobjektverknüpfung deren Schlussfolgerungskette gespeichert werden. Eine Schlussfolgerungskette wird in ihre einzelnen Schritte aufgespalten und in Paaren – Ausgangsdatum + resultierende Lernobjektverknüpfung – gespeichert. Tabelle 4.1 enthält die Schlussfolgerungsschritte, die zu der in Abbildung 4.1 dargestellten Situation, führten.

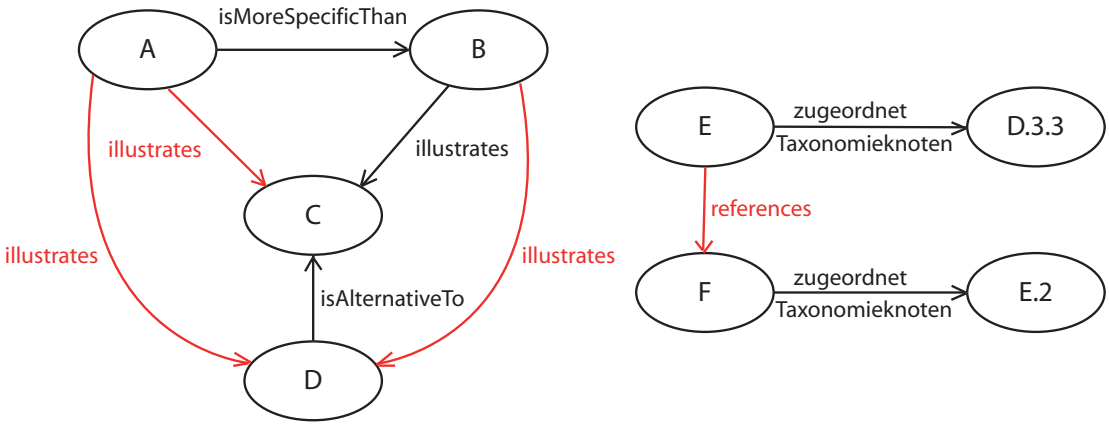

Abbildung 4.1: Ausgangsdaten und resultierende Lernobjektverknüpfungen

Prototypisch werden die Schlussfolgerungsketten in einer Textdatei gespeichert. Jedes Ausgangsdatum und jede Lernobjektverknüpfung kann als RDF-Tripel dargestellt werden. Das Lernobjekt, über das eine Aussage getroffen wird, bildet das Subjekt des RDF-Tripels. Das Prädikat des RDF-Tripels bezeichnet zum Bespiel eine benannte Relation oder den Sachverhalt, dass ein

Ausgangsdaten	resultierende Lernobjektverknüpfungen
A isMoreSpecificThan B	A illustrates C
B illustrates C	A illustrates C, B illustrates D
D isAlternativeTo C	B illustrates D, A illustrates D
A illustrates C	A illustrates D
E zugeordnet Taxonomieknoten D.3.3	E references F
F zugeordnet Taxonomieknoten E.2	E references F

Tabelle 4.1: in einzelne Schritte zerlegte Schlussfolgerungsketten

Lernobjekt einem Taxonomieknoten zugeordnet ist. Das Objekt des RDF-Tripels kann demzufolge ein Lernobjekt oder ein Taxonomieknoten, aber auch in Abhängigkeit vom Prädikat ein anderes Datum sein. Tabelle 4.1 enthält Ausgangsdaten und Lernobjektverknüpfungen als Tripel.

In jeder Zeile der Textdatei steht maximal ein Tripel. Ausgangsdatum und resultierende Lernobjektverknüpfungen bilden dergestalt einen Block, dass die erste Zeile des Blocks das Ausgangsdatum enthält und alle folgenden Zeilen daraus resultierende Lernobjektverknüpfungen sind. Blöcke werden durch mindestens eine Leerzeile voneinander getrennt.

Ein Tripel in Textform besteht aus eindeutigen Bezeichnern für Subjekt, Prädikat und Objekt, welche durch ein Leerzeichen voneinander getrennt sind. Die Textdatei ist nicht geeignet, um von einem Menschen gelesen zu werden. Dazu ist sie jedoch nicht gedacht. In der Textdatei werden Schlussfolgerungsketten abgelegt, um von der Verknüpfungsengine verarbeitet zu werden.

Langfristig sollen die Schlussfolgerungsschritte in der Datenbank, die die Lernobjektdaten enthält, abgelegt werden. Dem HyLOS-System liegt eine relationale Datenbank zugrunde. Eine Lernobjektverknüpfung kennt die Art der Verknüpfung (benannte Relation) und das Ziellernobjekt. Ein Lernobjekt kennt alle seine Lernobjektverknüpfungen. Das Datenbankschema kann um eine Tabelle erweitert werden, deren Datensätze aus den Primärschlüsseln der Ausgangs- und der resultierenden Lernobjektverknüpfung bestehen. Ein Problem dieser Vorgehensweise ist, dass die Fremdschlüsselbedingung das Löschen einer Lernobjektverknüpfung verbietet, auch wenn auf sie nur als resultierende Lernobjektverknüpfung verwiesen wird.

Eine spätere Lösung muss berücksichtigen, dass neben Lernobjektverknüpfungen auch andere Daten – zum Beispiel die Zuordnung eines Lernobjektes zu einem Taxonomieknoten – ein Ausgangsdatum sein können.

4.3 Skalierbarkeit der Verknüpfungsengine

Um Lernobjektverknüpfungen zu identifizieren, werden hauptsächlich Graphen durchsucht, wobei die Ontologie der Lernobjektmetadaten oder der Relationen den Graphen bildet. Suchoperationen auf einem Graphen sind ressourcen- und rechenintensiv.

Die Verknüpfungsengine sollte so wenig Ressourcen wie möglich verbrauchen und so schnell wie möglich arbeiten. Die Vorgehensweise, alle Lernobjekte – genauer die Lernobjektmetadaten, die zum automatischen Identifizieren von Lernobjektverknüpfungen genutzt werden – aus der Datenbank zu lesen und die daraus erstellte Ontologie für alle zu verknüpfenden Lernobjekte zu verwenden, führt zu einem unnötig hohen Ressourcenverbrauch und unnötig langen Antwortzeiten. Die Komblexität des Identifizierens sollte nicht von der Gesamtzahl der Lernobjekte abhängen.

Eine gute Skalierbarkeit der Verknüpfungsengine kann erreicht werden, indem zu verknüpfende Lernobjekte nacheinander verarbeitet werden. Eine Verknüpfungsbedingung kann sein, dass ein Lernobjekt nach einem festgelegten Zeitpunkt geändert oder erstellt wurde. Die Ontologie der Lernobjektmetadaten wird bezogen auf das betrachtete Lernobjekt erstellt; sie wird nur aus den Lernobjekten aufgebaut, die aufgrund der in Abschnitt 3.3 entwickelten Strategien mit dem betrachteten Lernobjekt in Beziehung stehen können. Dazu gehören alle Lernobjekte,

- die demselben Taxonomiezweig wie das Startlernobjekt zugeordnet sind,
- deren Metadaten das Anwenden eines heuristischen Verfahrens ermöglichen (falls Heuristiken mittels Regeln abgebildet werden),
- die direkt oder über mehrere Schritte mit dem betrachteten Lernobjekt mittels einer benannten Relation verknüpft sind.

Die Menge der Kind- und Elternlernobjekte ist nicht aufgeführt, da sie nicht mittels einer externen Schlussfolgerungsmaschine verarbeitet wird. Aus der reduzierten Menge wird eine Ontologie aufgebaut und diese der externen Schlussfolgerungsmaschine übergeben.

Zu Beginn des Einsatzes der Verknüpfungsengine wird diese Vorgehensweise erfolgreich sein und die reduzierte Menge im Vergleich zu dem gesamten Datenbankbestand klein sein. Ist jedoch ein stark verwobenes Netz vorhanden, können reduzierte Menge und Datenbankbestand fast identisch sein. Die Komplexität des Identifizierens hängt von dem betrachteten Lernobjekt ab. Je stärker ein Lernobjekt verknüpft ist oder je mehr andere Lernobjekte demselben Taxonomiezweig zugeordnet sind, desto umfangreicher ist die reduzierte Menge. Da die benannten Relationen jedoch hauptsächlich Lernobjekte verknüpfen, die im weitesten Sinne demselben Themengebiet entstammen, werden sich innerhalb des gesamten Lernobjektbestandes Themeninseln bilden. Dieser Umstand spricht für die aufgezeigte Vorgehensweise.

Die Menge ausgewählter Lernobjekte reicht nicht aus. Ausgehend von allen Lernobjekten, die zu einer reduzierten Menge gehören, können reduzierte Mengen erstellt werden – beispielsweise die Menge aller direkt oder über mehrere Schritte mit einem Lernobjekt, welches demselben Taxonomiezweig wie das betrachtete Lernobjekt zugeordnet ist, verknüpfte Lernobjekte.

Damit alle möglichen Lernobjektverknüpfungen zu einem betrachteten Lernobjekt ohne unnötig hohen Ressourcenverbrauch gefunden werden, können die Untermengen einer reduzierten Menge nacheinander verarbeitet werden. Zuerst wird das betrachtete Lernobjekt mit seinen

Kind- und Elternlernobjekten mittels *hasPart* beziehungsweise *isPartOf* verknüpft. Anschließend werden die Lernobjekte desselben Taxonomiezweiges verarbeitet und die heuristischen Verfahren angewandt. Zuletzt wird die Menge der mit dem betrachteten Lernobjekt verknüpften Lernobjekte untersucht; sie kann aufgrund der vorherigen Schritte erweitert worden sein. Ausgehend von dem betrachteten Lernobjekt A werden nur direkt verknüpfte Lernobjekte B_n und deren direkt verknüpfte Lernobjekte C_n eingelesen und der Schlussfolgerungsmaschine übergeben. Wird durch das Anwenden von Schlussfolgerungsregeln oder dem Verarbeiten der Relationseigenschaften eine neue Lernobjektverknüpfung für A identifiziert, werden alle direkt verknüpften Lernobjekte des jeweiligen Lernobjektes C_n eingelesen und überprüft, ob weitere neue Lernobjektverknüpfungen für A identifiziert werden können. Dieser Vorgang kann solange wiederholt werden, bis keine Lernobjektverknüpfungen mehr identifiziert werden. Für alle Schlussfolgerungsregeln (siehe Anhang A) kann diese Vorgehensweise angewandt werden.

4.4 Aufbau der Verknüpfungsengine

Die Verknüpfungsengine wird als Java-Anwendung implementiert. Die Implementierung muss flexibel sein. Alle möglichen Einsatzszenarien und Strategien der Skalierung sollen ohne aufwendige Änderungen des Quellkodes realisiert werden können. Der Persistenzmechanismus zum Speichern der Schlussfolgerungsketten soll leicht austauschbar sein. Dasselbe gilt für das Speichern der Informationen über logisch unkorrekte Lernobjektverknüpfungen oder der Lernobjektverknüpfungen, die aufgrund heuristischer Verfahren identifiziert wurden, falls wegen fehlender Nutzerinteraktion ein Speichern notwendig ist. Aufgrund der genannten Anforderungen wurden die folgenden Klassen und Interfaces definiert. Abbildung 4.2 zeigt das zugehörende Klassendiagramm.

