Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

eLearning in sozialen Netzwerken: Eine Erweiterung von Diaspora um semantische Content-Netze

Nicolas With

Masterarbeit

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Nicolas With

eLearning in sozialen Netzwerken: Eine Erweiterung von Diaspora um semantische Content-Netze

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Schmidt
Zweitgutachter: Prof. Dr. Zukunft

Eingereicht am: 02.04.2015

Nicolas With

Thema der Arbeit

eLearning in sozialen Netzwerken:
Eine Erweiterung von Diaspora um semantische Content-Netze

Stichworte

Semantisches Web, E-Learning, Diaspora, soziales Netzwerk, Metadaten, Ontologie, JENA, Lernobjekte

Kurzzusammenfassung

Die vorliegende Masterarbeit beschäftigt sich mit der Konzeption und Implementierung eines semantischen Content-Netzes in das soziale Netzwerk Diaspora. Es werden zwei Komponenten implementiert. Einerseits eine Komponente für das Verwalten und Organisieren der Lernobjekte und andererseits eine Komponente, die die Lernobjekte über Analyse der Metadaten und die Anwendung von Schlussfolgerungsregeln semantisch verknüpft. Die Relationen werden mit den zugehörigen Lernobjekten in einer Graphdatenbank gespeichert und mit anderen Komponenten, wie Themen, Lerngruppen und Anwendern, verknüpft.

Nicolas With

Title of the paper

eLearning in social networks: Extending Diaspora by semantic content nets

Keywords

Semantic web, eLearning, Diaspora, social network, metadata, ontologies, JENA, learning objects

Abstract

This master-thesis engages in the conception and implementation of a semantic content net within the social network Diaspora. Two components will be implemented. On the one hand a component to manage and organise the learning objects and on the other hand a component to semantically link the learning objects by analysing their metadata and by applying reasoning rules. The relations and the correspondend learning objects will be saved to a graph databases where they will be connected to other components like users, topics and learning groups.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Die sozialen Netzwerke sind ein integraler Bestandteil des alltäglichen Lebens geworden. Sie dienen zur Kommunikation unter Freunden, zur eigenen Unterhaltung oder werden zur Organisation eigener Interessen verwendet. Diese steigende Benutzung für unterschiedliche Tätigkeiten erhöht die Nachfrage nach immer mehr Anwendungen, die in das soziale Netzwerk integriert werden.
Daneben hat sich E-Learning in den letzten Jahren von einem statischen System immer mehr zu einer dynamischen Plattform entwickelt, mit der Lerninhalte präsentiert und zur Verfügung gestellt werden können. *Learning Content Management Systeme* (LCMS) erlauben den Zugriff auf strukturierte Lerninhalte, an denen in Gruppen gearbeitet werden kann. Dabei organisieren moderne LCMS die Inhalte in Lernobjekten, die zusammenhängend ein semantisches Netzwerk formen. In den meisten Systemen gibt es dazu einen Dozenten, der die Gruppen kreiert, die Themen vorgibt und den Lernprozess lenkt. Mit dem Projekt Mindstone[1], welches 2009 gestartet wurde, wird versucht, den Lernprozess und die Lerngruppenbildung zu öffnen, indem ein soziales Netzwerk mit einer Lernplattform verbunden wird. Dies entfernt die Abhängigkeit von einem Dozenten und fördert selbstverantwortliches Lernen. Solch eine Lernumgebung erlaubt den Nutzern einen selbst gesteuerten Lernprozess mit Themen, die ihn persönlich interessieren und Lerngruppen seiner Wahl. Um das zu erreichen, wird das freie soziale Netzwerk Diaspora mit einer E-Learning-Umgebung erweitert. Dazu gehören eine Lerngruppenkomponente und ein semantisches Content-Netz. Ein erster Ansatz dazu wurde von Roreger und Schmidt (2012) gegeben. Die Beseitigung des Dozenten wirft drei Design-Fragen bei der Erstellung solch einer Plattform auf:

1. Wie stimulieren wir den Gruppenbildungsprozess, so dass er effektiv ist für den Lernenden?

2. Wie stellen wir relevanten Inhalt für Lerngruppen und Benutzer zur Verfügung?

[1] http://mindstone.hylos.org

3. Wie bekommen wir einen konsistenten Lernprozess mit Feedback und Korrekturmaßnahmen?

Ein Ansatz zur Beantwortung der ersten Frage wurde in Brauer und Schmidt (2012) und Brauer u. a. (2013) diskutiert. In der vorliegenden Arbeit wird die Frage nach dem zur Verfügung stellen relevanter Inhalte für Lerngruppen und Benutzer behandelt. Vorarbeiten dazu sind das LCMS hylOs (Engelhardt u. a., 2006b) und die darin integrierte Verknüpfungskomponente, die in Engelhardt u. a. (2006a) beschrieben wurde. Die Integration des *hylOs-Reasoners* wurde schon in With (2013) besprochen.
Die Arbeit ist in fünf größere Abschnitte gegliedert. In den ersten beiden Abschnitten wird ein Blick auf die Vorarbeiten geworfen und es werden die nötigen Grundlagen erläutert. Im dritten Abschnitt wird das Konzept aufgestellt. Dabei wird das Konzept der Lerngruppenkomponente kurz erläutert. In dem Hauptteil dieses Abschnitts wird die Konzeption des semantischen Content-Netzes bearbeitet. Dabei werden die Anforderungen an das System gestellt, die in der Implementierung zu erfüllen sind. Die Implementierung ist der vierte Abschnitt und umfasst die Lernobjektverwaltung, welche die Organisation der Lernobjekte übernimmt und die Schnittstelle zum Benutzer darstellt und das Verknüpfungsmodul, welches die Lernobjekte miteinander verknüpft und das semantische Content-Netz letztlich aufbaut. Im letzten Abschnitt wird die fertige Anwendung getestet und die Arbeit noch einmal reflektiert und das Fazit gezogen. Danach wird eine Resümee der vorliegenden Arbeit und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Semantic Web und E-Learning

2.1 Semantic Web

Das Semantic Web ist eine Vision des World Wide Webs, in der alle Daten strukturiert sind und semantisch in eine unterliegende Ontologie geordnet sind. Ziel ist es, ein maschinenlesbares Web zu kreieren, in dem der Computer Informationen automatisch interpretieren und verarbeiten kann. Das Konzept des semantischen Webs stammt unter anderem von Tim Berners-Lee, der es wie folgt definiert hat:

> *„The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.“* (Berners-Lee u. a., 2001)

In 2006 schrieb Berners-Lee, dass diese Idee noch nicht realisiert wurde und das Web immer noch zum größten Teil aus ausschließlich menschenlesbaren Dokumenten besteht und nicht aus maschinenlesbaren Daten, die vom Computer interpretiert werden könnten (Shadbolt u. a., 2006).
Der Trend des Webs ging vor allem in Richtung *„Social Web“*. Wichtige Aspekte davon sind unter anderem das kollaborative Arbeiten in Gemeinschaften und die Erstellung von Inhalten durch den Anwender. Gleichzeitig haben sich Technologien zur dynamischen Inhaltsdarstellung, vor allem javascript-basierende Frameworks, rasant weiterentwickelt während die Technologien zur maschinenverständlichen Beschreibung und Interpretation von Daten vernachlässigt wurden.
Trotzdem gibt es einen stetigen Wandel hin zum semantischen Web. Wer mittlerweile eine Suche bei bekannten Suchmaschinen durchführt und zum Beispiel nach 'Paris' sucht, der merkt, dass die Suchmaschine die Bedeutung des Begriffs als Stadt versteht und die Position der Stadt auf einer Karte einblendet. Ein Beispiel, welches Gruber 2007 noch als Vision des semantischen Webs verwendet hatte. (Gruber, 2007)

Aufbau des semantischen Webs

Im Semantic Web kommen viele verschiedene Technologien und Werkzeuge zum Einsatz, die jeweils eine bestimmte Aufgabe haben und teilweise aufeinander aufbauen. Berners-Lee (2000) hat dies mit einem Schichtenmodell verdeutlicht, welches den theoretischen Aufbau des semantischen Webs skizziert, dargestellt in Abbildung 2.1.

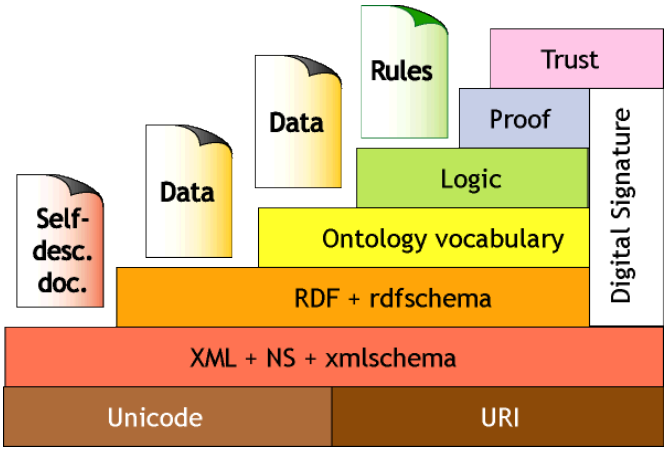

Abbildung 2.1: Das ursprüngliche Schichtenmodell nach Tim Berners-Lee (Berners-Lee, 2000)

Die Grundlage des Semantic Web bilden die Technologien XML und URI. XML ist eine einfache Sprache, mit der sich Daten strukturieren lassen. Die Sprachen der darüberliegenden Ebenen bauen auf der XML-Syntax auf. URIs werden benutzt, um Ressourcen zu identifizieren. Darüber befinden sich weitere Technologien des semantischen Webs, die Schichten sind:

- Die RDF-Ebene, um die Beziehungen der Daten zu repräsentieren
- Die Ontologie-Ebene, welche den formellen Standard über die Bedeutung der Daten aufstellt
- Die Logik-Ebene, welche intelligente Schlussfolgerung mit den strukturierten Daten ermöglicht

Im Gegensatz zur RDF-Ebene, in der sich die verwendete spezifizierte Technologie im Namen wiederfindet, gab es für die beiden anderen Schichten noch keine standardisierte Sprache, weswegen dort allgemein von Ontologie- und Logikebene gesprochen wurde.
Die abschließenden Schichten sind die Prüfungs- und Vertrauensschicht (*Proof* und *Trust*).

Die Prüfungsschicht soll prüfen, ob die in der Logikschicht erstellten Schlussfolgerungen korrekt sind. Dazu muss überprüft werden, wie die Schlussfolgerungen gemacht wurden und von welchen Daten die Schlussfolgerungen erstellt wurden. Die Vertrauensschicht überprüft hingegen, von wem die Daten stammen und ob der Anbieter dieser Daten vertrauenswürdig ist. Ein Schritt, um dies zu bewerkstelligen, sind digitale Signaturen (*digital signatures*), welche die vertikale Schicht in dem Modell darstellen. So ist es möglich, Statements oder Dokumente zu signieren, um zu zeigen, dass eine bestimmte Person diese verfasst hat. Wenn nun ein Nutzer dieser Person vertraut, können seine signierten Daten verwendet werden. (Swartz und Hendler, 2001)
Eine Erweiterung davon wäre das *"Web of Trust"*, bei dem auch den Personen vertraut wird, denen die vertraute Person vertraut. Dies kann jedoch im Falle des bedingungslosen Vertrauens auch zu Schwierigkeiten führen, da das „Vertrauensnetz" schnell über eine unüberschaubare Größe wachsen kann und Personen indirekt ins Vertrauen gezogen werden, denen in direkter Linie nicht vertraut werden würde. Als Abhilfe können auch verschiedene *Trust*- oder *Distrust*-Level vergeben werden, so dass bestimmten Personen oder Dokumenten mehr vertraut, anderen nur wenig vertraut werden kann oder diese direkt als falsch oder nicht vertrauenswürdig abgestempelt werden können. Eine Implementation von Trust-Levels im semantischen Web haben Golbeck u. a. (2003) in ihrem Bericht beschrieben.

In den Jahren ist das Schichtenmodell weiter verfeinert und auch erweitert worden. In Abbildung 2.2 wird das Schichtenmodell aus einer Präsentation von Steve Bratt[1] aus dem Jahr 2007 gezeigt.

Zu erkennen ist, dass weitere standardisierte Sprachen hinzu gekommen sind. So zum Beispiel SPARQL als Abfragesprache für RDF-Dokumente, OWL als Beschreibungssprache von Ontologien und die Regelbeschreibungsspezifikation RIF (Rule Interchange Format). RDF-Schema wurde beibehalten und ist in dem neuen Modell von RDF getrennt und zwischen RDF und OWL gelegt worden. Eine genauere Beschreibung dieser Technologien kann in Kapitel 3.1 nachgelesen werden. Diese Sprachen sind jedoch nur eine Empfehlung des W3C, von dem dieses Modell stammt, und somit können auch andere Spezifikationen benutzt werden.
Die Schicht *Digital Signature* wurde ersetzt und erweitert durch die *Crypto*-Schicht, welches ein größeres Aufgabengebiet abdeckt. So geht es nicht mehr nur um digitale Signaturen, um die Authentizität von Dokumenten sicherzustellen, sondern vermehrt

[1] Steve Bratt war von 2002 bis 2008 Chief Executive Officer des W3C

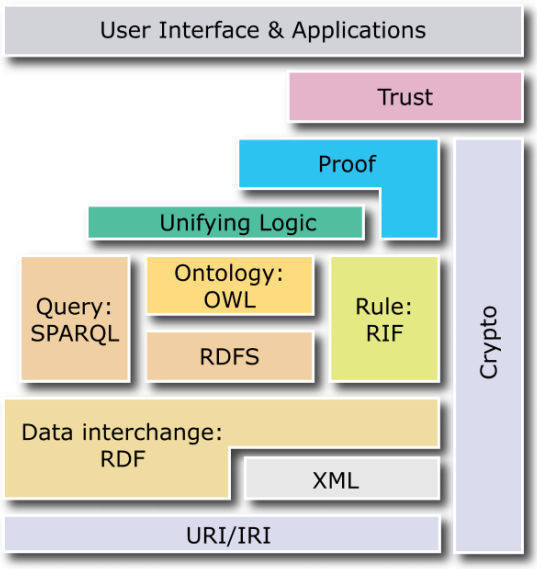

Abbildung 2.2: Eine überarbeitete Version des Schichtenmodells von Bratt (2007)

auch um den sicheren Transport von Daten. Der Begriff *„Crypto“* steht für kryptografische Tools, die eine Reihe von Technologien zur Verschlüsselung und Authentifizierung zum sicheren Austausch von Daten abdecken, so zum Beispiel TLS[2] (Transport Layer Security) und zertifikatbasierte Authentifizierung. (Harth u. a., 2011)
Innerhalb der *Logic-*/*Proof*- und *Trust*-Ebenen haben sich auch zur Zeit der vorliegenden Ausarbeitung noch keine standardisierten Sprachen oder Technologien durchgesetzt.
In dieser Arbeit wird der Schwerpunkt vor allem auf die RDF-/Ontologie- und Logikschicht gelegt. Für die RDF- und Ontologieschicht werden die standardisierten Sprachen verwendet. Für die Schlussfolgerungsengine wird das Framework JENA[3] verwendet, das es ermöglicht, logische Schlussfolgerungen aus formal aufgestellten Ontologien herzuleiten.

2.1.1 Social Semantic Web

Der Begriff Social Semantic Web bezeichnet ein Konzept zur Erstellung, Bearbeitung und Verteilung von Informationen durch die Verbindungen der Technologien und Ansätze vom Semantic und dem Social Web. Mit dem Semantic Web wird versucht, Daten eine Bedeutung zu geben und maschinenlesbar zu machen, unter anderem durch die Benutzung von Ontologien und Metadaten. Hingegen ist der Ansatz des Social Webs

[2] http://tools.ietf.org/rfcmarkup/5246
[3] https://jena.apache.org/

eine Plattform zu erstellen, in der Benutzer sich treffen, miteinander interagieren und gemeinsam Content erstellen und verbreiten können über, zum Beispiel, Wikis, Blogs und Photo-/Videosharing-Services. (Torniai u. a., 2008)
Eine frühe Form dieses Konzepts haben Cahier und Zacklad (2004) diskutiert. Sie vertraten die Auffassung, dass die Idee, den Daten im Web eine maschinenlesbare Bedeutung zu geben, ein guter Ansatz ist. Jedoch sollte der Mensch nicht aus diesem Prozess ausgeschlossen, sondern in den gesamten Prozess miteinbezogen werden. In ihrem Aufsatz haben sie dieses Konzept als **„Socio-Semantic Web“** betitelt. In einem späteren Artikel erläuterten sie die Möglichkeiten des „Socio-Semantic Web“ wie folgt:

> *„“Socio-semantic Web“ applications make it possible for communities to co-produce and use symbolic organizational artefacts, such as “maps”, “tag clouds” or shared indexes making the collective knowledge and actions/activities both more visible and more reflexive.“* (Cahier u. a., 2007)

Einen ähnlichen Ansatz hatten auch Schoop u. a. (2006) beschrieben, die von einem **„Pragmatic Web“** sprachen. Ein Schwerpunkt der Diskussion war, dass Ontologien, welche ein wichtiger Bestandteil des semantischen Webs sind, nicht starr sein dürften, sondern sich kontinuierlich mitentwickeln müssten und dies in einer Gemeinschaft abzusprechen sei. Somit war es die Vision des *Pragmatic Webs* das semantische Web zu erweitern *„by improving the quality and legitimacy of collaborative, goal-oriented discourses in communities.“*. Im Vergleich zu Cahier und Zacklad, ist die Vision des Pragmatic Web eher konkret auf die Instandhaltung und kontinuierliche Weiterentwicklung von fachspezifischen Ontologien gerichtet und deckt damit eher nur einen kleinen Teil der Möglichkeiten des Social Semantic Web ab.
Gruber (2007) wiederum griff den Begriff „collective knowledge“ auf, der schon in der Arbeit von Cahier und Zacklad diskutiert wurde und fokussierte seine Recherche auf das Social Semantic Web zur Wissensakquirierung und zur Unterstützung von E-Learning. Er erläuterte, dass das Social Web eine einfache Kommunikation zwischen Menschen ermögliche und Kommunikation einen wichtigen Grundpfeiler darstelle, um neues Wissen zu erlangen. Das semantische Web ermöglicht es Maschinen, Daten zu sammeln und zu strukturieren, damit einerseits die Daten eine klare, global definierte Bedeutung bekommen, andererseits diese Daten unkompliziert zwischen verschiedenen Systemen geteilt werden können. Gruber sieht in dem Social Semantic Web eine Synergie zwischen Menschen und Maschinen. Seiner Ansicht nach sind Menschen die *producer* und *customer*, denn sie sind die Quelle des Wissens und benutzen dieses Wissen für ihr Interesse und zur Lösung ihrer Probleme. Maschinen dagegen sind die *enablers*, das heißt, sie sammeln,

vernetzen und speichern Daten zur leichteren Weiterverarbeitung (Gruber, 2007). Diese Konzepte hat Gruber in Beispielsystemen verdeutlicht, die er als *„collective knowledge systems“* betitelte. Ein solches System wird in Kapitel 2.3.4 behandelt.

2.2 Ontologien

Der Begriff Ontologie (griechisch: die Lehre des Seins) kommt ursprünglich aus der Philosophie und bezeichnet dort eine Disziplin, in der es um die Struktur der Wirklichkeit und die Einteilung grundlegender Typen von Entitäten und deren strukturellen Beziehungen geht. Dieser Zweig der Philosophie ist weitgehend auch als „allgemeine Metaphysik“ bekannt. Ein Begriff, der auf eine Ansammlung von Texten Aristoteles' mit dem Titel *Metaphysik*[4] zurückgeht, in denen er sich mit diesem Gebiet als einer der ersten auseinandergesetzt hatte.
Der Begriff wurde in die Informatik übernommen und bedeutet dort ein formales Modell einer Wissensstruktur mit dessen relevanten Entitäten und Beziehungen (Guarino u. a., 2009). In einem anderen wissenschaftlichen Aufsatz von Tom Gruber wird die Ontologie, wie sie im Bereich der Informatik benutzt wird, definiert als eine *„explizite, formale Spezifikation einer Konzeptualisierung“* (Gruber, 1993). Dabei wird der Ausdruck Konzeptualisierung definiert als ein Versuch einer einfachen, abstrakten Abbildung der Realität. Laut Gruber wird eine Ontologie durch eine Menge von Begriffen und Relationen definiert, mit dem die Objekte und deren Beziehungen untereinander in dem zu betrachtenden Gegenstandsbereich repräsentiert werden, sowie formale Axiome, welche die Interpretation und die richtige Benutzung dieser Begriffe bestimmt. (Gruber, 1993) Guarino und Giaretta (1995) haben diese Definition von Gruber in Frage gestellt, genauer die Bedeutung des Begriffs der Konzeptualisierung. Unter Gruber stellt eine Konzeptualisierung nur eine Abbildung einer einzigen Gegenstandslage dar. Jedoch zeigen Guarino und Giaretta, dass eine Konzeptualisierung besser definiert ist als eine Art konzeptuelles Raster, welches auf viele Gegenstandslagen angewendet werden kann, wenn dasselbe Vokabular benutzt wird.

In einem Beispiel, dargestellt in Abbildung 2.3, werden zwei Gegenstandslagen abgebildet. Beide mit den Objekten *{a, b, c, d, e}* und den Relationen *{on, above, clear, table}*. Bei der ersten Definition von Gruber würden der linke und der rechte Tisch zwei unterschiedliche Konzeptualisierungen zeigen, da es sich um unterschiedliche Sachlagen

[4] Der Begriff stammt nicht von Aristoteles selbst, sondern wurde vermutlich von Andronikos von Rhodos geprägt, der diese Texte zusammen getragen hatte.

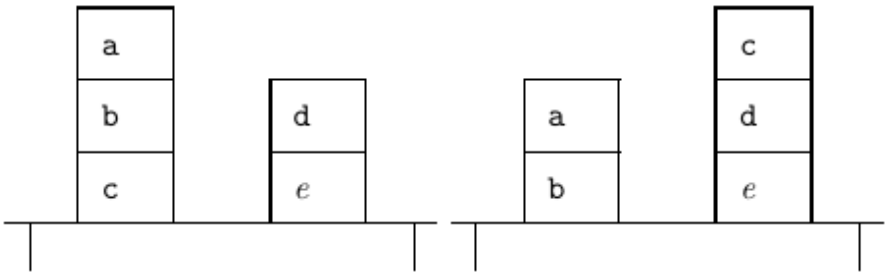

Abbildung 2.3: Ein Beispiel anhand einer Blockwelt, mit den Objekten {a,b,c,d,e} in zwei unterschiedlichen Gegenstandslagen (Guarino und Giaretta, 1995)

handelt (*a on b* auf dem linken Tisch und *c on d* auf dem rechten). Nach der Definition von Guarino und Giaretta würde jedoch von der gleichen Konzeptualisierung gesprochen werden, die aber auf verschiedene Sachlagen angewendet wird.
Darauf aufbauend hat Guarino seine eigene Definition einer Ontologie abgeleitet. In einem Bericht von 1998 beschrieb er eine Ontologie als

> *„(...) a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models.“* (Guarino, 1998)

Abbildung 2.4 verdeutlicht den Zusammenhang zwischen einer Konzeptualisierung, der Sprache und einer Ontologie. Zu beachten ist, dass eine Konzeptualisierung immer sprachunabhängig ist. Eine Ontologie ist dagegen sprachabhängig und versucht durch aufgestellte Axiome sich dem beabsichtigten Modell anzunähern, welches durch die benutzte Sprache am besten die Konzeptualisierung beschreibt. Wenn eine Ontologie das gedachte Modell gut approximiert, wird von einer *„good ontology“*, ansonsten von einer *„bad ontology“* geredet.

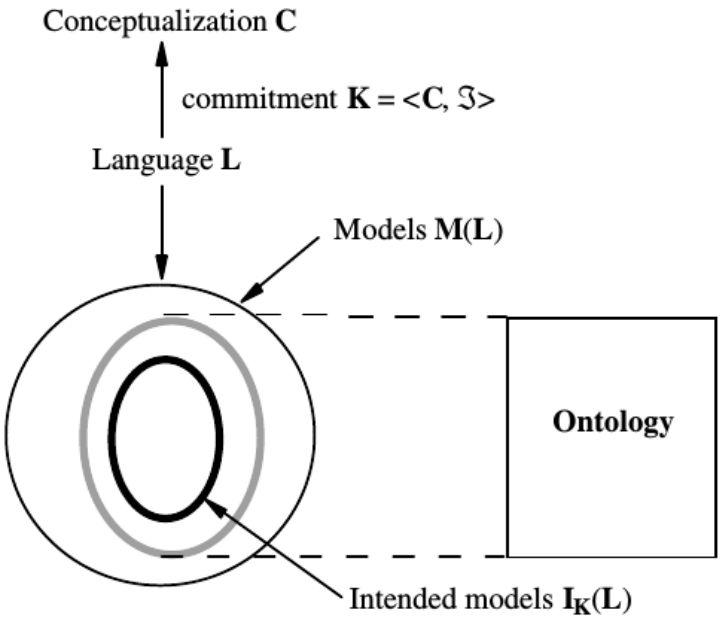

Abbildung 2.4: Definition einer Ontologie im Zusammenhang mit der Konzeptualisierung (Guarino, 1998)

2.2.1 Ontologiebeispiel

Um das Prinzip einer Ontologie besser zu erläutern, wird in Abbildung 2.5 ein Beispiel gegeben. Der Gegenstandsbereich, der hier illustriert werden soll, ist die Beziehung zwischen Musikern und der Musik mit den dazugehörigen Relationen und Begriffen.