4.4.1 Das Interface *Database*

Das Interface *Database* definiert die einheitliche Schnittstelle, um Lernobjektdaten zu lesen und zu schreiben. Für jedes System, das angesprochen werden soll, muss es implementiert werden. Auf jedes einzelne System bezogen ist das Interface *Database* eine Fassade [GHJV01], die eine einzelne Schnittstelle zur Funktionalität des Systems bietet. Klienten kommunizieren mit der Fassade; diese leitet die Anfragen an die zuständigen Subsystem-Objekte weiter. Das Interface *Database* definiert Methoden zum:

- Lesen von Lernobjekten aufgrund verschiedener Kriterien
- Lesen aller Verknüpfungen eines Lernobjektes
- Lesen aller Taxonomieknoten, unterhalb derer ein Lernobjekt eingeordnet ist
- Lesen aller Kind- oder Elternlernobjekte eines Lernobjektes

- Lesen des Wertes/der Werte einer Property eines Lernobjektes
- Speichern von Lernobjektverknüpfungen
- Löschen von Lernobjektverknüpfungen

4.4.2 Die Klasse *Identifier*

Eindeutige Bezeichner (zum Beispiel Primärschlüssel in einer Datenbank) können sich im Datentyp unterscheiden. Die Klasse *Identifier* kapselt einen eindeutigen Bezeichner. Sie besitzt Felder unterschiedlichen Datentyps, um den Wert des eindeutigen Bezeichners aufzunehmen, und ein Feld, das den verwendeten Datentyp kennzeichnet. Diese Klasse dient als Parameter und als Rückgabewert von Methoden des Interfaces *Database*. Werden Lernobjekte aus einer Datenbank gelesen, so liefert die jeweilige Methode eine Liste von *Identifier*-Objekten, wobei jedes *Identifier*-Objekt für genau ein Lernobjekt steht. Sollen alle Verknüpfungen eines Lernobjektes gelesen werden, wird der Methode ein *Identifier*-Objekt als Parameter übergeben, um das Lernobjekt zu kennzeichnen.

4.4.3 Die Klasse *Relation*

Die Klasse *Relation* kapselt Informationen über eine Lernobjektverknüpfung. Sie kennt die eindeutigen Bezeichner des Quell- und des Ziellernobjektes und die benannte Relation. Die Methode des Interfaces *Database*, die alle Lernobjektverknüpfungen eines Lernobjektes liest, liefert eine Liste von *Relation*-Objekten zurück.

4.4.4 Die Klasse *Classification*

Die Klasse *Classification* kapselt Informationen über eine Zuordnung eines Lernobjektes zu einem Taxonomieknoten. Sie kennt die eindeutigen Bezeichner des Lernobjektes und des Taxonomieknotens. Die Methode des Interfaces *Database*, die alle Taxonomieknoten, unterhalb derer ein Lernobjekt eingeordnet ist, liest, liefert eine Liste von *Classification*-Objekten zurück.

4.4.5 Die Klasse *Property*

Die Klasse *Property* kapselt Information über den Wert einer Property eines Lernobjektes. Sie kennt den eindeutigen Bezeichner des Lernobjektes, den Namen der Property und deren Wert. Die Methode des Interfaces *Database*, die den Wert/die Werte einer Property eines Lernobjektes liest, liefert eine Liste von *Property*-Objekten zurück.

4.4.6 Das Interface *Derivation*

Das Interface *Derivation* definiert die einheitliche Schnittstelle für das Lesen und Speichern von Schlussfolgerungsketten. Eine Klasse muss das Interface in Abhängigkeit des verwendeten Persistenzmechanismus implementieren. Das Interface *Derivation* definiert Methoden zum:

- Speichern eines Schlussfolgerungsschrittes
- Lesen aller aufgrund eines Datums geschlussfolgerten Lernobjektverknüpfungen
- Löschen einer geschlussfolgerten Lernobjektverknüpfung aus den gespeicherten Schlussfolgerungsketten

4.4.7 Das Interface *IncorrectRelationship*

Das Interface *IncorrectRelationship* definiert eine einheitliche Schnittstelle für das Lesen und Speichern logisch unkorrekter Lernobjektverknüpfungen. Falls aufgrund fehlender Nutzerinteraktion ein Zwischenspeichern zur nachträglichen Bearbeitung notwendig ist, muss eine Klasse das Interface in Abhängigkeit des verwendeten Persistenzmechanismus implementieren. Das Interface *IncorrectRelationship* definiert Methoden zum:

- Speichern einer logisch unkorrekten Lernobjektverknüpfung und ihrer ursächlichen Lernobjektverknüpfungen
- Löschen einer ursächlichen Lernobjektverknüpfung aus den gespeicherten Informationen über logisch unkorrekte Lernobjektverknüpfungen
- Lesen aller logisch unkorrekten Lernobjektverknüpfungen
- Lesen aller ursächlichen Lernobjektverknüpfungen einer logisch unkorrekten Lernobjektverknüpfung

4.4.8 Das Interface *SupposedRelationship*

Das Interface *SupposedRelationship* bildet die Schnittstelle für das Lesen und Speichern von Lernobjektverknüpfungen, die aufgrund heuristischer Verfahren identifiziert wurden. Falls aufgrund fehlender Nutzerinteraktion ein Zwischenspeichern zur nachträglichen Bearbeitung notwendig ist, muss eine Klasse das Interface in Abhängigkeit des verwendeten Persistenzmechanismus implementieren. Das Interface *SupposedRelationship* definiert Methoden zum:

- Speichern einer vermuteten Lernobjektverknüpfung und ihrer Ausgangsdaten
- Löschen einer vermuteten Lernobjektverknüpfung
- Lesen aller vermuteten Lernobjektverknüpfungen
- Lesen aller Ausgangsdaten einer vermuteten Lernobjektverknüpfung

4.4.9 Das Interface *RelationshipFinder*

Das Interface *RelationshipFinder* bildet die Schnittstelle für das Identifizieren von Lernobjektverknüpfungen. Es definiert Methoden, die alle vor dem Schlussfolgern notwendigen Aktionen durchführen oder Lernobjektverknüpfungen identifizieren und speichern. Die Verknüpfungsengine benötigt sowohl eine Implementierungen bezogen auf die verwendete Schlussfolgerungsmaschine als auch Implementierungen, die die Strategien, die nicht mittels einer externen Schlussfolgerungsmaschine realisiert werden, umsetzen.

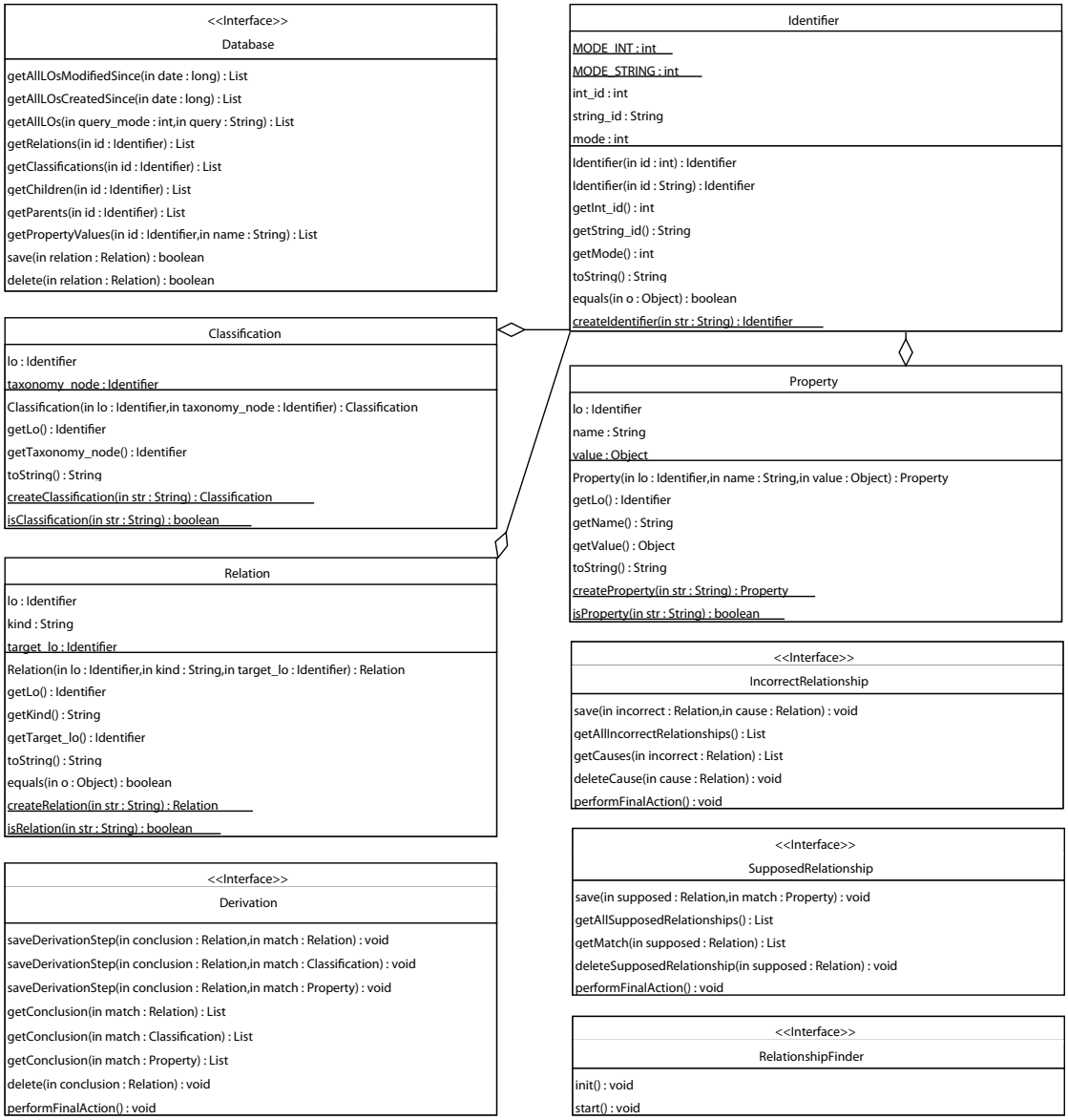

Abbildung 4.2: Klassendiagramm der Verknüpfungsengine-API

4.5 Jena – Ein Semantic-Web-Toolkit

Jena [JEN05] wird als Schlussfolgerungsmaschine innerhalb der Verknüpfungsengine eingesetzt. Jena ist ein in Java implementiertes Toolkit zum Erstellen von Applikationen für das Semantic Web, das von der Semantic Web Research Group der HP Labs entwickelt wurde.

Jena kann sowohl RDF als auch OWL verarbeiten. Das heißt konkret, dass mittels der Jena-API RDF- und OWL-Graphen erzeugt, manipuliert und gespeichert werden können. Zum einen stellt Jena Klassen zur Verfügung, um RDF- und OWL-Graphen zu erstellen; zum anderen können serialisierte RDF- und OWL-Graphen eingelesen werden. Jena versteht folgende Serialisierungssyntaxen: RDF/XML, N-Triple, N3. Zusätzlich können Datenbanken zur dauerhaften Speicherung genutzt werden. Zur Zeit werden MySQL, Oracle und PostgreSQL unterstützt.

Die Ontologie-API innerhalb Jenas ist sprachenneutral. Neben OWL Lite, OWL DL und OWL Full können DAML+OIL und RDFS als Ontologiesprachen verwandt werden.

Außerdem bietet Jena eine Abfragesprache für RDF-Graphen in Jena, genannt RDQL. RDQL wurde ebenfalls an den HP Labs entwickelt und im Januar 2004 eine W3C Member Submission[1] dazu eingereicht. RDQL ist datenorientiert; Anfragen arbeiten nur auf den Daten, die zum Zeitpunkt der Anfrage in einem RDF-Graphen enthalten sind; es werden keine neuen Daten durch Inferenz erarbeitet. Ab Jena-Version 2.3 (10/05) wird zusätzlich SPARQL[2], eine von der W3C RDF Data Access Working Group entwickelte Abfragesprache für RDF, unterstützt.

So genannte Reasoner (Schlussfolgerer) liefern zusätzliche, durch die Anwendung von Regeln hergeleitete Daten. Jena enthält vordefinierte Reasoner unterschiedlicher Leistungsfähigkeit. Diese können je nach Anforderung an einen Graphen gekoppelt werden.

- Der *Transitive Reasoner* arbeitet auf Hierarchien von Properties und Klassen, welche durch rdfs:subPropertyOf und rdfs:subClassOf aufgebaut sind.
- Der *RDFS Rule Reasoner* unterstützt alle RDFS-Axiome und -Schlussfolgerungen bis auf eine Regel im Zusammenhang mit typisierten Literalen.
- Die *OWL-, OWL Mini-, OWL Micro Reasoners* unterstützen unterschiedlich umfangreiche Untermengen von OWL Full.
- Der *DAML Micro Reasoner* ist ein um Axiome, die DAML-Sprachelemente auf RDFS-Sprachelementen abbilden, angereicherter RDFS Reasoner.
- Der *Generic Rule Reasoner* ist ein regelbasierter Reasoner. Die Regeln, welche unterstützt werden sollen, können definiert werden. Sowohl Vorwärts- und Rückwärtsverkettung als auch eine gemischte Verarbeitung sind möglich.