In Abbildung 2.5 sind Begriffe durch Ellipsen dargestellt. Die Pfeile symbolisieren eine Beziehung zwischen zwei Begriffen, dabei zeigen die dicken Pfeile eine Vererbungsbeziehung an. Die Rechtecke enthalten Objekttypen. Die abgebildete Ontologie gibt an, dass Musiker Künstler sind, welche wiederum eine Unterklasse von Menschen sind. Mensch und Künstler haben jeweils Typen zugewiesen bekommen, im Falle vom Menschen sind dies Vorname und Nachname, im Falle des Künstlers der Künstlername. Durch die Vererbung besitzen die vererbten Objekte auch jeweils diese Typen, das heißt, Musiker und Komponisten haben einen Vornamen, Nachnamen und Künstlernamen.
Die Beziehung zwischen Komponisten und Musikwerken besagt, dass Komponisten Musik komponieren und auch, dass Musik von Komponisten komponiert wird. Diese doppelseitige Beziehung wird eine *inverse* Beziehung genannt und ist später für die logische Ableitung von Schlussfolgerungen wichtig. Darüber hinaus schränkt diese Beziehung die

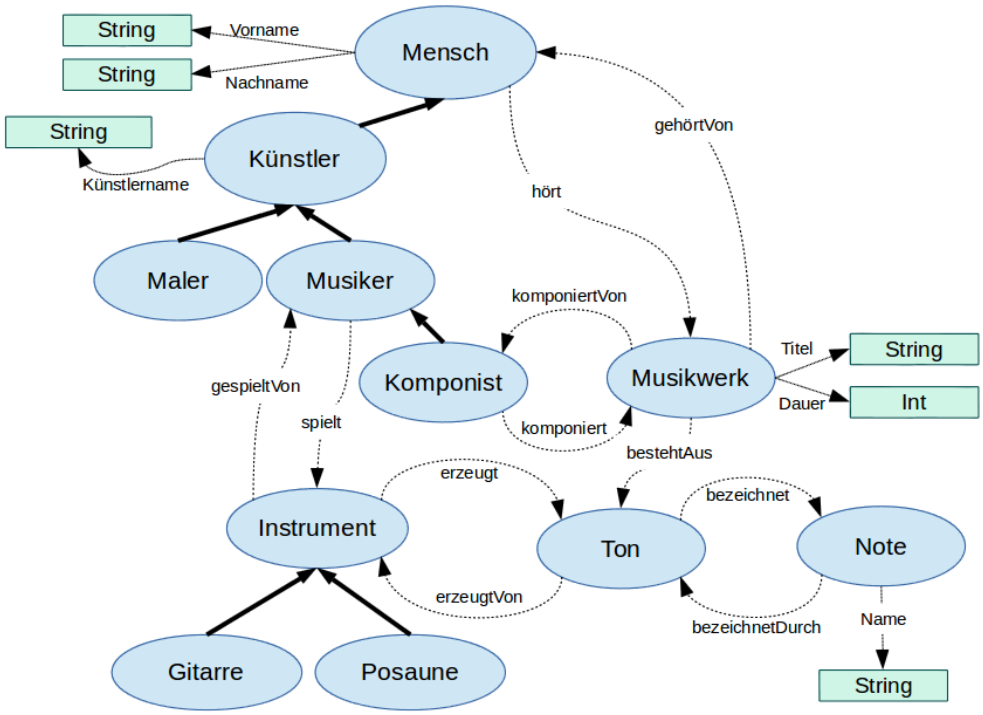

Abbildung 2.5: Eine Beispielontologie im Bereich der Musik mit deren Klassifizierungen und Beziehungen

Begriffe ein. Ein Musikwerk kann nicht von jedem Musiker oder Künstler erzeugt werden, sondern nur von Komponisten. In die anderer Richtung werden die Beziehungen jedoch vererbt, zum Beispiel wird Musik von Musikern gehört, da Menschen Musik hören.
Die ontologische Darstellung dieses Gegenstandsbereichs könnte natürlich auch anders dargestellt werden. Es könnten Begriffe weggelasssen oder die Ontologie an anderen Stellen erweitert werden. Zum Beispiel könnte zwischen Note und Komponist die Beziehung *schreibt* und in anderer Richtung die Beziehung *geschriebenVon* oder weitere Unterbegriffe zur Musik hinzugefügt werden. Es würde immer noch der gleiche Bereich abgebildet werden, aber mit einer anderen Ontologie. Ob der Bereich gut abgebildet wurde und das gewollte Modell der Realität zeichnet, ist daher eine subjektive Entscheidung.

Ausgehend von dieser Ontologie können nun Instanzen erzeugt werden, die konkrete Objekte oder Personen darstellen. Zum Beispiel könnte eine Instanz des Begriffs Musiker ein Objekt mit der URI *http://www.musiker.de/J_S_Bach* und dem Vornamen *Johann Sebastian* und dem Nachnamen *Bach* sein, der ein Musikstück komponiert hat mit der

URI *http://www.musikstueck.de/Air* und dem Titel *Air Suite No.3* und einer Dauer von *290* Sekunden. So könnte eine Abbildung der Realität aufbauend auf der Ontologie erzeugt werden.

2.3 E-Learning

Sehr allgemein gefasst ist der Begriff E-Learning definiert als Lernen mit Hilfe von elektronischen oder digitalen Medien. Der Begriff E-Learning ist an sich kein wissenschaftlicher Begriff, sondern eher ein Modewort, welches erst in den 90ern mit dem Aufkommen des World Wide Webs an Popularität gewann. Im wissenschaftlichen Kontext wird unter anderem von *Computer Based Training* (CBT), *Computer Aided Instuction* (CAI) oder *Computer Aided Learning* (CAL) gesprochen (Ehlers, 2011). Mittlerweile hat sich der Terminus E-Learning etabliert, nicht zuletzt, weil er in einer Reihe steht mit Begriffen wie E-Mail oder E-Commerce und damit eine Beziehung zum Internet ausdrückt, die in dem Bereich immer stärker wird.
Da es sich um eine modische Wortschöpfung handelt, die den Aufgabenbereich nicht sehr konkret widerspiegelt, ist es schwierig, eine einheitliche Definition des Begriffs zu finden. In einem Bericht der Delphi Group über die Vorzüge von E-Learning wird es definiert als:

> *„(...) just-in-time education integrated with high velocity value chains. It is the delivery of individualized, comprehensive, dynamic learning content in real time, aiding the development of communities of knowledge, linking learners and practitioners with experts.“* (Delphi Group, 2000)

Weiter wird in dem Artikel behauptet, dass E-Learning den nächsten Evolutionsschritt in der Didaktik darstellt. Es wird verglichen mit der Entwicklung der Industrie, in der durch Fließbänder und Warenlieferung die Arbeit zum Arbeiter gebracht wurde. Das Prinzip von E-Learning ist es, das Wissen und die Werkzeuge, welches benötigt wird, um die Arbeit auszuführen, zum Arbeiter zu bringen, egal wo oder wer sie sind. Um die Unterschiede und die Vorteile des E-Learnings besser aufzuzeigen, wurde das traditionelle Lernen, also das Lernen in einer Institution, dem Prinzip des E-Learnings gegenübergestellt, aufgeführt in Tabelle 2.1.

Die Charakteristiken, die in der Tabelle beschrieben wurden, sind nicht maßgeblich für alle Formen des E-Learnings. Vielmehr erfüllen verschiedene Formen des E-Learnings jeweils andere Funktionen. Zum Beispiel gibt es bei einfacher Lernsoftware, in denen

Dimensionen	**tradionelles Lernen**	**eLearning**
Delivery	Push – Lehrer bestimmt Agenda	Pull – Lernender bestimmt Agenda
Responsiveness	Anticipatory – das dargestellte Problem wird vorausgesetzt	Reactionary – reagiert auf ein aktuelles Problem
Access	Linear – ein vordefinierter Lernprozess	Non-Linear – Kann Lernprozess selbst bestimmen
Symmetry	Asymmetric – Lernen ist eine separate Aktivität	Symmetric – Lernen wird zu einer integrierten Aktivität
Modality	Discrete – Lernen in Blöcken	Continouous – Lernen als Strom
Authority	Centralized – Content wird nur von Pädagogen bereitgestellt	Distributed – Content kommt durch Interaktion von Pädagogen mit Anderen
Personalization	Mass Produced – Content muss alle gleichermaßen zufrieden stellen	Personalized – Content ist persönlich zugeschnitten
Adaptivity	Static – Content ändert sich nur langsam	Dynamic – Ständig wechselnder, aktualisierter Content

Tabelle 2.1: Gegenüberstellung von eLearning und traditionellem Lernen (Delphi Group, 2000)

die Benutzer eine festgelegte Reihenfolge von Lektionen durcharbeiten müssen, keinen individuell angepassten, dynamischen Lerninhalt. Was jedoch als definierendes Merkmal eines E-Learning-Systems bezeichnet werden kann, ist die Orts- oder Zeitunabhängigkeit des Lernens.
Einen anderen Fokus in der Definition setzte Ehlers (2011) in dem Buch „Qualität im E-Learning aus Lernersicht“, wo er E-Learning beschrieb als:

> *„(...) eine Lernform, bei dem die neuen Informations- und Kommunikationsmedien (Computer und Internet) in Lernarrangements eingebunden werden, entweder zur Unterstützung des Lernprozesses (als sogenannte „hybride“ Lernarrangements) oder als ausschließliche Form der Vermittlung.“* (Ehlers, 2011)

Anders als in der Arbeit der Delphi Group, wo der gelieferte Content und die Lern-Communities in den Mittelpunkt gestellt wurden, betont Ehlers in seiner Definition die Lernumgebung und die elektronischen Medien als Rahmen des Lernprozesses. Wobei er in seinem Buch den Begriff E-Learning als paradox hinstellt, da der Lernprozess als

solches nicht elektronisch ist, sondern nur die Lieferung des Inhalts, weswegen aus seiner Sicht der Begriff E-Teaching treffender wäre.
Zusammengefasst beschreibt der Begriff E-Learning ein sehr breites Spektrum von elektronisch unterstützten Lernarrangements. Von kleinen Lernsoftwares, bis hin zu großen Learning Management Systemen fällt alles unter diesen Begriff. Dabei können sich diese Systeme sehr fein in ihrem grundlegenden Konzept unterscheiden und mit vielen verschiedenen Funktionen ausgestattet sein.
Im Folgenden wird das Konzept eines Learning Content Management Systems dargestellt, welches auch am besten die Anwendung beschreibt, die in dieser Arbeit behandelt wird.

2.3.1 Learning Content Management System

Das Learning Content Management System (kurz: LCMS) hat sich aus einer Verbindung zwischen Learning Management Systemen (LMS) und Content Management Systemen (CMS) heraus entwickelt.
LMS sind Lernplattformen, die es erlauben, komplette Onlinekurse anzubieten und zu verwalten. Es werden Lerninhalte bereitgestellt, die vom Lernenden abgearbeitet und vom Lehrenden organisiert werden können. Dabei bietet es auch Funktionen an, die die Kommunikation zwischen dem Lernenden und Lehrenden ermöglicht. Ein Beispiel für ein LMS ist die Open-Source Plattform Moodle[5]. Dessen Konzept ist es, Kursräume anzubieten, in der sich Benutzer anmelden können, um an diesem Kurs teilzunehmen, welcher von einem Lehrer organisiert und überwacht wird.
Als grundlegende Orientierung können sechs Bereiche angegeben werden, die den Funktionsumfang einer Lernplattform charakterisieren. Dabei müssen nicht alle Funktionen zwingend vorhanden sein: (Schulmeister, 2005, S. 12)

- Benutzerverwaltung
- Kursverwaltung
- Rollen- und Rechtevergabe
- Kommunikationsmethoden zwischen Lerneden und Lehrern
- Werkzeuge für das Lernen (interaktives Whiteboard, Notizbuch, Annotationen, Kalender...)
- Darstellung der Kursinhalte, Lernobjekte und Medien über einen Browser

[5] https://moodle.org/

Während LMSes vor allem Werkzeuge für die virtuelle Lernorganisation darstellen, haben CMSes die Erstellung und Organisation von Inhalten als zentrale Aufgabe (Baumgartner und Kalz, 2004). Der Content wird häufig in einem zentralen Repository innerhalb wiederverwendbarer Lernobjekte gespeichert. Somit können ein und derselbe Content aus verschiedenen Kursen referenziert werden. Baumgartner und Kalz (2004) zählen 7 zentrale Funktionen eines Content Management Systems auf:

- Beschaffung und Erstellung von Inhalten
- Präsentation und Publikation von Inhalten
- Aufbereitung und Aktualisierung von Inhalten
- Management und Organisation von Inhalten
- Verteilung und Integration von Inhalten
- Verarbeitung von Inhalten (Workflow)
- Wiederverwendung von Inhalten

Die Erstellung, Organisation und Speicherung von Lerninhalten wird durch ein Autorenwerkzeug unterstützt. Dabei wird versucht, dem Benutzer die Erstellung so einfach wie möglich zu gestalten, so dass er keine Kenntnisse über fortschrittliche Computertechnologien haben muss. Oft gilt im E-Learning-Bereich, dass ein Autorensystem auf eine bestimmte Lernplattform abgestimmt ist, so dass nur Content für dieses System erstellt werden kann. Dies ist auf unterschiedliche Content-Modelle zurückzuführen, die den Lernplattformen zugrunde liegen. Es gibt jedoch auch Möglichkeiten Content auf mehreren System anzuwenden und es zwischen Lernplattformen auszutauschen, indem auf diesen Systemen dasselbe Referenzmodell für den Content verwendet wird. Ein solches Modell bietet SCORM[6] (**S**harable **C**ontent **O**bject **R**eference **M**odel), welches eine Sammlung von Standards und Spezifikationen anbietet. Unter anderem wird festgelegt, wie Lerninhalte strukturiert und in zip-Dateien komprimiert werden sollten, um es zwischen den Systemen unkompliziert austauschen zu können.
Ein LCMS verbindet nun diese beiden Konzepte und bietet - neben der Erstellung und Verwaltung von Content über ein Autorenwerkzeug - eine Lernplattform an, in dem Benutzer diese einzelnen Lernbausteine für größere Lerneinheiten, Kurse, Präsentationen oder Dokumentationen einsetzen können.