[1] http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
[2] http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050721/

Eigentlich kennt Jena nur zwei Reasoner, nämlich den *Transitive Reasoner* und den *Generic Rule Reasoner*. Der *Generic Rule Reasoner* wird genutzt, um Reasoner für RDFS, OWL und DAML zu implementieren. Diese Reasoner unterscheiden sich nur in ihren sprachspezifischen Regeln; diese können durch eigene Regeln erweitert werden. Der *Transitive Reasoner* ist eine festverdrahtete Java-Implementierung, die performanter und effizienter als der *Generic Rule Reasoner* arbeitet. Aus diesen Gründen kann der *Generic Rule Reasoner* optional den *Transitive Reasoner* nutzen, um rdfs:subPropertyOf und rdfs:subClassOf zu handhaben.

Kapitel 5

Implementierung

5.1 Maschinenverarbeitbare Darstellung relevanter Daten

5.1.1 Jena-spezifische Darstellung der Regeln

Schlussfolgerungsregeln (3.3.2.2), Regeln zum Verknüpfen von Lernobjekten aufgrund ihrer Klassifizierung (3.3.1.1) und Regeln zum Identifizieren von logisch unkorrekten Lernobjektverknüpfungen (4.2.3) werden als Jena-Regeln spezifiziert. Eine Regel besteht aus ein oder mehreren Voraussetzungen und Schlussfolgerungen, einer Ausführungsrichtung – Vorwärts- oder Rückwärtsverkettung – und einem Namen. Zusätzlich kann eine Regel Jena-spezifische Funktionen enthalten. Die Regeln der Verknüpfungsengine werden derart notiert, dass Jena sie in dem Modus der Rückwärtsverkettung ausführt.

Rückwärtsverkettung bedeutet, dass alle Schlussfolgerungen einer Regel als Ziele angesehen werden. Um ein Ziel zu erreichen, wird versucht, alle Veraussetzungen der Regel zu erfüllen. Dabei werden zum einen alle Ausgangsdaten herangezogen. Zum anderen werden auf den Ausgangsdaten Regeln angewandt und die resultierenden Schlussfolgerungen mit den zu erfüllenden Voraussetzungen verglichen. Im Gegensatz dazu wird im Modus der Vorwärtsverkettung nur die Übereinstimmung zwischen Voraussetzungen und Ausgangsdaten überprüft. Die Vorwärtsverkettung resultiert mit denselben Ausgangsdaten in weniger oder gleich vielen Schlussfolgerungen als die Rückwärtsverkettung.

Listing 5.1 zeigt drei Regeln.

Zeilen 1-3: Die Regel A *isReferencedBy* B $\wedge$ B *isPartOf* C $\rightarrow$ A *isReferencedBy* C wird ergänzt durch eine Jena-spezifische Funktion. Die Funktion *notEqual(?x,?y)* prüft die semantische Gleichheit ihrer Argumente; so werden zum Beispiel 1,0 und 1 als gleich angesehen. Innerhalb der Regel wird durch diesen Zusatz eine Selbstreferenz verhindert.

Zeilen 5-7: Diese Regel besagt, dass ein Lernobjekt A spezieller als ein Lernobjekt B ist, wenn A einem Taxonomieknoten zugeordnet ist, der spezieller als der Taxonomieknoten, der

B klassifiziert, ist. Die Property *lom:Taxon* wird innerhalb der Ontologie, die die Relationen abbildet definiert. Sie wird genutzt, um in der Lernobjektontologie Lernobjekte in Beziehung zu einem Taxonomieknoten zu setzen. Die Property *skos:broader* des SKOS-Vokabulars verweist auf ein allgemeineres Konzept. Die Funktion *notEqual(?x,?y)* ergänzt die Regel, ebenfalls um Selbstreferenzen zu verhindern.

Zeilen 9-10: Die aufgezeigte Regel identifiziert eine logisch unkorrekte Lernobjektverknüpfung mittels *isPartOf* oder *hasPart*. Die Relationen, die angeben, dass zwei Lernobjekte logisch unkorrekt verknüpft sind, werden innerhalb der Ontologie, die die Relationen abbildet, definiert.

Listing 5.1: eigene Regeln in Jena

```
1  [rule1: (?A lom:IsReferencedBy ?C) <-
2     (?A lom:IsReferencedBy ?B) (?B lom:IsPartOf ?C)
3     notEqual(?A, ?C)]
4
5  [Taxonomie1: (?A lom:IsNarrowerThan ?B) <-
6     (?A lom:Taxon ?P) (?B lom:Taxon ?Q) (?P skos:broader ?Q)
7     notEqual(?A, ?B)]
8
9  [incorrect1: (?A lom:IncorrectIsPartOf ?B) <-
10    (?A lom:IsPartOf ?B) (?B lom:IsPartOf ?A)]
```

5.1.2 OWL-Lite-Darstellung der Relationen

Die Relationen und ihre erweiterten Eigenschaften werden mittels einer OWL-Lite-Ontologie in eine maschinenverarbeitbare Form gebracht. Inversität, Transitivität und Symmetrie können durch die Sprachelemente *owl:inverseOf*, *owl:TransitiveProperty* und *owl:SymmetricProperty* abgebildet werden. Jenas Regelwerk zum Verarbeiten dieser Sprachelemente verhindert jedoch keine Selbstreferenzen, die durch bestimmte Konstellationen der Ausgangsdaten entstehen können.

Diese Konstellationen sind hauptsächlich Folgen logisch unkorrekter Lernobjektverknüpfungen. Folgende Situation führt zum Beispiel zu einer Selbstreferenz: A und B stehen gegenseitig mittels der transitiven Property *hasPart* in Beziehung. Jena zieht daraus den Schluss A *hasPart* A. Jedoch auch korrekte Verknüpfungen bedingen Selbstreferenzen: Aus A *isAlternativeTo* B schlussfolgert Jena A *isAlternativeTo* A, da die Relation *isAlternativeTo* sowohl transitiv als auch symmetrisch ist.

Um Selbstreferenzen zu verhindern, werden die genannten drei OWL-Sprachelemente überschrieben. Dazu werden zum einen die Klassen *lom:TransitiveProperty*[1] und *lom:SymmetricProperty* und die Property *lom:inverseOf* definiert. Zum anderen werden Regeln für das Ver-

[1] Der Namensraumpräfix *lom* ersetzt die URI *http://hylos.fhtw-berlin.de/HylosLOM#*.

arbeiten der neuen Sprachelemente aufgestellt. Dabei werden die Regeln der ursprünglichen OWL-Sprachelemente übernommen und um einen Zusatz, der Selbstreferenzen verhindert, ergänzt.

Listing 5.2 zeigt die Definition von *lom:SymmetricProperty* und *lom:inverseOf*.

Zeilen 6-8: Die Klasse *lom:SymmetricProperty* ist Unterklasse von *owl:ObjectProperty* und darf somit nur ein Individuum in Beziehung zu einem anderen Individuum setzten.

Zeilen 10-13: Die Property *lom:inverseOf* verlangt sowohl für die Domäne als auch für den Wertebereich die Angabe einer *owl:ObjectProperty*.

Listing 5.2: Definition von lom:inverseOf und lom:SymmetricProperty

```
1  ...
2     <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
3  ...
4  <rdf:RDF xml:base="http://hylos.fhtw-berlin.de/HylosLOM#">
5
6     <rdfs:Class rdf:ID="SymmetricProperty">
7        <rdfs:subClassOf rdf:resource="&owl;ObjectProperty;"/>
8     </rdfs:Class>
9
10    <rdf:Property rdf:ID="inverseOf">
11       <rdfs:domain rdf:resource="&owl;ObjectProperty"/>
12       <rdfs:range rdf:resource="&owl;ObjectProperty"/>
13    </rdf:Property>
14 ...
```

Listing 5.3 enthält die Regeln zu *lom:SymmetricProperty* und *lom:inverseOf*. Die Funktion *notEqual(?x,?y)* bewirkt, dass eine Regel nicht angewandt wird, wenn ein Individuum mit sich selbst in Beziehung gesetzt würde. Beide Regeln sind derart aufgebaut, dass eine Rückwärtsregel kreiert wird, falls die Voraussetzungen einer Vorwärtsregel erfüllt sind. Dadurch wird der Regelsatz der Verknüpfungsengine nur dann um die Regel zum Anwenden der Symmetrie beziehungsweise Inversität ergänzt, wenn eine symmetrische beziehungsweise inverse Property vorhanden ist.

Listing 5.3: Jena-spezifische Regeln für lom:inverseOf und lom:SymmetricProperty

```
1  [hylosSymmetric: (?P rdf:type lom:SymmetricProperty) ->
2     [hylosSymmetric_A: (?A ?P ?B) <- (?B ?P ?A) notEqual(?A, ?B)]
3  ]
4
5  [hylosInverse: (?P lom:inverseOf ?Q) ->
6     [hylosInverse_A: (?A ?P ?B) <- (?B ?Q ?A) notEqual(?A, ?B)]
7  ]
```

Zum Beschreiben der Relationen werden die neudefinierten Sprachelemente genutzt. Listing 5.4 zeigt die OWL-Beschreibung der Relationen *hasPart* und *isFormatOf*.

Zeile 4: Die Klasse *lom:LearningObject* wird definiert. Da alle Relationen Lernobjekte zueinander in Beziehung setzen, wird diese Klasse den Relationen als Domäne und Wertebereich zugewiesen. *lom:LearningObject* wird zusätzlich benötigt, um eine Lernobjektontologie zu erstellen. Lernobjekte sind Instanzen dieser Klasse.

Zeilen 6-11: Die Property *lom:HasPart* ist vom Typ *lom:TransitiveProperty* und invers zu *lom:IsPartOf*.

Zeilen 13-17: Die Property *lom:IsFormatOf* ist vom Typ *lom:SymmetricProperty*.

Listing 5.4: OWL-Darstellung von lom:HasPart und lom:IsFormatOf

```
1  ...
2  <rdf:RDF xml:base="http://hylos.fhtw-berlin.de/HylosLOM#">
3
4     <owl:Class rdf:ID="LearningObject"/>
5
6     <owl:ObjectProperty rdf:ID="HasPart">
7        <rdf:type rdf:resource="#TransitiveProperty"/>
8        <lom:inverseOf rdf:resource="#IsPartOf" />
9        <rdfs:range rdf:resource="#LearningObject"/>
10       <rdfs:domain rdf:resource="#LearningObject"/>
11    </owl:ObjectProperty>
12
13    <owl:ObjectProperty rdf:ID="IsFormatOf">
14       <rdf:type rdf:resource="#SymmetricProperty"/>
15       <rdfs:range rdf:resource="#LearningObject"/>
16       <rdfs:domain rdf:resource="#LearningObject"/>
17    </owl:ObjectProperty>
18 ...
```

5.1.3 SKOS-Darstellung der Taxonomien

Die Taxonomien, anhand derer Lernobjekte klassifiert werden, werden mittels SKOS in eine maschinenverarbeitbare Form gebracht. Die RDF-RDFS-Beschreibung von SKOS wird der Verknüpfungsengine zur Verarbeitung der Taxonomiebeschreibungen übergeben; sie kann von der SKOS-Hauptseite[2] heruntergeladen werden. Jeder Taxonomieknoten ist ein *skos:Concept*. SKOS-Properties beschreiben die Art der Beziehung zwischen Taxonomieknoten.

Die Property *skos:related* wird in der RDF-RDFS-Beschreibung von SKOS als symmetrisch definiert. Diese Angabe wurde auskommentiert, da *skos:related* auf der Relation *references* ab-

[2] http://www.w3.org/2004/02/skos/

gebildet wird und Symmetrie zu Kreisen innerhalb der Lernobjektverknüpfungen führt. Diese Kreise sind zwar nicht falsch, jedoch entsteht durch die inverse Relation *isReferencedBy* ein zweiter Kreis. Um doppelte Kreisbildung zu verhindern, wird die Symmetrie der Property *skos:related* auf den inversen Relationen *references/isReferencedBy* abgebildet. Ein weiteres Argument gegen Symmetrie ist, dass die ACM-Taxonomie nirgendwo besagt, dass Querverweise bidirektional sind.