[6] http://www.adlnet.org/scorm/

2.3.2 Lernobjekte

Für den Begriff Lernobjekt gibt es mehrere, unterschiedliche Versuche einer Definition. Der Begriff *„Learning Objects“* wurde von dem *Learning Technology Standards Committee* (LTSC) ausgewählt, wahrscheinlich übernommen von der *CEdMA* (**C**omputer **Ed**ucation **M**anagement **A**ssociation) *working group* mit dem Namen *„Learning Architectures, API's, and Learning Objects“*, welche von Wayne Hodgins 1994 gegründet wurde (Wiley, 2000a, S. 4). Das LTSC gründete die *Learning Object Metadata Working Group*, die in ihrem finalen Entwurf für die Konzeption eines Learning Object Metadata (LOM) Standard den Term *Learning Objects* wie folgt definiert haben:

> *„(...) a learning object is defined as any entity, digital or non-digital, that may be used for learning, education or training.“* (Learning Object Metadata working group, 2002)

Bevor diese Definition in dem Bericht der LOM Working Group aufgestellt wurde, hat sich David Wiley in seiner Dissertation mit dem Design von Lernobjekten auseinandergesetzt und im Rahmen dieser eine eigene Definition ausgearbeitet:

> *„any digital resource that can be reused to support learning.“* (Wiley, 2000b)

Es sind zwei wichtige Unterschiede in den beiden Definitionen auszumachen. Erstens spricht Wiley nur von digitalen Ressourcen. Er vertritt damit die Ansicht, dass nur in einem digitalen Umfeld von Lernobjekten gesprochen werden sollte, während in dem LOM-Entwurf auch nicht-digitale Dinge, wie zum Beispiel Bücher, in die Kategorie der Lernobjekte fallen würden. Zweitens betont Wiley die Wiederverwendbarkeit der Lernobjekte. Das bedeutet, dass Lernobjekte in sich geschlossen sein sollten, um in möglichst vielen, unterschiedlichen Kontexten benutzt werden zu können. Dieses Konzept ist aus der objektorientierten Programmierung entlehnt, in der die Wiederverwendbarkeit der Objekte wichtig und ein wesentlicher Bestandteil vieler Design Patterns ist. (Gamma u. a., 1994; Wiley, 2000b, S. 2) Aus der Definition von Wiley kann auch geschlossen werden, dass erst die Wiederverwendbarkeit einer Ressource oder Information in einem Lernkontext dieses zu einem Lernobjekt macht. Daher wird unter Umständen auch von *„Reusable Learning Objects“* (kurz: RLO) gesprochen.
So gesehen ist die Definition der LOM Working Group breiter gefasst, Wileys Definition ist dagegen präziser und liefert eine ideelle Sicht auf Lernobjekte. Wenn in Betracht gezogen wird, wie Lernobjekte in der Lernplattform, welche in der vorliegenden Arbeit beschrieben wird, verwendet werden, ist Wileys Definition zutreffender.

Lernobjekte werden in LCMS eingesetzt, um Informationen zu speichern und wiedergeben zu können. Wie genau die Lernobjekte in einer Lernplattform aufgebaut und eingesetzt werden kann, wird in einem Content-Modell skizziert. Ein Beispiel eines solchen Modells ist das "Autodesk[7] Content Model", dargestellt in Abbildung 2.6.

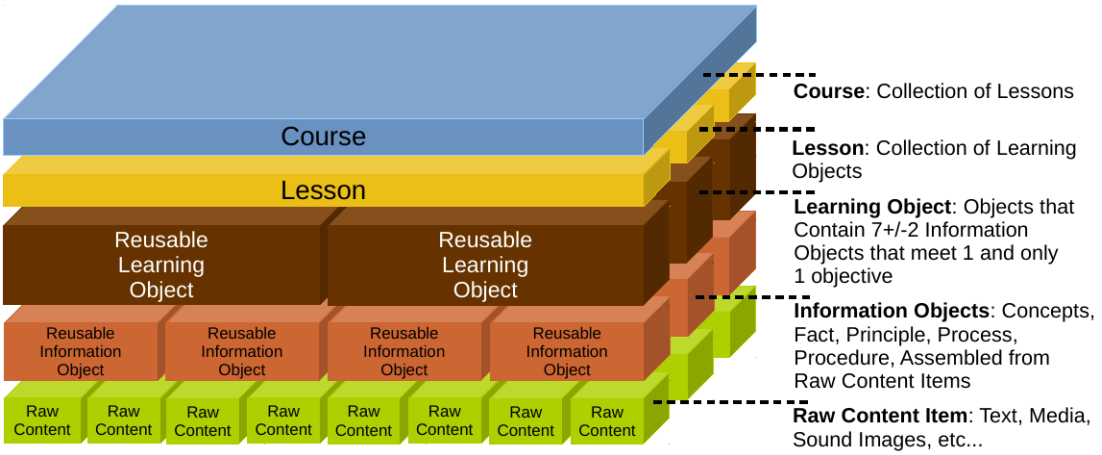

Abbildung 2.6: Darstellung eines Content-Modells für ein LCMS mit wiederverwendbaren Lernobjekten (S3 Working Group, 2002)

Die unterste Schicht bilden die rohen Daten, wie Text- oder Mediendaten. Mehrere dieser Daten können in einem *Reusable Information Object* zusammengefasst werden, die zur Klärung eines Sachverhalts dienen kann. Eine Sammlung von mehreren Informationsobjekten lassen sich in einem *"Reusable Learning Object"* zusammenfassen, die dann exakt ein Lernziel unterstützen. Aus mehreren wiederverwendbaren Lernobjekten lässt sich eine *"Lesson"* zusammenstellen, woraus sich wiederum ein *"Course"* bauen lässt, welches auch gleichzeitig die abstrakteste Ebene in dem Modell darstellt.
Dieses Modell verdeutlicht sehr gut die Idee eines Schichtenmodells, wie es häufig in E-Learning-Szenarien anzutreffen ist. Jede Schicht bedeutet eine neue Abstraktionsebene, angefangen mit einem sehr konkreten Datenobjekt, welches genau ein Element umschließt. Jede weitere Ebene ist eine Schachtelung des „unterliegenden" Objekts und fügt unter Umständen noch weitere Attribute hinzu, wie zum Beispiel ein Lernziel oder andere Beschriftungen.
Baumgartner und Kalz (2005) haben an diesem Modell jedoch kritisiert, dass hiernach jedes Lernobjekt genau einem Lernziel dient und für dieses optimiert wurde. So wird

[7] Autodesk, Inc. ist ein Unternehmen, welches vor allem für die AutoCAD Software bekannt ist. Sie bieten auch Lernsoftware speziell für ihre Produkte an (siehe http://www.autodesk.com/education/home).

für ein neues Lernziel ein entweder leicht modifiziertes oder auch ein radikal verändertes Lernobjekt benötigt. Das Dilemma, welches Baumgartner sieht, ist, dass das Lernobjekt einerseits kontextfrei bleiben muss, um in möglichst vielen Szenarien verwendet werden zu können. Andererseits braucht das Lernobjekt aus didaktischer Sicht einen Kontext, um möglichst effizient in einem Szenario benutzt zu werden. Er plädiert dafür das **fachliche** (*Technical Object*) von dem **didaktischen** (*Educational Object*) Objekt zu trennen. Diese beiden Teilobjekte sind wiederverwendbar und ergeben zusammen ein Lernobjekt. Abbildung 2.7 illustriert dieses Prinzip.

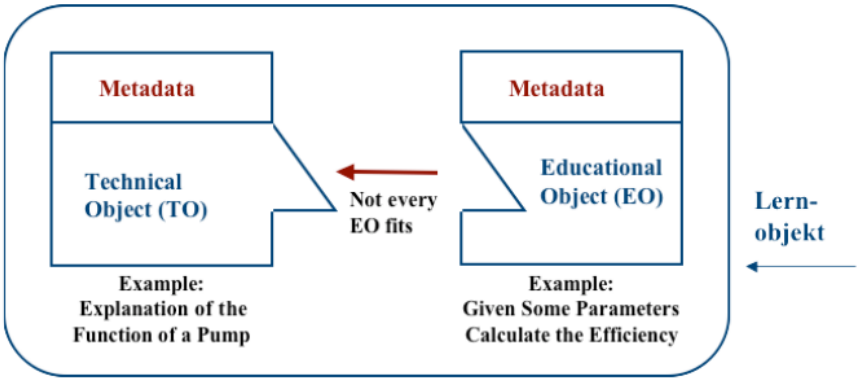

Abbildung 2.7: Darstellung eines Lernobjekts, welches sich aus einem didaktischen und einem technischen Teilobjekt zusammensetzt (Baumgartner und Kalz, 2005)

Die Grafik ist jedoch irreführend in dem Aspekt, dass die inhaltlichen und didaktischen Aspekte nicht hundertprozentig getrennt werden können. Sowohl das fachliche, als auch das didaktische Objekt enthalten jeweils beide Aspekte, nur in unterschiedlichem quantitativen Verhältnis. Daher ist dieses Prinzip in der Praxis schwer realisierbar.

Natürlich besteht ein Lernobjekt aus sehr viel mehr Metadaten als nur technischen und didaktischen Angaben. Die Learning Object Metadata working group (2002) haben in ihrem Entwurf 86 Metadatenfelder defniert, um alle Aspekte eines Lernobjekts beinhalten zu können. LOM ist heutzutage das am meisten verbreitetste Modell, um Lernobjekte innerhalb des Systems zu repräsentieren. Diese Felder teilen sich in insgesamt 9 Kategorien auf. Diese sind: (Learning Object Metadata working group, 2002)

- **General** – Allgemeine Informationen über das Lernobjekt (z.B. Identifier, Titel)

- **Lifecycle** – Werdegang und zeitliche Daten des Lernobjekts (z.B. Erstellungsdatum, Versionsnummer)
- **Meta-Metadaten** – Informationen über die Metadaten (z.B. Bearbeiter, Sprache der Metadaten)
- **Technical** – Technische Daten (z.B. Format, Größe)
- **Educational** – Didaktische Angaben (z.B. Interaktivität der Ressource, semantische Dichte, Kontext)
- **Rights** – Rechteangaben (z.B. Kosten, Copyright)
- **Relation** – Beziehungsangaben zu anderen Objekten (siehe Sektion 4.3.2)
- **Annotation** – Weitere Notizen
- **Classification** – Klassifizierungsangaben (z.B. Keywords, taxonomischer Knoten)

Diese Metadaten helfen einerseits bei der Suche nach speziellen Inhalten und Dokumentenformaten und andererseits bei der Beurteilung, ob bestimmte Lernprogramme für die eigenen Zwecke geeignet sind.
Bei der Menge an Metadaten kommt natürlich die Frage auf, wie nützlich die Metadaten für ein Lernobjekt sind und ob wirklich alle davon benötigt werden. Ochoa u. a. (2011) haben eine Untersuchung durchgeführt, welche Datenelemente aus dem LOM-Modell wie häufig benutzt werden. Dazu haben sie aus einer der größten Ansammlung von Lernobjektmetadaten, die den LOM Standard benutzt, mehr als eine halbe Million Metadatensätze gesammelt und analysiert. Von den 86 Feldern werden 43 aufgelistet. Gründe dafür sind, dass viele untergegliederte Felder weggelassen wurden und das GLOBE nicht alle Felder durchgehend implementiert hat. In der Abbildung 2.8 ist grafisch dargestellt, wie oft ein LOM-Datenelement ausgefüllt wurde.

Wichtige Erkenntnisse, die aus der Analyse festgehalten werden können, sind, dass nur etwa 16 der 43 Elemente öfter als 60% der Zeit ausgefüllt werden. Darüber hinaus werden 15 Elemente weniger als 30% der Zeit genutzt.
Daraus kann der Schluss gezogen werden, dass entweder die Benutzer mehr dazu motiviert werden müssen, diese Elemente auszufüllen, um die Objekte schlussendlich besser benutzen zu können oder es einen Weg geben muss, ein bessere Unterscheidung und einen spezifischeren Einsatz der Objekte zu erreichen mit weniger zur Verfügung stehenden Daten.

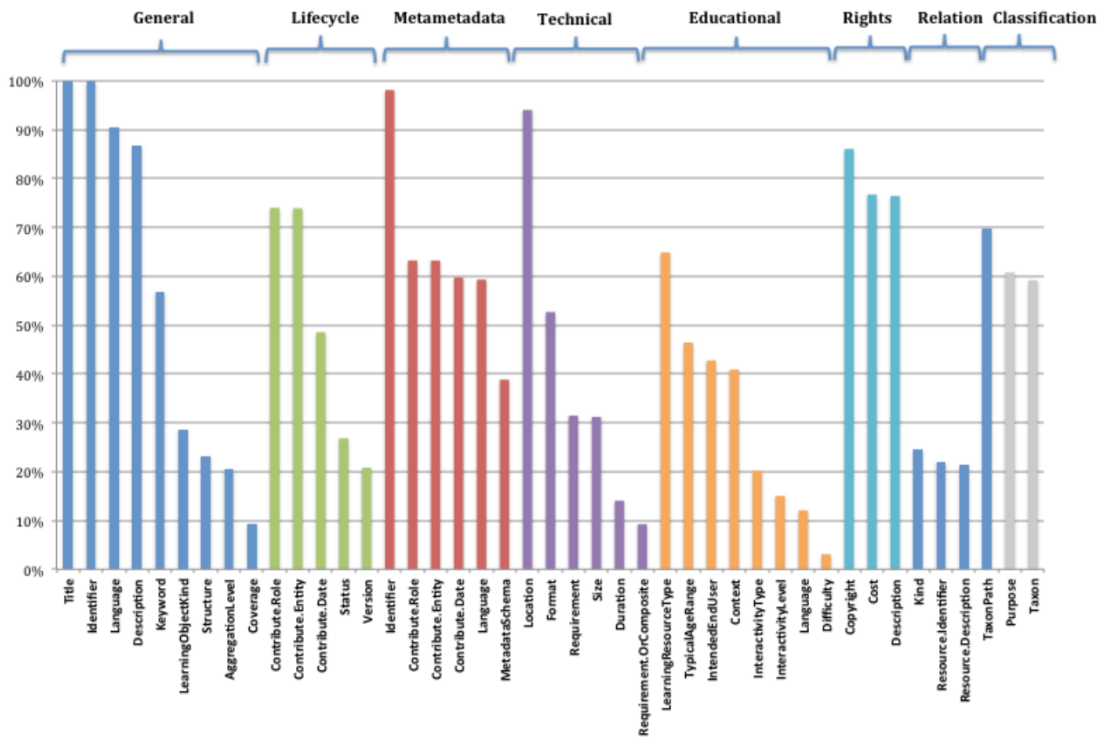

Abbildung 2.8: Prozentuale Angabe, wie oft ein Datenfeld ausgefüllt wurde von allen Datensätzen, die aus dem Repository GLOBE (Global Learning Objects Brokered Exchange) gesammelt wurden (Ochoa u. a., 2011)

2.3.3 E-Learning im (Social) Semantic Web

Seit dem Aufkommen der Idee vom semantischen Web wird versucht, E-Learning auf Basis des semantischen Webs zu realisieren. Schon 2002 haben Nilsson u. a. (2002) die Überlegung angestellt, E-Learning-Anwendungen mit den Technologien des Semantic Webs zu gestalten. In ihrem Bericht wurde darauf eingegangen, dass die Benutzung von Metadaten zur Beschreibung eines Lernkontextes essentiell für das Verständnis der Lerninhalte sind. Denn nur ein Lernobjekt in dem richtigen Kontext ist für den Lernenden verständlich und sinnvoll.

Auch die Ideen und die Technologien des *Social Semantic Webs* sind schnell im Bereich E-Learning verwendet worden. So geht Gruber (2007) auf die Vorzüge von Social Semantic Web Techniken im Bereich des E-Learnings ein. Seiner Ansicht bringt es uns näher an das Konzept von **collective intelligence** oder wie er es auch nennt: *„wisdom of crowds“*. Wie schon in Kapitel 2.1.1 beschrieben, ermöglicht das *Social Semantic Web* eine große

Synergie zwischen Menschen und Maschinen als **producer/customer** und **enablers**. Solche Systeme, welche die Eigenschaften von Semantic und Social Web verbinden, werden von Gruber **kollektive Wissenssysteme** genannt.
Als Eigenschaften von kollektiven Wissenssystemen nennt Gruber vier Eigenschaften: (Gruber, 2007)

- *User-generated content.* Der Großteil der Informationen wird von Menschen geliefert, die an einem sozialen Prozess teilnehmen. Im Gegensatz dazu, kriegt eine herkömmliche Datenbank oder ein Expertensystem den Großteil seiner Information durch systematische Datensammlung oder Wissensmodellierung.

- *Human–machine synergy.* Die Kombination von Mensch und Maschine liefert nützliche Informationen, die anderenfalls nicht zu erhalten wären. Diese Systeme decken einen größeren Bereich ab, erlauben vielseitigere Perspektiven und geben eine ungleich größere Menge an Informationen ab, als es durch traditionelle Literaturrecherche oder Expertengespräche möglich wäre.

- *Increasing returns with scale.* Umso mehr Leute mitwirken, umso nützlicher wird das System. Das Belohnungssystem, mit dem Mitwirkenden und deren Beiträgen Beachtung geschenkt wird, bleibt stabil, wenn die Menge an Informationen größer wird. Im Gegensatz dazu wird ein Textkörper und eine einfache Stichwortsuche nicht nützlicher, wenn die Content-Menge so groß wird, dass die Aussagekraft der Stichwörter nicht mehr reicht, um zwischen einzelnen Dokumenten zu unterscheiden. Gleichermaßen wird auch das System weniger nützlich umso mehr es wächst, wenn das Belohnungssystem Fälschungen belohnt oder es nicht schafft den hochqualitativsten Content nach oben zu bringen.

- *Emergent knowledge.* Das System ermöglicht Berechnungen und Schlussfolgerungen aus den gesammelten Daten, die zu Antworten, Entdeckungen oder anderen Resultaten führen, die nicht in den menschlichen Beiträgen zu finden sind.

Daneben haben Torniai u. a. (2008) darauf hingewiesen, dass das Semantic Web trotz der aussichtsreichen Aspekte immer noch nicht weitreichend benutzt wird. Dies kann unter anderem auf die Schwierigkeit zurückgeführt werden, eine gute Ontologie zu erstellen und zu pflegen und dem Prozess, Daten mit semantischen Anmerkungen zu bestücken. Dabei betonen sie, dass das Social Semantic Web eine kritische Rolle im Bereich des E-Learning spielen kann. Ihre Vision ist die eines *„Education Social*

Semantic Web“, in dem pädagogisch fokussierte Lernmaterialien und Aktivitäten für Schüler und Lehrer einfach zu erstellen, zu teilen und zu benutzen sind, ohne tiefgehende Computerkenntnisse zu haben oder Verständnis von den fortschrittlichen Technologien. Diese Vision haben sie später in dem Personal Learning Environemt DEPTHS aufgeführt, welches in Kapitel 2.3.4 erläutert ist.

2.3.4 Beispiele verschiedener E-Learning-Systeme

Es folgen drei konkrete Beispiele, die verschiedene Aspekte von E-Learning-Systemen beleuchten. Es gibt viele unterschiedliche Möglichkeiten ein E-Learning-System zu erstellen, aufbauend auf den verschiedenen Formen, die in vorigen Kapiteln erwähnt wurden. Jede der vorgestellten Systeme bietet verschiedene Features an und hat einen interessanten Ansatz im Aufbau eines E-Learning-Systems, der die Entwicklung des in dieser Arbeit beschriebenen Systems beeinflusst hat.
Darunter ist hylOs, ein **LCMS** mit integriertem Autorensystem und einer ontologischen Auswertungsschicht, DEPTHS, ein **Personal Learning Environment** und RealTravel, ein theoretisches System basierend auf den Forschungen von Gruber im Bereich der **collective knowledge systems**.

hylOs

Die Abkürzung hylOs steht für **hy**permedia **l**earning **O**bject **s**ystem. Bei hylOs handelt es sich um ein Learning Content Management System, in dem Lernobjekte erstellt und verwaltet werden und das dem Benutzer verknüpfte, personalisierte und wiederverwendbare Lerninhalte anbietet.

Die Grundlagen von hylOs wurden mit den Forschungen am Media Information Repository (MIR) gelegt. Es wurde als ein Datenbanksystem für Medienobjekte konzipiert, welches neben der Speicherung der Daten auch für die Strukturierung der gespeicherten Informationen verantwortlich ist und auf dem eine Vielzahl von Lernsystemen aufgebaut werden kann. (Feustel u. a., 2001)
Im MIR werden alle Daten in XML-Format gespeichert und ausgelesen, um einfache, individuelle Präsentation der Daten für den User über XSL anzubieten. Damit wird versucht, eine durchgehende Trennung von Content, Struktur, Logik und Design zu erreichen. (Engelhardt u. a., 2006b)
Das MIR bietet auch noch ein Linking System an, welches MIR Adaptive Context Linking

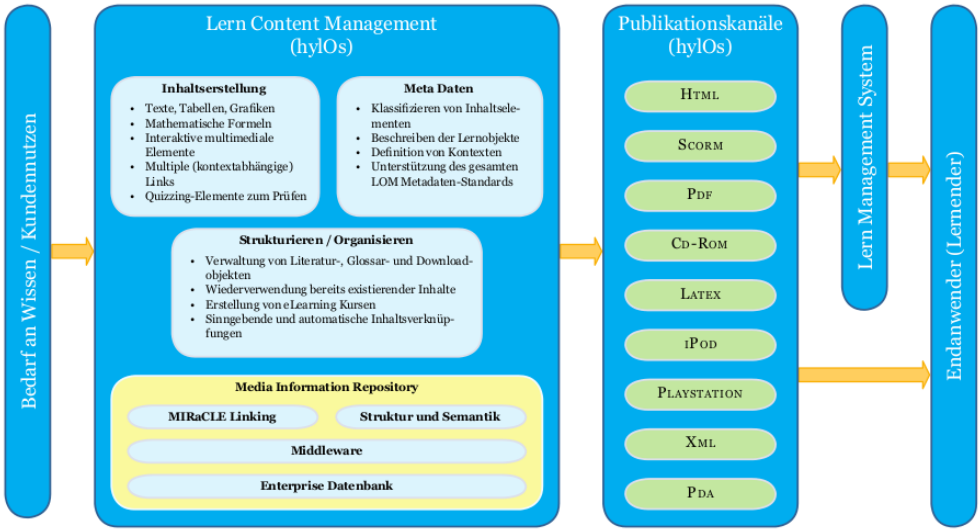

Abbildung 2.9: hylOs Architektur (Schmidt u. a., 2007)

Environment (MIRaCLE) genannt wird. Das Linking System besteht aus vier Schichten. In der Datenebene werden die rohen Daten festgehalten. Die Ankerebene adressiert Teile der Daten in der Datenebene. Die Linkebene verweist auf einen oder mehrere Anker. Die letzte Schicht ist die Linkkontextebene, welche aufgrund von semantischen Regeln die Hyperlinks dynamisch, an den jeweiligen Kontext angepasst, erzeugt (Engelhardt und Schmidt, 2003). Der Content kann mit verschiedenen Linkschemata ausgestattet werden, so dass eine dynamische Linksauswahl angeboten wird, die die Bedürfnisse des Benutzers decken (Schmidt u. a., 2009). Um dies zu ermöglichen, werden die XML-Technologien XLink, XPath und XPointer angewendet (deRose u. a., 2010; Berglund u. a., 2011; Grosso u. a., 2003).

Aus diesen gelegten Grundlagen hat sich das Learning Management System hylOs heraus gebildet. Statt in Medienobjekten, wird der Lerninhalt in E-Learning-Objekten gehalten, die nach dem LOM-Standard der Learning Object Metadata Working Group modelliert sind. Für die Erfassung der Metadaten bietet hylOs eine integrierte Autorenumgebung, in dem der Benutzer selber die Lerninhalte erstellen und Metadaten hinzufügen kann. Dabei bietet das Autorensystem einen WYSIWYG XML-Editor an, der wenig bis keine XML-Kenntnisse voraussetzt. So können auch Benutzer, die keine Kenntnisse in diesem Gebiet haben, selber Content bereit stellen. (Engelhardt u. a., 2004)

Für die Strukturierung und Verarbeitung der Lerninhalte wurde die ***Ontological Evaluation Layer*** (OEL) entwickelt, welches ein Framework zur semantischen Prozessierung

der Lernobjekte bereitstellt. Die OEL ist dafür zuständig die Lernobjekte automatisch in Beziehung zueinander zu setzen, ausgehend von ihren Metadaten und ihren schon bestehenden Beziehungen (Schmidt u. a., 2009).
Der Aufbau der Lernobjekte orientiert sich weiterhin an dem LOM-Standard und den Design-Prinzipien von Wiley (2000b). Der LOM-Standard gibt unter anderem eine ganze Reihe von verschiedenen technischen, didaktischen und anderen Elementen vor und ermöglicht es, dass Lernobjekte geschachtelt, klassifiziert und in Beziehung zueinander gesetzt werden können. Diese Beziehungen sind eine Erweiterung der Relationen des Dublin Core Metadatenstandards[8] und erlauben es die Lernobjekte semantisch zu verknüpfen. Daraus entsteht ein engmaschiges Wissensnetz, das es zum Beispiel erlaubt, automatische Content-Vorschläge zu machen, basierend auf den semantischen Beziehungen und Beschriftungen der Objekte. Die benutzten Relationen sind: (Engelhardt u. a., 2006b)

- hasPart/isPartOf
- hasVersion/isVersionOf
- isFormatOf
- references/isReferencedBy
- isBasedOn/isBasisFor
- requires/isRequiredBy
- isNarrowerThan/isBroaderThan
- isAlternativeTo
- illustrates/isIllustratedBy
- isLessSpecificThan/isMoreSpecificThan

Diese Relationen werden mittels der Ontologie-Beschreibungssprache OWL modelliert und können mittels definierten Regeln von einer Schlussfolgerungsmaschine prozessiert werden. Der Reasoner wurde als Grundlage für den Schlussfolgerungsprozess, des in dieser Arbeit beschriebenden semantischen Content-Netzes verwendet. Die genaue Implementation und Funktionsweise des Reasoners wird daher in Kapitel 4 und 5 noch genauer erläutert.