Listing 5.5 enthält die SKOS-Beschreibung des ACM-Taxonomieknotens *D.3.3 Language Constructs and Features*.

Zeile 4 gibt die eindeutige Bezeichnung des Taxonomieknotens an.

Zeilen 5-7 geben den bevorzugten Namen des Taxonomieknotens in englischer Sprache an.

Zeilen 8-9 geben an, dass der Taxonomieknoten zur ACM-Taxonomie-Version des Jahres 1998 gehört.

Zeilen 10-14 geben alle direkten Kindknoten an. Diese sind spezieller.

Zeilen 15-16 geben an, dass E.2 ein inhaltlich nahe stehender Taxonomieknoten ist.

Listing 5.5: ACM-Knoten D.3.3 in SKOS

```
1  ...
2  <skos:Concept
3     rdf:about="http://www.acm.org/class/1998/D.3.3">
4     <skos:externalID>D.3.3</skos:externalID>
5     <skos:prefLabel xml:lang="en">
6        Language Constructs and Features
7     </skos:prefLabel>
8     <skos:inScheme
9        rdf:resource="http://www.acm.org/class/1998/"/>
10    <skos:narrower
11       rdf:resource="http://www.acm.org/class/1998/D.3.3.1"/>
12    ...
13    <skos:narrower
14       rdf:resource="http://www.acm.org/class/1998/D.3.3.16"/>
15    <skos:related
16       rdf:resource="http://www.acm.org/class/1998/E.2"/>
17 </skos:Concept>
18 ...
```

5.1.4 OWL-Lite-Darstellung der Lernobjektmetadaten

Die benötigten Lernobjektmetadaten werden während des Programmlaufs aus der Datenbank gelesen und daraus eine OWL-Lite-Ontologie erstellt. Listing 5.6 zeigt die Beschreibung des Lernobjektes LO_A in solch einer Lernobjektontologie.

Zeile 7: LO_A ist Instanz der Klasse *lom:LearningObject*.

Zeile 8: LO_A steht mittels *isPartOf* in Beziehung zu LO_B.

Zeile 9: LO_A steht mittels *isReferencedBy* in Beziehung zu LO_C.

Zeile 10: LO_A ist unterhalb des Taxonomieknotens D.3.2 der ACM-Taxonomie eingeordnet.

Listing 5.6: Ausschnitt aus Lernobjektontologie

```
1  ...
2  <!ENTITY lom 'http://hylos.fhtw-berlin.de/HylosLOM#'>
3  <!ENTITY hylos 'http://hylos.fhtw-berlin.de/server/content/'>
4  <!ENTITY acm 'http://www.acm.org/class/1998/'>
5  ...
6  <rdf:Description rdf:about="&hylos;LO_A">
7     <rdf:type rdf:resource="&lom;LearningObject"/>
8     <lom:IsPartOf rdf:resource="&hylos;LO_B"/>
9     <lom:IsReferencedBy rdf:resource="&hylos;LO_C"/>
10    <lom:Taxon rdf:resource="&acm;D.3.2"/>
11 </rdf:Description>
12 ...
```

5.2 Ausgewählte Klassen

Die in Abschnitt 4.4 konzipierte API der Verknüpfungsengine wurde vollständig umgesetzt. Die Interfaces *Derivation*, *IncorrectRelationship* und *SupposedRelationship* sind in der Weise implementiert worden, dass eine Textdatei zur Speicherung der Daten genutzt wird. Für das Interface *RelationshipFinder* liegen zwei Implementierungen vor. Eine Implementierung verwendet Jena als externe Schlussfolgerungsmaschine; die aktuelle Version verarbeitet neben den Schlussfolgerungsregeln und den erweiterten Relationseigenschaften auch die Klassifikationsinformation eines Lernobjektes. Die zweite Implementierung realisiert die Strategie, Lernobjektverknüpfungen aus der Lernobjektstruktur abzulesen. Die Umsetzung der heuristischen Verfahren steht noch aus. Die folgenden Unterabschnitte stellen ausgewählte Klassen vor.

Ziel der aktuellen Version war, den allgemeinen Ablauf der Verknüpfungsengine zu implementieren. Deshalb sind die in Abschnitt 4.3 entwickelten Strategien zur Skalierbarkeit nicht berücksichtigt worden.

5.2.1 Die Klasse *Identifier*

Die Klasse *Identifier* kapselt einen eindeutigen Bezeichner. Da eindeutige Bezeichner von unterschiedlichen Datentypen sein können, besitzt diese Klasse mehrere Felder, um den Wert eines eindeutigen Bezeichners aufzunehmen. Listing 5.7 zeigt ausschnittsweise die Klasse *Identifier*.

Zeile 4: Das Feld *mode* gibt an, welches Feld den Wert des eindeutigen Bezeichners aufgenommen hat.

Zeilen 5-6: Die aktuelle Implementierung besitzt ein Feld vom Typ *int* und eines vom Typ *String*. Sie kann problemlos um weitere Datentypen erweitert werden.

Zeilen 8-11: Für jeden möglichen Datentyp besitzt diese Klasse einen Konstruktor, der sowohl den Wert des eindeutigen Bezeichners als auch den passenden Wert des Feldes *mode* setzt. Es existieren keine Methoden, um diese Werte nachträglich zu verändern.

Zeilen 13-18: Für jeden möglichen Datentyp besitzt diese Klasse eine Methode, die den Wert des eindeutigen Bezeichners liefert. Wird eine dieser Methoden für den falschen Datentyp aufgerufen, wird eine *Exception* geworfen.

Zeilen 20-30: Die Klasse *Identifier* bietet eine statische Methode, um ein *String*-Objekt in ein *Identifier*-Objekt zu konvertieren. (Zusätzlich wird die Methode *toString* der Klasse *Object* überschrieben, die ein *Identifier*-Objekt in ein *String*-Objekt umwandelt.) Die aktuelle Implementierung überprüft, ob das übergebene *String*-Objekt in einen *int*-Wert umgewandelt werden kann. Ein *Identifier*-Objekt wird entweder aus dem ursprünglichen oder dem in einen *int*-Wert umgewandelten *String*-Objekt erzeugt und zurückgeliefert.

Zeilen 32-37: Die Methode *equals* der Klasse *Object* wird überschrieben. Besitzen die eindeutigen Bezeichner zweier *Identifier*-Objekte nicht denselben Datentyp, wird *false* zurückgeliefert. Ansonsten werden die *String*-Repräsentationen beider Objekte verglichen und das Ergebnis zurückgeliefert.

Listing 5.7: Klasse *Identifier*

```
1  public class Identifier {
2    public static final int INT = 0;
3    public static final int STRING = 1;
4    private int mode;
5    private int int_id;
6    private String string_id;
7
8    public Identifier(int id) {
9      this.mode = Identifier.INT;
10     this.int_id = id;
11   }
12 ...
13   public int getInt_id() throws Exception {
14     if(this.mode == Identifier.INT)
15       return this.int_id;
16     else
17       throw new Exception();
```

```
18   }
19 ...
20   public static Identifier createIdentifier(String str) {
21
22     int i;
23     try {
24       i = Integer.parseInt(str);
25     }
26     catch(NumberFormatException e) {
27       return new Identifier(str);
28     }
29     return new Identifier(i);
30   }
31
32   public boolean equals(Object o) {
33
34     if(this.mode != ((Identifier)o).getMode())
35       return false;
36     return this.toString().equals(o.toString());
37   }
38 }
```

5.2.2 Die Klasse *DatabaseMIRImpl*

Die Klasse *DatabaseMIRImpl* implementiert das Interface *Database* derart, dass sie auf das dem HyLOS-System zugrunde liegende Media Information Repository zugreift.

Die Methode *getAllLOs* (siehe Listing 5.8) verlangt neben einem Suchfilter einen Modus, welcher angibt, wie der Suchfilter interpretiert werden muss. Das Interface *Database* definiert drei Modi: LDAP, SQL, XQUERY.

Listing 5.8: Definition der Methode *getAllLOs* des Interfaces *Database*

```
public List getAllLOs(int query_mode, String query) throws Exception;
```

Die Methode *getAllLOs* der Klasse *DatabaseMIRImpl* delegiert in Abhängigkeit des Modus den Suchfilter an eine klassenspezifische Methode. Ist der Modus unbekannt, wird eine *Exception* geworfen. Listing 5.9 zeigt die Methode *ldapQuery*, welche einen LDAP-konformen Suchfilter verarbeitet.

Zeilen 3-4: Ein *Repository*-Objekt, mittels dessen auf das Media Information Repository zugegriffen werden kann, wird abgerufen.

Zeilen 8-10: Die Methode *search* der Klasse *Repository* verlangt neben weiteren Parametern einen LDAP-konforme Suchfilter. Die Konstante *Constants.DataModel.ROOTPATH* gibt das Wurzelverzeichnis für die Suche an. (Innerhalb eines Media Information Repositorys werden Lernobjekte virtuellen Verzeichnissen zugeordnet.) Der Rückgabewert ist eine

Liste von *DirectoryEntry*-Objekten. Die Klasse *DirectoryEntry* kapselt unter anderem den globalen Pfad eines Objektes innerhalb des virtuellen Verzeichnisses. Der globale Pfad ist eindeutig.

Zeilen 11-17: Innerhalb einer Schleife wird für jedes *DirectoryEntry*-Objekt der globale Pfad extrahiert, ein Objekt des Typs *Identifier* erzeugt und dieses einem *Vector*-Objekt hinzugefügt, welches der Rückgabewert der Methode *ldapQuery* ist.

Listing 5.9: Methode *ldapQuery* der Klasse *DatabaseMIRImpl*

```
1  private List ldapQuery(String query) throws Exception {
2
3    MirSession session = MirSession.getInstance();
4    Repository repository = session.getRepository();
5
6    Vector v = new Vector();
7
8    List dirEntries = repository.search(
9                        Constants.DataModel.ROOTPATH,
10                       query, 10000, true, false);
11   for(Iterator dirEntryIter = dirEntries.iterator();dirEntryIter.hasNext();){
12
13     DirectoryEntry dirEntry = (DirectoryEntry)dirEntryIter.next();
14     String globalPath = dirEntry.getGlobalPath();
15     Identifier id = new Identifier(globalPath);
16     v.add(id);
17   }
18   return v;
19 }
```

Listing 5.10 zeigt die Methode *getPropertyValues*. Die Methode verlangt als Parameter neben einem *Identifier*-Objekt, welches genau ein Lernobjekt bezeichnet, den eindeutigen Namen einer Property. Ein eindeutiger Property-Name kann der vollständige Name eines LOM-Datenelementes (zum Beispiel *Technical.Format*) sein. Im HyLOS-System bildet jede LOM-Kategorie eine eigene Klasse, deren Properties je nach Struktur der LOM-Kategorie einfache Datentypen oder Objekte sind. Ein Lernobjekt verweist auf Objekte dieser Klassen. Es kann mehrere Objekte für jede LOM-Kategorie kennen. Die Methode *getPropertyValues* ist flexibel in Bezug auf den übergebenen Property-Namen implementiert; sie arbeitet nicht mit einkodierten Property-Namen.

Zeilen 4-5: Der Property-Name wird in seine Einzelteile zerlegt (zum Beispiel in *Technical* und *Format*).

Zeile 10: Das Lernobjekt, welches von dem übergebenen *Identifier*-Objekt bezeichnet wird, wird als *GenericMob*-Objekt geladen. *GenericMob* ist eine Superklasse der Klasse aller Lernobjekte.

Zeilen 12-13: Ein *Vector*-Objekt wird erzeugt und mit dem Lernobjekt gefüllt. Es wird benötigt, um über Objekte des Typs *GenericMob* zu iterieren.

Zeile 14: Ein zweites *Vector*-Objekt wird erzeugt. Es wird benötigt, um Property-Werte zwischenzuspeichern.