RealTravel

Ein Beispiel eines **„collective knowledege systems“** ist RealTravel. Kollektive Wissenssysteme fallen nicht unter den E-Learning-Bereich, haben jedoch überschneidende Konzepte, wie die Speicherung von Wissen und Informationen mit Metadaten und die Wissensstrukturierung über Relationen. Gruber (2007) beschreibt eine Anwendung, in

[8] http://dublincore.org/documents/usageguide/elements.shtml#relation

der die Benutzer über ihre Reiseerfahrungen berichten können. Diese Daten werden von der Maschine semantisch verknüpft und den anderen Benutzern zur Verfügung gestellt. Diese können nun anhand von Erfahrungen anderer Benutzer ihre Reise besser planen. Die Abbildung 2.10 zeigt eine Skizze des Systems.

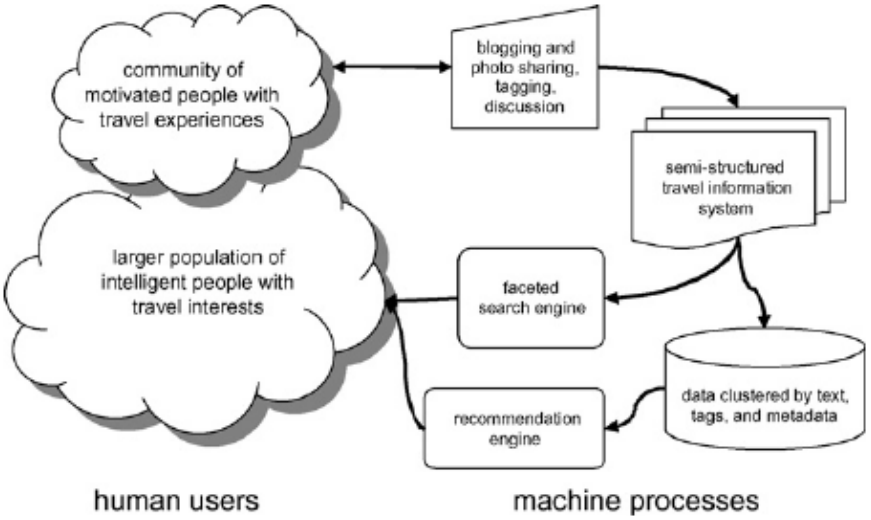

Abbildung 2.10: Abbildung des collective knowledge systems RealTravel (Gruber, 2007)

Eine „Travel-Community“ benutzt das System, um von ihren Urlaubserfahrungen zu berichten in Form eines Blogs. Diese Informationen werden vom System strukturiert und in einem Repository gespeichert. Reiseunerfahrene Benutzer können automatisch Daten aus den Informationen der erfahreneren Benutzer bekommen, die passend zu ihren Suchanfragen zugeschnitten sind. Das wird erreicht durch eine umfangreiche *Search Engine* und eine *Recommendation Engine*, die dem Anwender einerseits eine computerunterstützte Suchanfragen ermöglicht und ihm andererseits persönlich zugeschnittene, gefilterte Ergebnisse liefert. Wobei natürlich die Filterung von Informationen wiederum eine Zensur darstellt. Daher muss sichergestellt werden, dass nicht Daten aufgrund von schlechten Bewertungen oder ähnlichem zurückgehalten werden, sondern dass vermeintlich passendere Daten mit höherer Priorität angezeigt werden.
Um die Funktionen bereitstellen zu können, listet Gruber ein paar Techniken aus dem Social und Semantic Web auf, die seiner Ansicht nach den Zugriff auf strukturierte oder semi-strukturierte Daten erleichert und die Ergebnisse für den Benutzer optimiert.

- **Snap-to-grid** - Basierend auf den Eingaben des Users werden von dem Computer Vorschläge gemacht, die der Eingabe ähneln. Mittels Techniken wie Auto-Complete

können die Eingaben existierenden Objekten angepasst werden, so dass jeder Eintrag präzise zugeordnet wird.

- **Tagging** - Objekte werden getaggt, damit man verschiedene Wörter auf die gleiche interne Entity abbilden kann
- **Rating** - Content wird von Nutzern bewertet
- **Personalized Search** - aufgrund von Vorlieben des Benutzers können individuell angepasste Suchvorschläge angeboten werden

Die „collective knowledge systems“ benutzen viele Techniken des Social Semantic Webs und zeigen ein System, welches Wissen gewinnt durch die Synergie zwischen der Community und dem maschinellen Prozess. Daher sind viele der Techniken, die Gruber beschreibt, wichtig für den Bau eines „eLearning enabled Online Social Network“, welches die Grundlage des in dieser Arbeit behandelten Systems darstellt, und werden auch in den folgenden Kapiteln wieder aufgegriffen werden.

DEPTHS

Ein Beispiel für ein *Personal Learning Environment* (PLE) ist DEPTHS, welches ein Akronym ist für **DE**sign **P**atterns **T**eaching **H**elp **S**ystem. Eine ausführliche Beschreibung dazu findet sich in Jeremić u. a. (2011). Die Idee hinter dem System ist es, eine Lernplattform speziell für den Bereich Software Design Patterns für Softwareentwickler zu schaffen. Dabei verfolgt es einen projektbasierten Lernansatz, in dem es eine gemeinschaftliche Lernunterstützung gibt. Ein mögliches Szenario ist, dass ein Lehrer ein bestimmtes Software-Design Problem vorgibt, welches anschließend in einer Art Workshop von den Studenten, in Form eines Softwareprojekts, gelöst wird. Dazu können sie sich über das System mit anderen Studenten, Lehrern oder Experten austauschen und ihre Ergebnisse sammeln und veröffentlichen.
Um diese Anforderungen zu erfüllen, wurde das System auf den Prinzipien von Personal Learning Environments aufgebaut. PLEs zeichnen sich dadurch aus, dass sie Benutzern erlauben, ihre Lernumgebung individuell anzupassen. Es wird den Lernenden die Möglichkeit gegeben einen eigenen Blog zu erstellen und ihre Lösungen und Fortschritte anderen zu präsentieren. Wichtige Funktionen, die den Lernenden von PLEs bereitgestellt werden, sind: (Jeremić u. a., 2011; Attwell u. a., 2008)

- Strukturierung: Der Lerner kann die gefundenen Informationen sammeln und für einer ihm sinnvollen Weise strukturieren, um einen besseren Überblick zu bekommen und die Zusammenhänge besser zu erfassen.
- Bearbeitung: Der Lerner kann für sich Wissensbausteine bearbeiten
- Präsentation: Der Lerner kann einerseits über eigens verfasste Blogs oder Artikel das Gelernte nochmal reflexiv betrachten und andererseits den Mitlernenden die eigenen Fortschritte und Lösungen präsentieren
- Teilen: Der Lerner kann gewonnenes Wissen teilen und vom Wissen anderer profitieren über Kontakt mit anderen Lernenden oder Experten

Im Gegensatz zum LMS ist der Lernende in einem PLE mehr in einer aktiven Rolle und kann seine Umgebung selbst gestalten und ist für den Lerninhalt und seinen Lernprozess selber verantwortlich. Dadurch, dass er den Lerninhalt nicht nur konsumiert, sondern auch zum Teil produziert, wird von einer symmetrischen Beziehung zwischen dem Lernenden und dem PLE gesprochen. (Martindale und Dowdy, 2010) Im Kontrast dazu nimmt der Lernende in einem LMS eher eine passive Rolle ein und der Lehrer ist in Kontrolle des Lerninhalts und dessen Präsentation.
Dabei gilt es zu erwähnen, dass das Konzept eines PLE kein klar definierter Rahmen ist, sondern dass die Grenzen zwischen einem LMS und einem PLE eher verschwimmend sind. Eine persönlich gestaltbare Umgebung und die Kontrolle über den eigenen Lernprozess sind jedoch konkrete Merkmale eines PLEs.
Diese angesprochenen Merkmale von PLEs bieten allerdings auch Schwierigkeiten in der Implementation solcher Systeme. So wird von den Benutzern mehr Eigenverantwortung und eine größere Medienkompetenz abverlangt (Schaffert und Hilzensauer, 2008). Gleichzeitig wirft es die Frage auf, wie LMS und PLE miteinander oder nebeneinander existieren können. LMS sind ein wesentlich weiter verbreitertes und länger etabliertes Lernsystem. Wilson u. a. (2007) listet dazu drei Szenarien auf, wie PLEs und LMS koexistieren könnten. Das Erste ist, dass beide Systeme nebeneinander existieren und verschiedene Bereiche abdecken. PLE den *informal learning space* und LMS die *formal education*. Die zweite Möglichkeit ist, dass LMS seine Services anbietet, die von PLEs genutzt werden könnten. Das dritte und wahrscheinlichste Szenario ist, dass sich beide Systeme weiter annähern und die Charakteristiken von PLEs immer mehr in LMS eingebaut werden.
DEPTHS liefert nun ein Beispiel, wie die Funktionen eines PLEs implementiert werden könnten. Als Basisplattform für DEPTHS dient ein Open-Source-LMS, welches erweitert

und angepasst wird. Ergänzend dazu werden Social- und Semantic Web Komponenten verwendet, um die oben genannten Anforderungen zu erfüllen. So wurden Social-Web-Komponenten eingebaut, wie Blogs und Chats, damit die Benutzer die Möglichkeit haben miteinander zu kommunizieren und ihren Lernprozess zu teilen und zu präsentieren. Für die Bearbeitung und Strukturierung der Daten wurden Semantic-Web-Komponenten verwendet, so zum Beispiel Werkzeuge, um semantische Anmerkungen an Content anzufügen. (Jeremić u. a., 2011)
Das Problem der erforderlichen größeren Medienkompetenz der Benutzer wird umgangen, indem es sich um ein System für Softwareentwickler handelt, die eine gewisse Medienkompetenz mitbringen.

Alle drei Systeme bieten einen eigenen Ansatz von Lernplattformen, die für die Konzeption des Diaspora-Systems, welches im nachfolgenden Kapitel erläutert wird, von Bedeutung sind.
Der Aufbau, die Struktrurierung und die Vernetzung der Lernobjekte im semantischen Content-Netz orientiert sich an dem Ontological Evaluation Layer aus hylOs. Die im Rahmen der Arbeit erstellten Relationen, Schlussfolgerungsregeln und die Ontologie werden als Grundlage für den Aufbau des semantischen Content-Netzes genommen. Die Abgrenzungen in der Implementierung werden in Kapitel 5.2 beschrieben. Die Techniken, die Gruber (2007) im Rahmen von RealTravel aufgeführt hat, können in dem Diaspora-System verwendet werden, um das Wissen und die Informationen aus dem semantischen Content-Netz einfach zugänglich für die Anwender zu machen. Diese Techniken bieten eine Möglichkeit, die sozialen mit den semantischen Komponenten zu verbinden.
Für die Konzeption der „*Social Web*-Komponente“ des Diaspora-Systems können die Funktionen, welche in Personal Learning Environments angeboten werden, Verwendung finden. Im sozialen Web geht es nicht nur um die Gemeinschaft, sondern das jeder die Möglichkeit hat gehört und gesehen zu werden. Individuelle Expression in ein Bestandteil des sozialen Webs. Die individuellen Gestaltungsmöglichkeiten von PLEs können diese Ideen in das Diaspora-System übertragen.

3 Verwendete Technologien

3.1 Sprachen und Werkzeuge im semantischen Web

3.1.1 RDF

Mit dem **R**esource **D**escription **F**ramework lassen sich Ressourcen im Internet formal beschreiben und logische Aussagen über Ressourcen formulieren (Schreiber und Raimond, 2014). Die erste stabile Version der RDF-Spezifikation, die den Aufbau von RDF-Modellen und eine Serialisierung mit XML beschrieb, wurde 1999 veröffentlicht und als W3C-Recommendation zugelassen (Lassila und Swick, 1999). Im W3C ist eine *Recommendation* definiert als eine *„specification or set of guidelines or requirements that, after extensive consensus-building, has received the endorsement of W3C Members and the Director“* und das W3C empfiehlt diese Spezifikationen als Standards zu benutzen (McCathie Nevile, 2014).

Über die Zeit sind zu der XML-Syntax noch weitere Syntaxspezifikationen hinzugekommen, um gewisse Mängel der XML-Notation auszugleichen. So zum Beispiel **Turtle**, mit denen RDF-Modelle wesentlich menschenlesbarer notiert werden können oder **JSON-LD**, um RDF-Modelle mit dem weit verbreiteten JSON-Format zu serialisieren und zwischen Servern und Webanwendungen übermitteln zu können.

RDF-Modelle beschreiben Ressourcen mit Statements. Ein Statement ist ein Tripel der Form Subjekt-Prädikat-Objekt, wobei alle drei Teile mit einer IRI (International Resource Identifier) oder mit einer Zeichenkette identifiziert werden. Das **Subjekt** ist die Ressource die beschrieben werden soll und das **Objekt** steht mit dem Subjekt in Beziehung über das **Prädikat**, auch genannt *Property*.

Als Beispiel sollen die Eigenschaften eines Dokuments beschrieben werden. Dabei wird die Turtle-Notation verwendet.

```
1 BASE <http://www.example.org/data/>
2 PREFIX file: <http://www.example.org/>
3 PREFIX type: <http://www.w3.org/example/>
4
5 <datei.doc>
```

```
6   a type:File;
7   file:size "100Kb";
8   file:created "10.02.2014";
9   file:mime type:application/msword;
```

Listing 3.1: Ein Beispiel für einen in Turtle notierten RDF-Graphen

Die ersten drei Zeilen dienen als Deklaration der Präfixe und können als Kurzschreibweise benutzt werden. Die *BASE*-IRI wird für die relative Angabe von IRIs genutzt, wie *<datei.doc>* in Zeile 5, die nach *http://www.example.org/data/datei.doc* aufgelöst wird. In den darauf folgenden Zeilen werden die Eigenschaften des Subjekts definiert, darunter die Größe, das MIME-Format und das Erstellungsdatum. Daneben wird noch der Typ des Subjekts definiert über die Property *„a“*, eine Kurzschreibweise für *<rdf:type>*. Die logische Aussage des RDF-Beispiels ist, dass die Ressource mit dem Identifier *www.exampe.org/data/datei.doc*, 100kB groß ist, den MIME-Type *application/msword* hat und am 10.02.2014 erstellt wurde. Die Abbildung 3.1 zeigt das Beispiel nochmal als Graph.

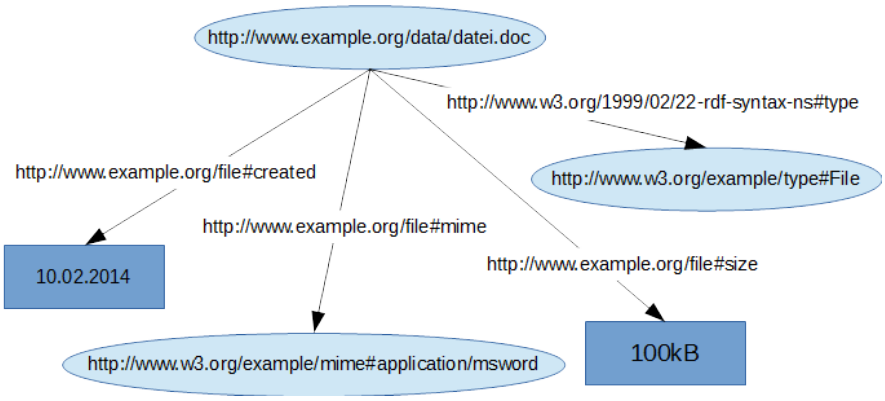

Abbildung 3.1: Darstellung des RDF-Beispiels in Listing 3.1 als Graph (Quelle: Eigene Darstellung in Anlehnung an Gandon und Schreiber (2014))

IRIs müssen nicht zwingend im Netzwerk zu erreichen sein. Es wird jedoch empfohlen Identifier als Webadresse in Form einer URL (Uniform Resource Locator) zu benutzen. Im Falle des Subjekts kann so die Ressource direkt verlinkt werden. Im Falle der Property kann die Bedeutung der Relation hinterlegt werden und so dieselbe Property in verschiedenen Szenarien mit der gleichen Bedeutung benutzt werden.
RDF ist dazu vorgesehen als Basisnotation für eine Vielzahl von erweiterten Notation verwendet zu werden, wie RDF Schema oder OWL, welche in den nachfolgenden Kapiteln

erläutert werden. Deren Ausdrücke können somit auch als RDF-Graphen codiert werden. (Hayes und Patel-Schneider, 2014)

3.1.2 RDF Schema

RDF Schema bietet ein Vokabular für die Datenmodellierung und wird zusammen mit dem RDF-Vokabular verwendet. RDF Schema ist eine semantische Erweiterung von RDF (Brickley und Guha, 2014). Dies bedeutet, dass den RDF-Graphen bestimmte Restriktionen oder syntaktische Bedingungen auferlegt werden können, um von der semantischen Erweiterung interpretiert werden zu können und keine Fehler zu verursachen. Es können zum Beispiel bestimmte Tripel benötigt werden oder bestimmte Kombinationen von Tripeln nicht zugelassen sein. (Hayes und Patel-Schneider, 2014)
In RDF Schema können **Klassen** und **Eigenschaften** definiert werden. Ressourcen können in Klassen eingeteilt werden. Wenn eine Ressource Mitglied einer Klasse ist, wird von einer *Instanz* dieser Klasse gesprochen. Ressource ist die Oberklasse. Somit ist alles, was in RDF beschrieben wird, eine Ressource, auch Klassen. In RDF Schema können Vererbungshierarchien aufgebaut werden und Klassen können Datentypen[1] zugeordnet werden, wie *String*, *Boolean*, *Integer* oder *Date*. (Brickley und Guha, 2014)
Eigenschaften gehören der Klasse *rdf:Property* an. Sie beschreiben die Beziehungen zwischen einer Subjektressource und einer Objektressource. Mit *rdfs:domain* kann die Klasse oder der Typ vom Subjekt des Prädikats beschrieben werden, mit *rdfs:domain* die Klasse oder der Typ vom Objekt des Prädikats. Weiter können über *rdfs:subClassOf* Unterklassen definiert werden.
Um das Beispiel in Kapitel 3.1.1 weiterzuführen könnten die Klassen *Program* und *File* eingeführt werden. Weiter wird deklariert, dass Ressourcen vom Typ *File* von *Programs* geöffnet werden können. Dies wird angezeigt durch die Eigenschaft *file:opensWith*. Als Dokument in Turtle-Notation könnte diese Deklaration so aussehen:

```
1 @base <http://www.example.org/data/> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4 @prefix file: <http://www.example.org/> .
5 @prefix type: <http://www.w3.org/example/> .
6
7 type:File rdf:Type rdfs:Class .
8 type:Program rdf:Type rdfs:Class .
9 file:opensWith rdf:type rdf:Property .
```

[1] Liste aller möglichen Datentypen: http://www.w3.org/TR/rdf11-concepts/#section-Datatypes

```
10
11 <datei.doc>
12   a type:File .
13
14 <http://www.programs.de/Microsoft_Word>
15   a type:Program .
16
17 file:opensWith rdfs:domain ex:File .
18 file:opensWith rdfs:range ex:Program .
```

Listing 3.2: Weiterführung des Beispiels in Listing 3.1 mit mit einem durch RDF Schema erweiterten Vokabular

Mit der Deklaration, dass *http://www.example.org/data/datei.doc* vom Typ *File* ist und *http://www.programs.de/Microsoft_ Word* vom Typ *Program* kann das Statement aufgebaut werden:

```
ex:datei.doc file:opensWith <http://www.programs.de/Microsoft_Word>.
```

3.1.3 OWL

Die Web Ontology Language ist eine Spezifikation, um Ontologien formal zu beschreiben. OWL ist wie RDF Schema eine semantische Erweiterung von RDF und benutzt RDF und RDF Schema als Basisnotation. OWL entwickelte sich aus der Sprache DAML+OIL heraus. OWL wurde 2004 eine W3C-Recommendation (Bechhofer u. a., 2004). Kurz darauf wurde die OWL Working Group gegründet, die OWL weiter verbesserten und mit OWL2 eine überarbeitete Version anboten, die OWL1 um weitere Funktionalitäten und Sprachkonstrukte erweiterte. OWL2 wurde 2012 eine W3C-Recommendation (W3C OWL Working Group, 2012).

Abbildung 3.2 zeigt den Aufbau von OWL2. Ontologien können mit verschiedenen Syntaxspezifikationen serialisiert werden. Dabei ist die RDF/XML-Syntax der Standard zum Austausch von OWL2-Ontologien und muss von jedem OWL-Tool implementiert werden. Die anderen Spezifikationen sind optional und dienen entweder der einfachen Prozessierung durch XML-Tools oder der einfacheren Lesbarkeit. (W3C OWL Working Group, 2012) Eine geparste Ontologie kann als RDF-Graph abgebildet werden. Durch Anwendung bestimmter Abbildungsregeln kann jede Ontologie in ein RDF-Graph transformiert werden. Andersherum können RDF-Graphen unter Einhaltung gewisser Restriktionen in eine OWL2 DL Ontologie überführt werden. Die Transformationen haben dabei keine veränderliche Auswirkung auf die Bedeutung der Ontologie. (Patel-Schneider

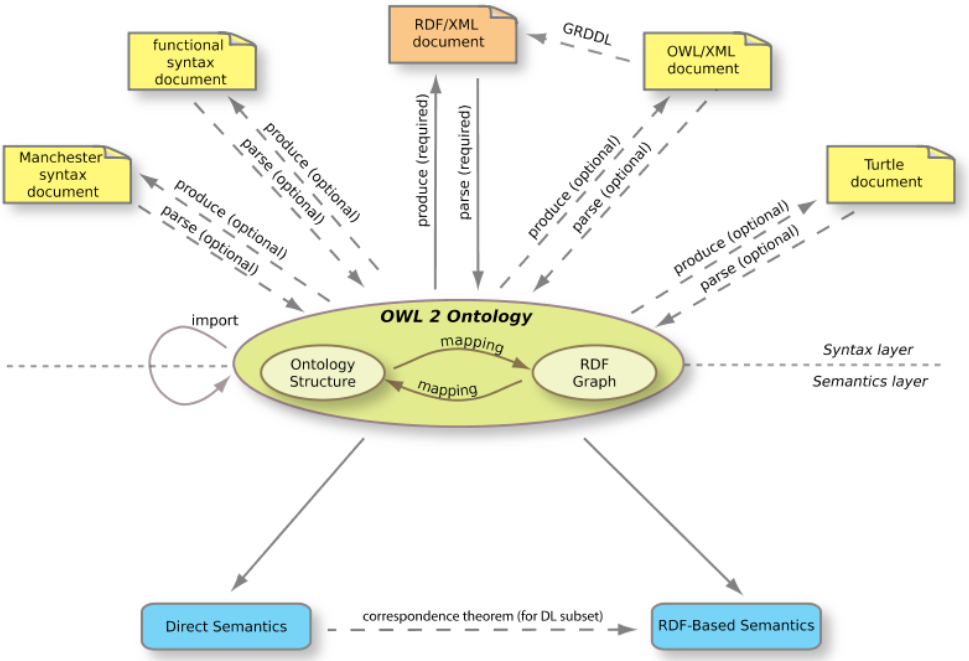

Abbildung 3.2: Eine Übersicht über die Struktur von OWL2. Die Ellipse im Zentrum repräsentiert die Ontologie. In der Syntaxebene gibt es zahlreiche konkrete Syntaxspezifikationen, mit der die Ontologie serialisiert und ausgetauscht werden kann. In der Semantikebene wird der Ontologie eine Bedeutung zugewiesen, entweder über *OWL2 Direct Semantics* oder über *RDF-Based Semantics* (W3C OWL Working Group, 2012)

und Motik, 2012) Um die Bedeutung der Ontologie zu definieren, bedarf es semantischer Spezifikationen. Dabei gibt es einerseits die *OWL2 Direct Semantics*, die sich stark an der Semantik der Beschreibungslogik orientiert (Motik u. a., 2012b) und andererseits die *RDF-Based Semantics*, die direkt den transformierten RDF-Graphen eine Bedeutung zuweist (Schneider, 2012). Diese Semantiken werden von logischen Schlussfolgerern und anderen Werkzeugen eingesetzt, um Klassenkonsistenz, Unterordnung von Begriffen und Instanzabfragen zu überprüfen und zu entscheiden (W3C OWL Working Group, 2012). OWL umfasst mehrere Profile, die einerseits für verschiedene Anwendungsbereiche konzipiert wurden und andererseits eine unterschiedliche Ausdruckskraft besitzen. Dabei wurden in der ursprünglichen OWL1-Referenz drei Profile definiert, die **OWL Lite**, **OWL DL** und **OWL Full** genannt wurden. In OWL2 wurden weitere drei Profile definiert mit Namen **OWL 2 EL**, **OWL 2 QL** und **OWL 2 RL**. Dabei sind bei den OWL1-Profilen die weniger mächtigen Varianten eine Teilmenge der mächtigeren Varian-

ten. Jede Ontologie und Schlussfolgerung, die in OWL-Lite gültig ist, ist auch in OWL-DL gültig und genauso steht es mit OWL-DL zu OWL-Full. (McGuinness und van Harmelen, 2004) Bei den OWL2-Profilen sind die drei Varianten keine Teilmengen voneinander. Die Unterschiede in den Profilen kommen von verschiedenen syntaktischen Restriktionen und das sie für unterschiedliche Anwendungsbereiche konzipiert sind. (Motik u. a., 2012a)