Zeile 16: Eine Schleife wird initiiert, die über die Einzelteile des Property-Names iteriert.

Zeile 20: Eine innere Schleife wird initiiert, die über die Elemente des *Vector*-Objektes, das beim ersten Durchlauf mit dem übergebenen Lernobjekt gefüllt ist, iteriert.

Zeilen 22-23: Der Wert des aktuellen Property-Namen-Teils des aktuellen Elementes wird gelesen.

Zeilen 25-33: Falls der Wert eine Liste von eindeutigen Bezeichnern von *GenericMob*-Objekten ist, werden diese Objekte geladen und dem zweiten *Vector*-Objekt hinzugefügt.

Zeilen 34-39: Falls der Wert ein eindeutiger Bezeichner eines *GenericMob*-Objektes ist, wird das Objekt geladen und dem zweiten *Vector*-Objekt hinzugefügt.

Zeilen 40-45: Falls beide Fälle nicht zutreffen, wird der Wert dem zweiten *Vector*-Objekt hinzugefügt, sofern der letzte Durchlauf der Schleife, die über die Einzelteile des Property-Namens iteriert, erreicht ist. Ansonsten endet die Methode, indem ein leeres *Vector*-Objekt zurückgeliefert wird.

Zeilen 47-49: Am Ende eines Durchlaufs der Schleife, die über die Einzelteile des Property-Namens iteriert, werden die Elemente des ersten *Vector*-Objekts durch die Elemente des zweiten *Vector*-Objekts ersetzt und dieses geleert.

Zeilen 51-55: Nach dem Ende der Schleife, die über die Einzelteile des Property-Namens iteriert, enthält das erste *Vector*-Objekt entweder Elemente des Typs *GenericMob* oder *Object*. Elemente des letzteren Typs umhüllen einfache Datentypen oder *String*-Objekte. Die Elemente werden in *Property*-Objekte umgewandelt und dem zweiten *Vector*-Objekt hinzugefügt, welches anschließend zurückgeliefert wird.

Listing 5.10: Methode *getPropertyValues* der Klasse *DatabaseMIRImpl*

```
1 public List getPropertyValues(Identifier id, String name) throws Exception {
2
3   //eindeutigen Property-Namen in Einzelteile zerlegen
4   StringTokenizer t = new StringTokenizer(name, ".", false);
5   int length = t.countTokens();
6
7   MirSession session = MirSession.getInstance();
8   Repository repository = session.getRepository();
```

```
9
10    GenericMob mob = repository.getMob(id.getString_id());
11
12    Vector mobs = new Vector();
13    mobs.add(mob);
14    Vector values = new Vector();
15
16    for(int i = 1; i <= length; i++) {
17
18      String propertyNamePart = t.nextToken();
19
20      for(Iterator iterator = mobs.iterator(); iterator.hasNext(); ) {
21
22        GenericMob mob_ = (GenericMob)iterator.next();
23        Object o = mob_.getValue(propertyNamePart);
24
25        if(this.isIdentifierList(o, mob_)) {
26
27          for(Iterator iList = ((List)o).iterator(); iList.hasNext(); ) {
28
29            Target target = mob_.getTarget((String)iList.next());
30            String globalPath = target.getGlobalPath();
31            values.add(repository.getMob(globalPath));
32          }
33        }
34        else if(this.isIdentifier(o, mob_)) {
35
36          Target target = mob_.getTarget((String)o);
37          String globalPath = target.getGlobalPath();
38          values.add(repository.getMob(globalPath));
39        }
40        else {
41
42          if(i != length)
43            return new Vector();
44          values.add(o);
45        }
46      }
47      mobs = null;
48      mobs = new Vector(values);
49      values.clear();
50    }
51    for(Iterator i = mobs.iterator(); i.hasNext(); ) {
52      Property p = new Property(id, name, i.next());
53      values.add(p);
54    }
55    return values;
```

```
56 }
```

5.2.3 Die Klasse *RelationshipFinderJenaImpl*

Die Klasse *RelationshipFinderJenaImpl* implementiert das Interface *RelationshipFinder*. Es setzt die Strategien, die mittels einer externen Schlussfolgerungsmaschine realisiert werden, mit dem Semantic-Web-Toolkit Jena um.

Der eigentliche Schlussfolgerungsmechanismus muss nicht implementiert werden. Alle Daten, die die Basis für das Schlussfolgern bilden, werden den zuständigen Jena-Klassen übergeben. Das Jena-Interface *InfModel* bietet Zugriff auf den Schlussfolgerungsmechanismus. Über eine Fabrikklasse wird eine Instanz eines Inferenzmodells kreiert. Solch eine Instanz kapselt Daten, Modell, Reasoner und Regeln. In Bezug auf die Verknüpfungsengine bilden Taxonomien und Lernobjektmetadaten die Daten. Die RDF-RDFS-Beschreibung von SKOS und die Relationenontologie – einschließlich der OWL-Beschreibungen der Property *lom:Taxon*, der Klasse *lom:LearningObject* und der Relationen, die angeben, dass zwei Lernobjekte logisch unkorrekt verknüpft sind – bilden das Modell.

Das Jena-Interface *InfModel* definiert unter anderem die Methode *listStatements*. Dieser Methode werden Jena-Objekte, die Subjekt, Prädikat und Objekt eines RDF-Tripels repräsentieren, übergeben. Anstatt eines konkreten Jena-Objektes kann auch *null* übergeben werden. Der Rückgabewert der Methode ist ein Iterator über alle Aussagen – entweder geschlussfolgerte oder in den Ursprungsdaten enthaltene –, deren Subjekt, Prädikat und Objekt den übergebenen Jena-Objekten entsprechen. Besitzt ein übergebenes Jena-Objekt den Wert *null*, so kann in den zurückgelieferten Aussagen an der entsprechenden Stelle jeder mögliche Wert stehen. Für jedes Lernobjekt, das auf mögliche neue Beziehungen überprüft werden soll, wird die Methode *listStatements* für jede Relation mit unspezifiziertem Objekt aufgerufen. Abbildung 5.1 auf Seite 69 zeigt, wie die gefundenen Aussagen verarbeitet werden.

Listings 5.11 und 5.12 zeigen, wie überprüft wird, ob eine Lernobjektverknüpfung korrekt hergeleitet wurde.

Zeile 3: Eine Variable vom Typ *boolean* wird erzeugt und mit dem Wert *false* initiiert.

Zeile 5: Eine Schleife wird initiiert, die über alle *RuleDerivation*-Objekte der übergebenen Aussage, die eine Lernobjektverknüpfung repräsentiert, iteriert. Die Methode *getDerivation* des Inferenzmodells liefert ein entsprechendes *Iterator*-Objekt. (Jena kennt meistens mehrere Wege, eine Aussage herzuleiten.) Ein *RuleDerivation*-Objekt kapselt keine Schlussfolgerungskette sondern genau einen -schritt – eine Schlussfolgerung, eine Regel, ein oder mehrere Ausgangsdaten. Soll eine Schlussfolgerungskette erstellt werden, müssen die *RuleDerivation*-Objekte der Ausgangsdaten rekursiv solange untersucht werden, bis ein Ausgangsdatum ein Ursprungsdatum darstellt.

Zeilen 7-9: Das aktuelle *RuleDerivation*-Objekt wird der Methode *isIncorrectRelationIncluded* übergeben. Liefert die Methode den Wert *false* zurück, ist in der aus dem übergebenen *RuleDerivation*-Objekt erstellten Schlussfolgerungskette keine logisch unkorrekte Lernobjektverknüpfung enthalten. In diesem Fall erhält die in der dritten Zeile erzeugte Variable den Wert *true*. Andernfalls bleibt ihr aktueller Wert unverändert.

Zeile 11: Nachdem die Schleife durchlaufen wurde, wird der aktuelle Wert der Variable zurückgeliefert.

Listing 5.11: Methode *isCorrectlyInferred* der Klasse *RelationshipFinderJenaImpl*

```
1  private boolean isCorrectlyInferred(Statement statement) {
2
3    boolean correct = false;
4
5    for(Iterator i = this.infModel.getDerivation(statement); i.hasNext();) {
6
7      RuleDerivation rd = (RuleDerivation)i.next();
8      correct =
9        !this.isIncorrectRelationIncluded(rd, new HashSet()) ? true : correct;
10   }
11   return correct;
12 }
```

Zeile 4: Aus dem übergebenen *RuleDerivation*-Objekt werden alle Ausgangsdaten abgerufen.

Zeile 5: Eine Schleife, die über die Ausgangsdaten iteriert, wird initiiert.

Zeilen 9-10: Falls das aktuelle Ausgangsdatum eine logisch unkorrekte Lernobjektverknüpfung ist, wird die Methode beendet und der Wert *true* zurückgeliefert.

Zeile 12: Ein *Iterator*-Objekt über alle *RuleDerivation*-Objekte des aktuellen Ausgangsdatums wird abgerufen.

Zeile 13: Es wird überprüft, ob über dieses *Iterator*-Objekt iteriert werden kann.

Zeilen 15-16: Das erste *RuleDerivation*-Objekt wird abgerufen.

Zeilen 17-21: Falls dieses *RuleDerivation*-Objekt nicht in dem der Methode übergebenen Set enthalten ist, wird es dem Set hinzugefügt und die Methode rekursiv mit diesem *RuleDerivation*-Objekt und dem übergebenen Set als Parameter aufgerufen.

Zeile 24: Falls die Methode nicht vorher beendet wurde, sind keine logisch unkorrekten Lernobjektverknüpfungen in einer Schlussfolgerungskette enthalten. Die Methode liefert den Wert *false* zurück.

Listing 5.12: Methode *isIncorrectRelationIncluded* der Klasse *RelationshipFinderJenaImpl*

```
1  private boolean isIncorrectRelationIncluded(
2                                  RuleDerivation derivation, Set seen){
3
4    List matches = derivation.getMatches();
5    for(int i = 0; i < matches.size(); i++) {
6
7      Triple match = (Triple)matches.get(i);
8
9      if(match != null && this.isIncorrectRelation(match))
10       return true; //logisch unkorrekte Verknüpfung
11
12     Iterator derivationsOfMatch = this.infGraph.getDerivation(match);
13     if(derivationsOfMatch != null && derivationsOfMatch.hasNext()) {
14
15       RuleDerivation nextDerivation =
16                              (RuleDerivation)derivationsOfMatch.next();
17       if(!seen.contains(nextDerivation)) {
18
19         seen.add(nextDerivation);
20         this.isIncorrectRelationIncluded(nextDerivation, seen);
21       }
22     }
23   }
24   return false;
25 }
```

5.2.4 Die Klasse *DerivationFileImpl*

Die Klasse *DerivationFileImpl* implementiert das Interface *Derivation*. Schlussfolgerungsschritte werden in einer Textdatei gespeichert. Dem Konstruktor wird der Name dieser Datei übergeben. Zur Laufzeit verwaltet die Klasse *DerivationFileImpl* die Daten in einer Hashtabelle, wobei die Ausgangsdaten als *String*-Objekte die Schlüssel bilden und ein zugehöriger Wert ein Set, das die direkt resultierenden Schlussfolgerungen als *String*-Objekte enthält, ist. Die übergebene Textdatei wird eingelesen und die Hashtabelle gefüllt; die Methode *performFinalAction* schreibt die Hashtabelle in die Textdatei zurück.

Listing 5.13 zeigt die Implementierung der Methode *delete*. Diese Methode löscht ein übergebenes *Relation*-Objekt, das eine geschlussfolgerte Lernobjektverknüpfung repräsentiert, aus den gespeicherten Schlussfolgerungsschritten; sie wird aufgerufen, nachdem alle aufgrund eines gelöschten Datums geschlussfolgerten Lernobjektverknüpfungen identifiziert und zurückgesetzt wurden.

Zeile 3: Das übergebene *Relation*-Objekt wird in ein *String*-Objekt umgewandelt.

Zeilen 5-6: Eine Schleife, die über alle Schlüssel der Hashtabelle iteriert, wird initiiert.

Zeilen 8-9: Das Set, welches Wert des aktuellen Schlüssels ist, wird abgerufen.

Zeilen 10-12: Wenn das übergebene *Relation*-Objekt in diesem Set enthalten ist, wird es aus dem Set gelöscht. Damit besitzt ein Ausgangsdatum eine Schlussfolgerung weniger.

Zeilen 13-15: Wenn dieses Set leer ist, existieren zu einem Ausgangsdatum keine Schlussfolgerungen mehr. Das Ausgangsdatum wird aus der Hashtabelle gelöscht.

Listing 5.13: Methode *delete* der Klasse *DerivationFileImpl*

```
1  public void delete(Relation conclusion) {
2
3    String r = conclusion.toString();
4
5    for(Enumeration enumeration = this.conclusionList.keys();
6                                  enumeration.hasMoreElements(); ) {
7
8      String match = (String)enumeration.nextElement();
9      HashSet conclusions = (HashSet)this.conclusionList.get(match);
10     if(conclusions.contains(r)) {
11       conclusions.remove(r);
12     }
13     if(conclusions.isEmpty()) {
14       this.conclusionList.remove(match);
15     }
16   }
17 }
```

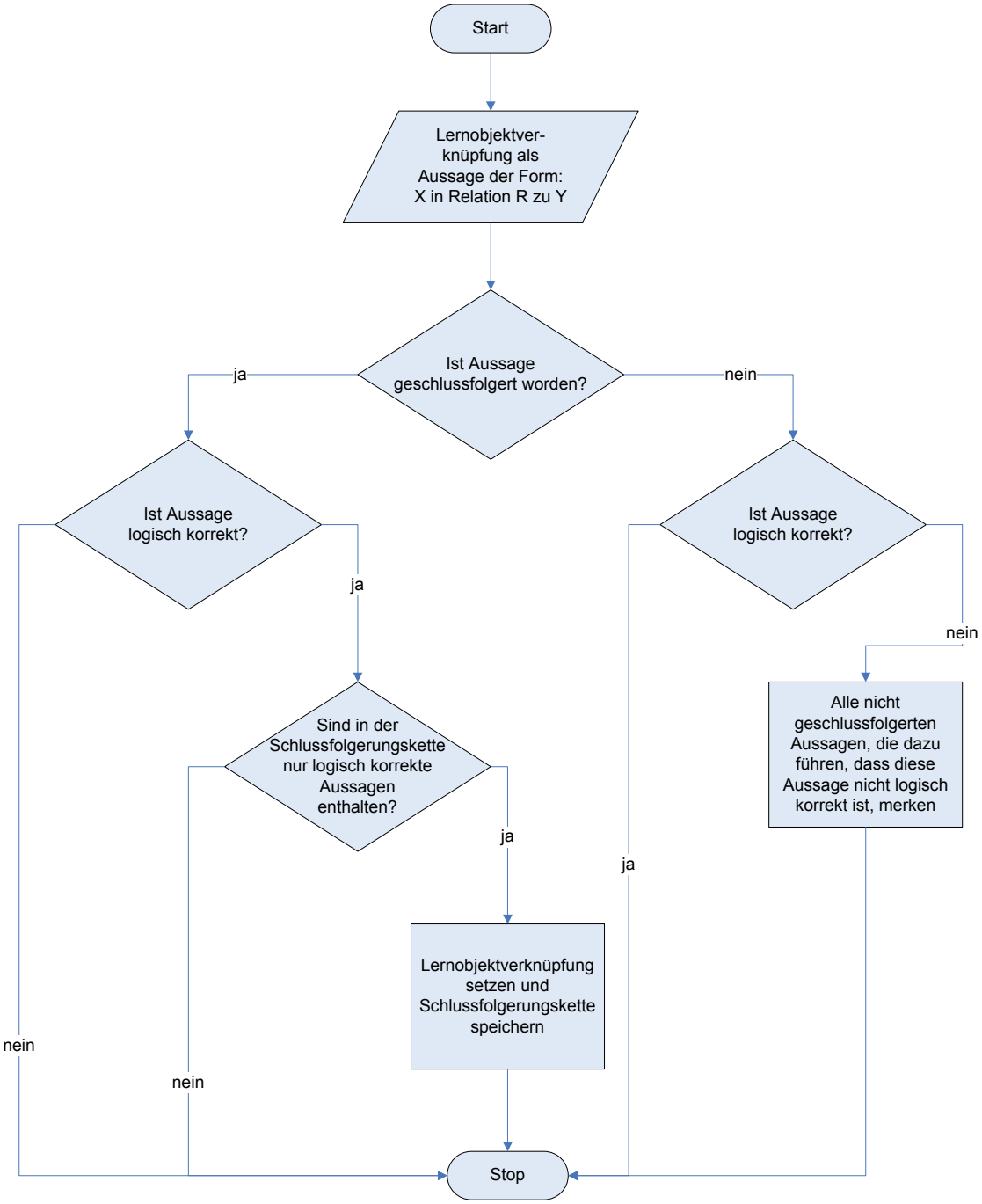

Abbildung 5.1: Algorithmus zur Verarbeitung gefundener Lernobjektverknüpfungen

Kapitel 6

Auswertung des Ablauftests

Der allgemeine Ablauf der Verknüpfungsengine sollte überprüft werden. Folgende Funktionen wurden dafür getestet:

- Einlesen relevanter Daten und das Erstellen eines Jena-spezifischen Modells aus diesen Daten
- Identifizieren neuer und logisch unkorrekter Lernobjektverknüpfungen
- Verarbeiten gelöschter Daten

Zu diesem Zweck wurde die Klasse *DatabaseMIRImpl* erweitert, sodass alle Lernobjekte unterhalb eines angegebenen Pfades unabhängig von ihrem Erstellungs- oder Änderungsdatum oder einem anderen Kriterium gelesen werden konnten. Diese Möglichkeit ist ausschließlich zum Testen geschaffen worden. Zusätzlich wurde ein Testverzeichnis angelegt. Abbildung 6.1 zeigt alle Lernobjekte dieses Verzeichnisses, die vorhandenen Lernobjektverknüpfungen, Strukturbeziehungen und Zuordnungen zu einem Taxonomieknoten.

Jena bietet die Möglichkeit, Ontologien zur Laufzeit zu erstellen. Um das Aufbauen des Jena-spezifischen Modells zu testen, wurden alle Lernobjekte des Testverzeichnisses mittels der Testmethode eingelesen. In einer Schleife wurden für jedes Lernobjekt dem zuvor erzeugten Modell Aussagen über Verknüpfungen und Zuordnungen zu einem Taxonomieknoten hinzugefügt. Danach wurde das Modell als RDF/XML in eine Textdatei geschrieben. (Listing 5.6 auf Seite 59 bildet ein Beispiel dieses Modells ab.) Anhand dieser Textdatei wurde erkannt, dass das Modell korrekt erstellt wurde. Das Erstellen des Modells einschließlich dem Einlesen der Daten dauerte 2469 ms.

In einem zweiten Schritt wurde das Identifizieren neuer und logisch unkorrekter Lernobjektverknüpfungen getestet. (Abbildung 5.1 auf Seite 69 verdeutlicht, warum beide Funktionen in einem Schritt überprüft wurden.) Die zwei momentanen Implementierungen des Interfaces *RelationshipFinder* wurden nacheinander ausgeführt. Zuerst wurden die Lernobjektstrukturen des Testverzeichnisses untersucht. Für das Einlesen der Daten und das Identifizieren und

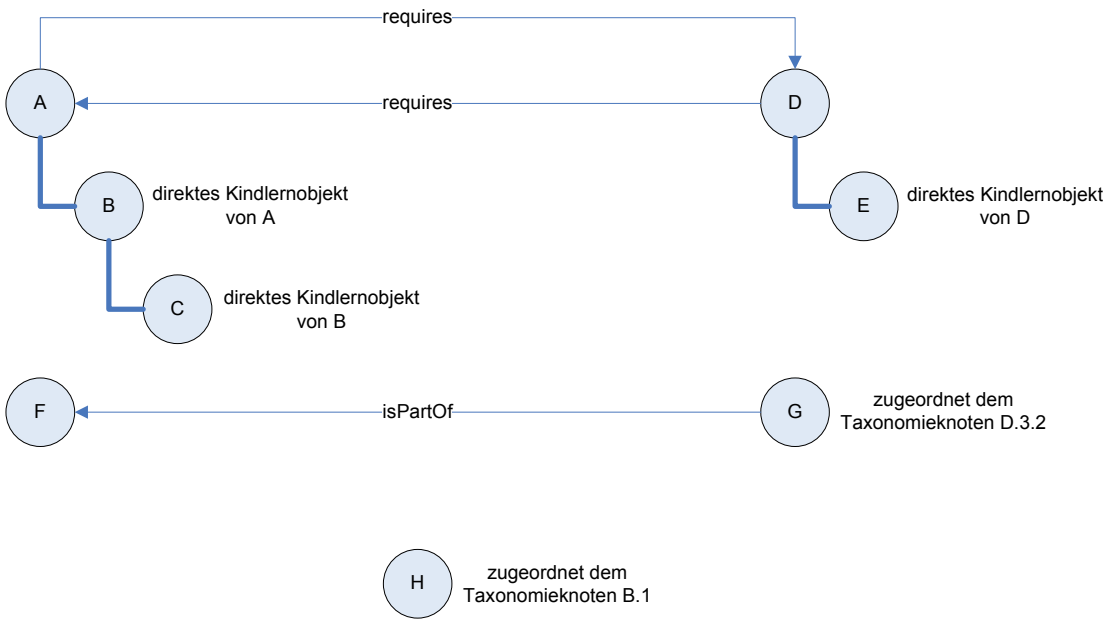

Abbildung 6.1: Ausgangsituation im Testverzeichnis

Speichern von Lernobjektverknüpfungen wurden 15656 ms benötigt. Anschließend erstellte die Jena-spezifische Implementierung ein Modell wie beschrieben und schlussfolgerte auf diesem. Das Schlussfolgern dauerte 14672 ms. Diese Zeitangabe beinhaltet das Speichern neuer Lernobjektverknüpfungen und ihrer Schlussfolgerungsketten sowie das Speichern logisch unkorrekter Lernobjektverknüpfungen und ihrer Ursachen. Die Arbeit der Verknüpfungsengine wurde überprüft, indem die Lernobjekte des Testverzeichnisses in dem Autorenwerkzeug geöffnet wurden. Außerdem wurden die gespeicherten Schlussfolgerungsketten und die gespeicherten Informationen über logisch unkorrekte Lernobjektverknüpfungen ausgewertet. Abbildung 6.2 zeigt einen Ausschnitt aus der Textdatei, in der die Schlussfolgerungsketten gespeichert wurden. (In Abschnitt 4.2.4 auf Seite 46 wird erläutert, wie die Textdatei interpretiert werden muss.) Abbildung 6.3 zeigt alle identifizierten Lernobjektverknüpfungen. Die Verknüpfungsengine identifizierte alle möglichen neuen und alle logisch unkorrekten Lernobjektverknüpfungen.

Zuletzt wurde das Verarbeiten gelöschter Daten getestet. Dazu wurden eine Lernobjektverknüpfung, die eine Ursache für eine logisch unkorrekte Lernobjektverknüpfung bildete, und eine Zuordung zu einem Taxonomieknoten, aufgrund derer neue Lernobjektverknüpfungen identifiziert wurden, gelöscht. In eine Textdatei wurden diese zwei Daten geschrieben; die Syntax eines Datums war gleich der in den gespeicherten Schlussfolgerungsketten verwendeten Syntax. Diese Textdatei wurde vor dem Starten der beiden *RelationshipFinder*-Implementierungen verarbeitet. Die Arbeit der Verknüpfungsengine konnte ebenfalls mittels des Autorenwerkzeuges und den gespeicherten Schlussfolgerungsketten und den gespeicherten Informationen über logisch unkorrekte Lernobjektverknüpfungen überprüft werden. Alle aufgrund der gelöschten Zuordnung zu einem Taxonomieknoten geschlussfolgerten Lernobjektverknüpfungen waren zurückgesetzt,

```
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno3/Lerno References /HyLOs/content/Data/dagmar-test/OntTest/Lerno4/Lerno
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno4/Lerno IsReferencedBy /HyLOs/content/Data/dagmar-test/OntTest/Lerno3/Lerno

RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno2/Lerno IsPartOf /HyLOs/content/Data/dagmar-test/OntTest/Lerno4/Lerno
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno3/Lerno References /HyLOs/content/Data/dagmar-test/OntTest/Lerno4/Lerno
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno4/Lerno HasPart /HyLOs/content/Data/dagmar-test/OntTest/Lerno2/Lerno

RELATION /HyLOs/content/Data/dagmar-test/OntTest/Unter1/Unter IsPartOf /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Unter1/Unter IsPartOf /HyLOs/content/Data/dagmar-test/OntTest/Lerno/Lerno

PROPERTY /HyLOs/content/Data/dagmar-test/OntTest/Unter1/Unter childOf /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Unter1/Unter IsPartOf /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter

PROPERTY /HyLOs/content/Data/dagmar-test/OntTest/Lerno/Lerno parentOf /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno/Lerno HasPart /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter

CLASSIFICATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno2/Lerno http://www.acm.org/class/1998/D.3.2
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno3/Lerno References /HyLOs/content/Data/dagmar-test/OntTest/Lerno2/Lerno

RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno/Lerno HasPart /HyLOs/content/Data/dagmar-test/OntTest/Unter/Unter
RELATION /HyLOs/content/Data/dagmar-test/OntTest/Lerno/Lerno HasPart /HyLOs/content/Data/dagmar-test/OntTest/Unter1/Unter
```

Abbildung 6.2: Ausschnitt aus den gespeicherten Schlussfolgerungsschritten

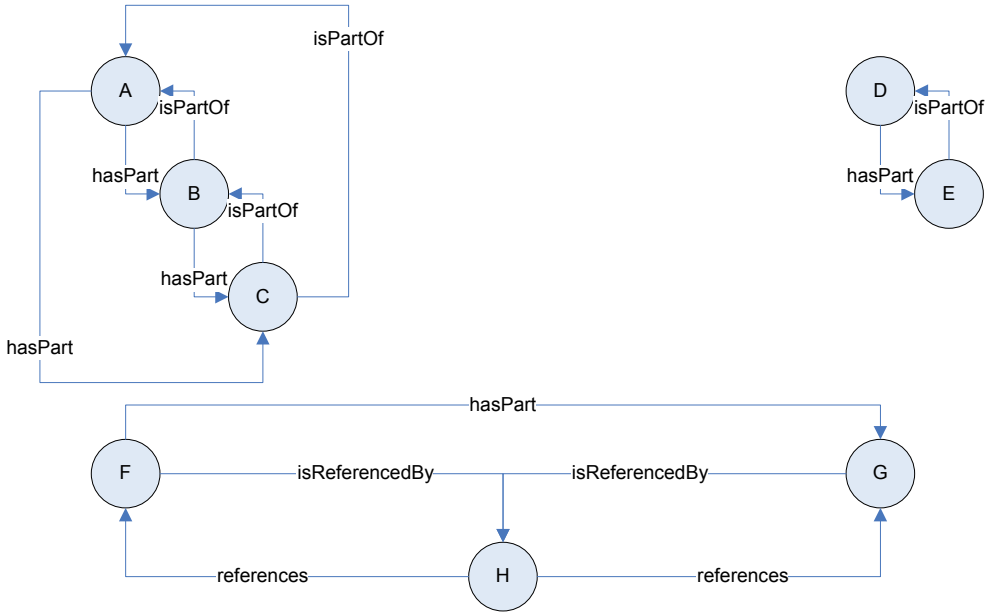

Abbildung 6.3: identifizierte Lernobjektverknüpfungen im Testverzeichnis

deren gespeicherte Schlussfolgerungsketten gelöscht worden. Die aufgrund der gelöschten Lernobjektverknüpfung identifizierten logisch unkorrekten Lernobjektverknüpfungen waren entfernt worden.

Der Test verifizierte das in Kapitel 4 entwickelte Konzept. Jena stellte sich als geeignet heraus, RDF-RDFS-/OWL-Ontologien und eigene Regeln zu verarbeiten.

Kapitel 7

Zusammenfassung

Das Ziel dieser Diplomarbeit war, eine semantische Verknüpfungs- und Retrievalengine für IEEE-LOM-konforme Lernobjekte innerhalb des Hypermedia Learning Object Systems HyLOS zu entwickeln.

Da Lernobjekte mittels benannter Relationen verknüpft werden, wurden die durch den LOM-Standard gegebenen Relationen auf ihren didaktischen Nutzen geprüft. Es stellte sich heraus, dass ihre Semantik einen Einsatz im Lernobjektumfeld erschwert. Durch Konkretisierung oder Interpretation der gegebenen Relationen und Erweiterung wurden zehn Relationspaare oder Relationen mit wohldefinierter Semantik gefunden, die zum Weben eines semantischen Wissensnetzes genutzt werden können.

Ausgehend von diesen Relationen wurden Strategien zum automatischen Identifizieren entwickelt. Diese arbeiten zum einen auf Lernobjektmetadaten, zum anderen auf vorhandenen Lernobjektverknüpfungen. Als besonderes Ergebnis sind die Schlussfolgerungsregeln hervorzuheben, welche durch die Kombination meist zweier Lernobjektverknüpfungen mittels unterschiedlicher Relationen auf eine neue Lernobjektverknüpfung schließen.

Im Rahmen dieser Diplomarbeit wurden Ontologien als ein Konzept der Wissensrepräsentation in maschinenverarbeitbarer Form vorgestellt und zwei Ontologiesprachen kennengelernt. Auf diesem Wissen aufbauend wurde die semantische Verknüpfungsengine konzipiert. Des Weiteren hatten Überlegungen zu verschiedenen Einsatzmöglichkeiten und notwendigen Anforderungen Einfluss auf die Konzeption der Verknüpfungsengine. Das Ergebnis ist eine API, die flexibel in Bezug auf die angesprochene Datenbank und die verwendete externe Schlussfolgerungsmaschine ist. Ebenso können die Persistenzmechanismen zum Speichern von Schlussfolgerungsketten, logisch unkorrekten und vermuteten Lernobjektverknüpfungen leicht ausgetauscht werden. Die Strategien zur Skalierung lassen sich mit dieser API umsetzen.

Die API wurde vollständig implementiert. Einige der definierten Interfaces sind prototypisch umgesetzt worden, so werden Schlussfolgerungsketten in der aktuellen Version in einer Textdatei gespeichert. Die Umsetzung der heuristischen Verfahren und der Strategien zur Skalierung

stehen noch aus. Da Jena als externe Schlussfolgerungsmaschine eingesetzt wurde, konnten alle weiteren Strategien zum automatischen Identifizieren umgesetzt werden.

Inwiefern durch das Verarbeiten der in Lernobjektverknüpfungen enthaltenen Informationen eine textbasierte Suche verbessert oder ersetzt werden kann, wurde aufgrund des kurzen Bearbeitungszeitraumes nicht betrachtet.

Kapitel 8

Ausblick

Die aktuelle Version der Implementierung setzt den allgemeinen Ablauf der Verknüpfungsengine um. Die heuristischen Verfahren sind noch nicht implementiert worden. Als nächster Schritt sollen diese und die Strategien der Skalierung umgesetzt werden. Neben diesen Entwicklungsmöglichkeiten können weitere identifiziert werden.

Die Verknüpfungsengine webt ein Wissensnetz bestehend aus Lernobjekten und benannten Relationen. Dieses Netz kann bei der Suche nach adäquaten Lerninhalten genutzt werden. Die Kombination von traditioneller Volltextsuche und einer relationsverarbeitenden Engine erscheint effektiver als ein Ersetzen der Volltextsuche. Nicht jede benannte Relationen eignet sich in gleicher Weise, bei der Suche verarbeitet zu werden. Da die benannten Relationen unterschiedliche Zusammenhänge zwischen Lernobjekten abbilden, muss untersucht werden, wie relevant eine Relation aufgrund ihrer Semantik für das Auffinden von Lerninhalten ist.

Ein weiterer interessanter Entwicklungsaspekt ist das Setzen von Relationen über Datenbankgrenzen hinweg. Die Verknüpfungsengine wurde flexibel konzipiert, sodass mit geringem Programmieraufwand neben der dem HyLOS-System zugrunde liegenden Datenbank weitere Datenbanken angesprochen werden können. Alle beteiligten Datenbanken müssen nicht denselben Relationssatz unterstützen. Jedoch müssen alle Relationen, die gleich einer in dieser Arbeit definierten Relationen sind und somit von der Verknüpfungsengine verarbeitet werden, die in dieser Arbeit definierte Semantik besitzen. Ansonsten werden Lernobjekte falsch verknüpft. Wie datenbankübergreifende Lernobjektverknüpfungen abgelegt werden und deren Konsistenz gewährleistet werden kann, muss gelöst werden.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Literaturverzeichnis

[ACM98] Association for Computing Machinery: *ACM Computing Classification System*. 1998. – http://www.acm.org/class/1998/

[BLHL01] BERNERS-LEE, Tim ; HENDLER, James ; LASSILA, Ora: The Semantic Web. In: *Scientific American* (2001), May, S. 29–37

[DDC05] Online Computer Library Center, Inc: *Dewey Decimal Classification System*. 2005. – http://www.oclc.org/dewey/

[DOS03] DACONTA, Michael C. (Hrsg.) ; OBRST, Leo J. (Hrsg.) ; SMITH, Kevin T. (Hrsg.): *The Semantic Web*. Wiley, 2003. – ISBN 0–471–43257–1

[Dub98] RUSCH-FEJA, Diann (Hrsg.): *Entwicklungen der Dublin Gore-Metadaten – Bericht über den 5. Dublin Gore Metadata Workshop in Helsinki und einige Bemerkungen über DC-Metadaten in Deutschland*. 1998. – http://bibliotheksdienst.zlb.de/1998/1998_all.htm

[Dub05] HILLMANN, Diane (Hrsg.): *Using Dublin Core – The Elements*. 2005. – http://dublincore.org/documents/usageguide/elements.shtml#relation

[EHLS05] ENGELHARDT, Michael ; HILDEBRAND, Arne ; LANGE, Dagmar ; SCHMIDT, Thomas C.: Ontological Evaluation as an Automated Guide to Educational Content Augmentation. In: ECKSTEIN, Rainer (Hrsg.) ; TOLKSDORF, Robert (Hrsg.): *Berliner XML Tage 2005 – Tagungsband*. Humboldt Universität zu Berlin, Januar 2005

[EHS05] ENGELHARDT, Michael ; HILDEBRAND, Arne ; SCHMIDT, Thomas C.: Automatisierte Augmentierung von Lernobjekten in einer semantischen Interpretationsschicht der HyLOS Plattform. In: HAAKE, Jörg M. (Hrsg.) ; LUCKE, Ulrike (Hrsg.) ; TAVANGARIAN, Djamshid (Hrsg.): *DeLFI 2005 3. Deutsche e-Learning Fachtagung Informatik* Bd. 66, 2005, S. 105–116

[ES03] ENGELHARDT, Michael ; SCHMIDT, Thomas C.: Semantic Linking – a Context-Based Approach to Interactivity in Hypermedia. In: TOLKSDORF, Robert (Hrsg.) ; ECKSTEIN, Rainer (Hrsg.): *Berliner XML Tage 2003 – Tagungsband*. Humboldt Universität zu Berlin, February 2003, S. 55–66

[FKRS01] FEUSTEL, Björn ; KÁRPÁTI, Andreas ; RACK, Torsten ; SCHMIDT, Thomas C.: An Environment for Processing Compound Media Streams. In: *Informatica* 25 (2001), July, Nr. 2, S. 201 – 209

[GHJV01] GAMMA, Erich (Hrsg.) ; HELM, Richard (Hrsg.) ; JOHNSON, Ralph (Hrsg.) ; VLISSIDES, John (Hrsg.): *Entwurfsmuster*. Addison-Wesley, 2001. – ISBN 3–8273–1862–9

[Gru93] GRUBER, Thomas R.: A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition* 5 (1993), Nr. 2, S. 199–220

[HyL05] *The HyLOS Homepage*. 2005. – http://hylos.fhtw-berlin.de

[JEN05] *Jena – A Semantic Web Framework for Java*. 2005. – http://jena.sourceforge.net/

[KS04] KARAMPIPERIS, P. ; SAMPSON, D.: Learning Object Metadata for Learning Content Accessibility. In: *World Conference on Educational Multimedia, Hypermedia and Telecommunications (EDMEDIA)* Bd. 2004. Lugano, Switzerland, 2004. – http://dl.aace.org/16232, S. 5204–5211

[LOM02] Learning Object Meta-Data / IEEE. 2002 (1484.12.1). – Draft Standard. http://ltsc.ieee.org/wg12/20020612-Final-LOM-Draft.html

[MIR05] *The MIR Homepage*. 2005. – http://www.rz.fhtw-berlin.de/content/Projekte/MIR/

[OWL04a] MCGUINNESS, Deborah L. (Hrsg.) ; VAN HARMELEN, Frank (Hrsg.): OWL Web Ontology Language Overview / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-owl-features-20040210/

[OWL04b] BECHHOFER, Sean (Hrsg.) ; VAN HARMELEN, Frank (Hrsg.) ; HENDLER, Jim (Hrsg.) ; HORROCKS, Ian (Hrsg.) ; MCGUINNESS, Deborah L. (Hrsg.) ; PATEL-SCHNEIDER, Peter F. (Hrsg.) ; STEIN, Lynn A. (Hrsg.): OWL Web Ontology Language Reference / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[OWL04c] HEFLIN, Jeff (Hrsg.): OWL Web Ontology Language Use Cases and Requirements / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-webont-req-20040210/

[phi04] *Der Brockhaus - Philosophie*. Mannheim, Leipzig : F.A. Brockhaus, 2004. – ISBN 3–7653–0571–5

[RDF04a] RDF Primer / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[RDF04b] RDF Vocabulary Description Language 1.0: RDF Schema / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[RDF04c] Resource Description Framework (RDF): Concepts and Abstract Syntax / World Wide Web Consortium. 2004. – W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[SSFS99] STEINACKER, Achim ; SEEBERG, Cornelia ; FISCHER, Stephan ; STEINMETZ, Ralf: Multibook: Metadata for Webbased Learning Systems. In: *2nd International Conference on New Learning Technologies*. Bern, Switzerland, 1999. – ftp://ftp.kom.e-technik.tu-darmstadt.de/pub/papers/SSFS99-paper.pdf

[tea03] VIRTUAL CAMPUS TEAM, The: *Architecture and conceptual model of Virtual Campus*. June 2003. – http://www.elet.polimi.it/res/vcampus/download/MCwebVersion.pdf

[URI05] BERNERS-LEE, T. (Hrsg.) ; FIELDING, R. (Hrsg.) ; MASINTER, L. (Hrsg.): Uniform Resource Identifiers (URI): Generic Syntax / IETF. 2005. – RFC 3986

[Wor05] *WordNet – a lexical database for the English language*. 2005. – http://wordnet.princeton.edu/

Anhang A

Schlussfolgerungsregeln

UND	B isPartOf C	B hasPart C
A isPartOf B	A isPartOf C	
A hasPart B		A hasPart C
A isVersionOf B		
A hasVersion B		
A isFormatOf B		
A isReferencedBy B	A isReferencedBy C	
A references B	A references C	
A isBasisFor B	A isBasisFor C	
A isBasedOn B	A isBasedOn C	
A isRequiredBy B	A isRequiredBy C	
A requires B		
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B		
A illustrates B	A illustrates C	
A isIllustratedBy B	A isIllustratedBy C	
A isLessSpecificThan B	A isLessSpecificThan C	
A isMoreSpecificThan B		A isMoreSpecificThan C

Tabelle A.1: Regeln in Verbindung mit *isPartOf/hasPart*

UND	B isVersionOf C	B hasVersion C
A isPartOf B		
A hasPart B		
A isVersionOf B		
A hasVersion B		
A isFormatOf B		
A isReferencedBy B		
A references B		
A isBasisFor B		
A isBasedOn B		
A isRequiredBy B		
A requires B		
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B		
A illustrates B		
A isIllustratedBy B		
A isLessSpecificThan B		
A isMoreSpecificThan B		

Tabelle A.2: Regeln in Verbindung mit *isVersionOf/hasVersion*

UND	B isFormatOf C
A isPartOf B	
A hasPart B	
A isVersionOf B	
A hasVersion B	
A isFormatOf B	
A isReferencedBy B	
A references B	
A isBasisFor B	A isBasisFor C
A isBasedOn B	A isBasedOn C
A isRequiredBy B	A isRequiredBy C
A requires B	A requires C
A isNarrowerThan B	A isNarrowerThan C
A isBroaderThan B	A isBroaderThan C
A isAlternativeTo B	
A illustrates B	A illustrates C
A isIllustratedBy B	A isIllustratedBy C
A isLessSpecificThan B	A isLessSpecificThan C
A isMoreSpecificThan B	A isMoreSpecificThan C

Tabelle A.3: Regeln in Verbindung mit *isFormatOf*

UND	B isReferencedBy C	B references C
A isPartOf B		
A hasPart B	A isReferencedBy C	A references C
A isVersionOf B		
A hasVersion B		
A isFormatOf B		
A isReferencedBy B		
A references B		
A isBasisFor B		
A isBasedOn B		
A isRequiredBy B		
A requires B		
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B		
A illustrates B		
A isIllustratedBy B		
A isLessSpecificThan B		
A isMoreSpecificThan B		

Tabelle A.4: Regeln in Verbindung mit *isReferencedBy/references*

UND	B isBasisFor C	B isBasedOn C
A isPartOf B		
A hasPart B	A isBasisFor C	A isBasedOn C
A isVersionOf B		
A hasVersion B		
A isFormatOf B	A isBasisFor C	A isBasedOn C
A isReferencedBy B		
A references B		
A isBasisFor B		
A isBasedOn B		
A isRequiredBy B	A isBasisFor C	
A requires B		A isBasedOn C
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B	A isBasisFor C	A isBasedOn C
A illustrates B		
A isIllustratedBy B		
A isLessSpecificThan B	A isBasisFor C	
A isMoreSpecificThan B		A isBasedOn C

Tabelle A.5: Regeln in Verbindung mit *isBasisFor/isBasedOn*

UND	B isRequiredBy C	B requires C
A isPartOf B		
A hasPart B		A requires C
A isVersionOf B		
A hasVersion B		
A isFormatOf B	A isRequiredBy C	A requires C
A isReferencedBy B		
A references B		
A isBasisFor B	A isBasisFor C	
A isBasedOn B		A isBasedOn C
A isRequiredBy B	A isRequiredBy C	
A requires B		A requires C
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B	A isRequiredBy C	A requires C
A illustrates B		
A isIllustratedBy B		
A isLessSpecificThan B	A isRequiredBy C	
A isMoreSpecificThan B		A requires C

Tabelle A.6: Regeln in Verbindung mit *isRequiredBy/requires*

UND	B isNarrowerThan C	B isBroaderThan C
A isPartOf B		
A hasPart B		
A isVersionOf B		
A hasVersion B		
A isFormatOf B	A isNarrowerThan C	A isBroaderThan C
A isReferencedBy B		
A references B		
A isBasisFor B		
A isBasedOn B		
A isRequiredBy B		
A requires B		
A isNarrowerThan B	A isNarrowerThan C	
A isBroaderThan B		A isBroaderThan C
A isAlternativeTo B	A isNarrowerThan C	A isBroaderThan C
A illustrates B		
A isIllustratedBy B		
A isLessSpecificThan B		A isBroaderThan C
A isMoreSpecificThan B	A isNarrowerThan C	

Tabelle A.7: Regeln in Verbindung mit *isNarrowerThan/ isBroaderThan*

UND	B isAlternativeTo C
A isPartOf B	
A hasPart B	
A isVersionOf B	
A hasVersion B	
A isFormatOf B	
A isReferencedBy B	
A references B	
A isBasisFor B	A isBasisFor C
A isBasedOn B	A isBasedOn C
A isRequiredBy B	A isRequiredBy C
A requires B	A requires C
A isNarrowerThan B	A isNarrowerThan C
A isBroaderThan B	A isBroaderThan C
A isAlternativeTo B	A isAlternativeTo C
A illustrates B	A illustrates C
A isIllustratedBy B	A isIllustratedBy C
A isLessSpecificThan B	A isLessSpecificThan C
A isMoreSpecificThan B	A isMoreSpecificThan C

Tabelle A.8: Regeln in Verbindung mit *isAlternativeTo*

UND	B illustrates C	B isIllustratedBy C
A isPartOf B		
A hasPart B	A illustrates C	A isIllustratedBy C
A isVersionOf B		
A hasVersion B		
A isFormatOf B	A illustrates C	A isIllustratedBy C
A isReferencedBy B		
A references B		
A isBasisFor B		
A isBasedOn B		
A isRequiredBy B		
A requires B		
A isNarrowerThan B		
A isBroaderThan B		
A isAlternativeTo B	A illustrates C	A isIllustratedBy C
A illustrates B	A illustrates C	
A isIllustratedBy B		A isIllustratedBy C
A isLessSpecificThan B		
A isMoreSpecificThan B	A illustrates C	A isIllustratedBy C

Tabelle A.9: Regeln in Verbindung mit *illustrates/isIllustratedBy*

UND	B isLessSpecificThan C	B isMoreSpecificThan C
A isPartOf B	A isLessSpecificThan C	
A hasPart B		A isMoreSpecificThan C
A isVersionOf B		
A hasVersion B		
A isFormatOf B	A isLessSpecificThan C	A isMoreSpecificThan C
A isReferencedBy B		
A references B		
A isBasisFor B	A isBasisFor C	
A isBasedOn B		A isBasedOn C
A isRequiredBy B	A isRequiredBy C	
A requires B		A requires C
A isNarrowerThan B		A isNarrowerThan C
A isBroaderThan B	A isBroaderThan C	
A isAlternativeTo B	A isLessSpecificThan C	A isMoreSpecificThan C
A illustrates B	A illustrates C	
A isIllustratedBy B	A isIllustratedBy C	
A isLessSpecificThan B	A isLessSpecificThan C	
A isMoreSpecificThan B		A isMoreSpecificThan C

Tabelle A.10: Regeln in Verbindung mit *isLessSpecificThan/isMoreSpecificThan*

1	A isAlternativeTo B ∧ B isFormatOf C ∧ (A isVersionOf B ∨ A hasVersion B) → A isFormatOf C
2	A isBasedOn B ∧ B isBasedOn C ∧ A isNarrowerThan C → A isBasedOn C
3	A isRequiredBy B ∧ B hasPart C ∧ A isBroaderThan C → A isRequiredBy C
4	A requires B ∧ B hasPart C ∧ A isNarrowerThan C → A requires C
5	A isBasisFor B ∧ B hasPart C ∧ A isBroaderThan C → A isBasisFor C
6	A isBasedOn B ∧ B hasPart C ∧ A isNarrowerThan C → A isBasedOn C
7	(A isVersionOf B ∨ A hasVersion B) ∧ A isFormatOf C ∧ B isFormatOf C → A isAlternativeTo B
8	A isFormatOf B ∧ B isFormatOf C ∧ A isFormatOf D ∧ (C isVersionOf D ∨ C hasVersion D) → A isFormatOf C ∧ C isAlternativeTo D
9	A illustrates B ∧ B isBasedOn C ∧ (A isNarrowerThan C ∨ A isBroaderThan C) → A isBasedOn C
10	A isIllustratedBy B ∧ B isBasisFor C ∧ (A isNarrowerThan C ∨ A isBroaderThan C) → A isBasisFor C
11	A illustrates B ∧ B isBasedOn C ∧ (A isNarrowerThan B ∨ A isBroaderThan B) → A isBasedOn C
12	A isIllustratedBy B ∧ B isBasisFor C ∧ (A isNarrowerThan B ∨ A isBroaderThan B) → A isBasisFor C
13	(A isVersionOf B ∨ A hasVersion B) ∧ A illustrates C ∧ B illustrates C → A isAlternativeTo B

Tabelle A.11: N-Schritt-Regeln

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 1. Januar 2006

.....

Dagmar Lange