OWL Lite ist eine abgespeckte Version von OWL und verzichtet auf einige Sprachkonstrukte. Einerseits um dem Benutzer eine einfach zu implementierende Sprache für simple Taxonomien und kleinere Ontologien zu geben. Andererseits um es den Entwicklern einfacher zu machen, Werkzeuge für OWL Lite zu entwickeln. In der Praxis hat sich dies jedoch nicht bewährt, da mit den zur Verfügung stehenden Sprachkonstrukten von OWL Lite es immer noch möglich ist, komplexe Sprachkonstrukte von den nicht begrenzten Versionen darzustellen. Schlussfolgerungsprobleme sind für OWL Lite auch nur geringfügig weniger komplex als bei OWL DL. Genauer benutzt OWL DL die Beschreibungslogik SHOIN, deren Entscheidbarkeitsprobleme der Komplexitätsklasse NEXPTIME angehören und OWL Lite die Beschreibungslogik SHIF, mit einer Komplexität von EXPTIME. (Grau u. a., 2008)

OWL DL wurde so entwickelt, dass dem Benutzer die größtmögliche Ausdruckskraft zur Verfügung steht. Dabei sind mit OWL2 DL verfasste Ontologien durch Schlussfolgerungswerkzeuge immer noch komplett entscheidbar. OWL DL beinhaltet alle Sprachkonstrukte von OWL Full, jedoch können diese teilweise nur mit bestimmten Restriktionen verwendet werden. Beispielsweise darf eine Klasse nicht Instanz einer anderen Klasse sein. Die Semantik von OWL DL, wie auch von OWL Lite, ist aufgebaut auf der Beschreibungslogik (engl.: *Description Logic*). Die Beschreibungslogik umfasst entscheidbare Teile der Prädikatenlogik erster Stufe. (McGuinness und van Harmelen, 2004)

OWL Full umfasst alle Sprachkonstrukte von OWL und ist gedacht für Benutzer, die maximale Ausdruckskraft benötigen, jedoch ohne Garantie, dass mit OWL Full erstellte Ontologien von Schlussfolgerungssoftware komplett verarbeitet werden können. OWL Full kann als semantische Erweiterung von RDF angesehen werden, wogegen OWL DL und OWL Lite als Erweiterung einer eingeschränkten Version von RDF angesehen werden können. (McGuinness und van Harmelen, 2004)

Die Benutzung von **OWL2 EL** ist nützlich bei Ontologien, die eine große Menge an Klassen und Eigenschaften haben und wo Ausdruckskraft wichtiger ist als Performanz. OWL EL gehört der Familie der Beschreibungslogik EL++ an und kann die grundlegenden Schlussfolgerungsprobleme in Polynomialzeit (PTIME) entscheiden. (Motik u. a., 2012a)

OWL2 QL ist für Anwendungen gedacht, in der das Antworten von Abfragen die wichtigste Schlussfolgerungsaufgabe ist und in der eine große Menge an Instanzen vorhanden sind. Das Abhandeln dieser Schlussfolgerungen kann mit logarithmischen Speicheraufwand gemessen am Input erledigt werden. (Motik u. a., 2012a)
OWL2 RL ist gezielt für Anwendungen entwickelt worden, die skalierbares Schlussfolgern brauchen. Die meisten grundlegenden Schlussfolgerungsprobleme können in Polynomialzeit gelöst werden, relativ zur Größe der Ontologie.(Motik u. a., 2012a) Es ist vor allem nützlich in Anwendungen, wo relativ leichtgewichtige Ontologien benutzt werden, um eine große Anzahl an Objekten zu organisieren und wo es nützlich ist direkt auf den RDF-Tripeln zu arbeiten. (W3C OWL Working Group, 2012) Um die Verwendung von OWL zu demonstrieren, kann das Beispiel aus den vorigen Sektionen weiter ausgeführt werden. So kann eine Klasse *TextFile* als Unterklasse zu *File* hinzugefügt werden. *TextFile* hat eine Restriktion, dass nur Files mit dem Format *text* als *TextFile* gelten.

```
@base <http://www.example.org/data/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix file: <http://www.example.org/> .
@prefix type: <http://www.w3.org/example/> .

type:Format rdf:type owl:Class .
type:File rdf:type owl:Class .
type:TextFile
  a owl:Class
  rdfs:subClassOf type:File
  rdfs:subClassOf
    [ a owl:Restriction .
      owl:Property :format .
      owl:hasValue "text"^^type:Format .
    ] .
<datei.doc>
  a type:File .
  file:size "100kB" .
  file:mime "application/msword" .
  type:format "text" .
```

Angenommen es existiert die Verbindung *#format rdfs:range #Format* und *#format rdfs:domain #File*, so kann durch eine Schlussfolgerungsengine ermittelt werden, dass die Datei *http://www.example.org/data/datei.doc* ein *TextFile* ist.

3.1.4 SPARQL

SPARQL (**S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage) ist eine weit verbreitete Abfragesprache für Daten im RDF-Format. Sie wurde 2008 als W3C Recommendation aufgenommen und existiert mittlerweile (Stand: Februar 2015) in der Version 1.1. (Harris und Seaborne, 2013)
SPARQL orientiert sich bei der Syntax an SQL und benutzt SELECT-WHERE-Abfragen, um die gewollten Daten zu bekommen. Da Daten in RDF-Format in Tripeln vorliegen, werden in SPARQL die Daten auch in Form von Tripeln abgefragt. Ähnlich wie in SQL werden die gesuchten Datenfelder mit Variablen ersetzt, die mit einem Fragezeichen beginnen.
Als Beispiel wollen wir alle *#Files* aus den vorigen Beispielen holen.

```
1 PREFIX class: <http://www.example.org/class/>
2 SELECT ?file
3 WHERE {?file a class:File}
```

Das Schlüsselwort „a“ fragt den Typen des Subjekts ab, wirkt also äquivalent zu *rdf:type*. Als Ergebnis würde die Ressource *http://www.example.org/data/datei.doc* zurück geliefert werden.

3.1.5 SKOS

SKOS (**S**imple **K**nowledge **O**rganization **S**ystem) ist ein Datenmodell für Systeme zur Wissensorganisation, wie Thesauri, Klassifikationsschemata und Taxonomien. Mit SKOS können solche Systeme als maschinenlesbare Daten formuliert werden. Das SKOS-Datenmodell ist spezifiziert als eine OWL Full Ontologie und kann mit jeder konkreten RDF-Syntax, wie RDF/XML oder Turtle, serialisiert werden. (Miles und Bechhofer, 2009)
Die Hauptklasse von SKOS ist *skos:Concept*, mit dem abstrakte oder konkrete Element bezeichnet werden können, wie Ideen, Bedeutungen, Objekte oder Events. In einer Taxonomie wird mit einem *Concept* zum Beispiel ein Knoten innerhalb der Taxonomie dargestellt. Für die Beschreibung von Taxonomien bietet SKOS weitere Konstrukte an wie *skos:narrower* und *skos:broader*, um auf unterliegende, bzw. überliegende Knoten innerhalb einer Taxonomie zu verweisen.

3.1.6 JENA

JENA[2] ist ein Framework zur Erstellung, Modifizierung und zum Schreiben von RDF-Graphen, welches ursprünglich von HP Labs entwickelt wurde, nun aber als freie Software unter der Apache-Lizenz weiterentwickelt wird. JENA wurde für die Programmiersprache Java entwickelt. Es erlaubt Graphen aus RDF- oder OWL-Dokumente zu erstellen und umgekehrt, in JENA erstellte Modelle als RDF- oder OWL-Dokumente auszugeben. JENA bietet Java-Klassen an, um die Sprachelemente eines RDF-Dokuments als Objekte darzustellen. So kann der RDF-Graph als ein objektorientiertes Modell dargestellt werden. Das Beispiel aus den vorigen Kapiteln könnte nun in JENA eingelesen werden:

```
1 OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_LITE_MEM);
2 m.read("path/to/model");
```

Dieses Modell kann nun in Java manipuliert werden und für Abfragen benutzt werden. Als Abfragesprache kann SPARQL oder die eigens von HP Labs kreierte Sprache RDQL (**R**esource **D**escription **Q**uery **L**anguage) verwendet werden. Die gleiche Abfrage, welche in Kapitel 3.1.4 benutzt wurde, würde mit JENA so aussehen:

```
1 String queryString = "PREFIX class: <http://www.example.org/class/>
2                       SELECT ?file WHERE {?file a class:File}";
3 Query query = new Query(queryString);
4 query.setSource(m);
5 QueryEngine qe = new QueryEngine(query);
6 QueryResults results = qe.exec();
```

Dies ist sehr ähnlich zu den SQL-Abfragen, welche mit dem JDBC-Adapter in Java erstellt werden.

Logisches Schlussfolgern

JENA hat verschiedene Schlussfolgerungs-Engines (engl.: *Reasoner*) integriert und kann auch mit weiteren externen *Reasonern* verwendet werden. Für die Schlussfolgerungen brauch Jena ein Ontologiemodell und eine Regeldatei. Das Ontologiemodell kann aus OWL-Dokumenten eingelesen werden. Eine Regeldatei muss mit einer Regelbeschreibungssprache erstellt werden, wie zum Beispiel RIF (**R**ule **I**nterchange **F**ormat), und kann dann von JENA eingelesen werden. JENA bringt auch eine eigene Regelsprache mit, die eine andere Syntax aufweist, als die RIF-Dialekte.
In den Regelsprachen werden die Schlussregeln im if-then-Muster deklariert. Eine Typische Schlussregel wäre der hypothetische Syllogismus: *if* $a \rightarrow b \wedge b \rightarrow c$ *then* $a \rightarrow c$.

[2] http://jena.apache.org/

Es können auch RDF-Konstrukte benutzt werden, um zum Beispiel den *Type* oder die Klasse einer Ressource abzufragen und daraus eine Schlussfolgerung zu ziehen. Ein Beispiel in der JENA-Regel-Syntax könnte so aussehen:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex: <http://www.w3.org/example/props#>.
[rule1: (?p rdf:type ex:symmetry) ->
  [rule1a: (?a ?p ?b) <- (?b ?p ?a) notEqual(?a ?b)]
]
```

In diesem Beispiel würde die Aussage aufgestellt werden, wenn das Prädikat *?p* eine symmetrische Beziehung wäre, so gilt bei dem Tripel *(?a ?p ?b)* auch *(?b ?p ?c)*, wenn *?a* und *?b* nicht gleich sind.

3.2 Diaspora

Diaspora[3] ist ein verteiltes soziales Netzwerk basierend auf einer Open-Source-Software. Der Name Diaspora betitelt dabei sowohl das soziale Netzwerk als auch die freie Software. Das Prinzip von Diaspora ist, dass jeder Nutzer die Möglichkeit hat selber einen eigenen Diaspora-Server, genannt *Pod*, aufzusetzen. Pods können untereinander kommunizieren und formen so das Diaspora-Netzwerk. Wenn ein Benutzer sich auf einem Pod anmeldet, werden seine Daten nur auf diesem Server gespeichert. Wenn der Pod ans Diaspora-Netzwerk angeschlossen wurde, ist er jedoch auch von anderen Pods aus erreichbar. Somit gibt es keinen zentralen Server, wo alle Benutzerdaten gespeichert werden, sondern viele einzelne - meist private - Server, die die Datenwahrung übernehmen. Somit hat kein Server Zugriff auf alle Daten des Diaspora-Netzwerks.
Wenn ein Benutzer dem Netzwerk beitritt, muss er sich für einen Pod entscheiden. Dabei kriegt er eine ID zugewiesen, bestehend aus seinem Benutzernamen und der Pod-Webadresse, die Pod-übergreifend zur Identifikation benutzt wird.
Diaspora wurde 2010 von vier Studenten als Kickstarter-Projekt[4] gestartet und mit den erzielten Spenden wurde die erste Alpha-Version im November gleichen Jahres veröffentlicht.
Die Diaspora-Software wurde in Ruby und mit dem Web-Framework Ruby on Rails programmiert. Weitere Sprachen und Bibliotheken, die benutzt werden, sind HAML (**H**TML **A**bstract **M**arkup **L**anguage) und SASS (**S**yntactically **A**wesome **S**tyle**s**heets),

[3] https://diasporafoundation.org/
[4] https://www.kickstarter.com/projects/mbs348/diaspora-the-personally-controlled-do-it-all-distr

zwei abstrakte Beschreibungssprachen, die in HTML-, respektive CSS-Code, interpretiert werden. Daneben werden Backbone.js und Handlebars.js benutzt. Backbone.js modelliert ein MVC-Pattern auf der Client-Seite und kommuniziert mit JSON über ein REST-Interface mit dem Server. Mit Handelbars.js können semantische Templates erstellt werden, die für die dynamische Inhaltsdarstellung verantwortlich sind. In den folgenden Unterabschnitten werden diese Sprachen ausführlicher aufgegriffen und erläutert. Abbildung 3.3 skizziert die Architektur der Diaspora-Software.

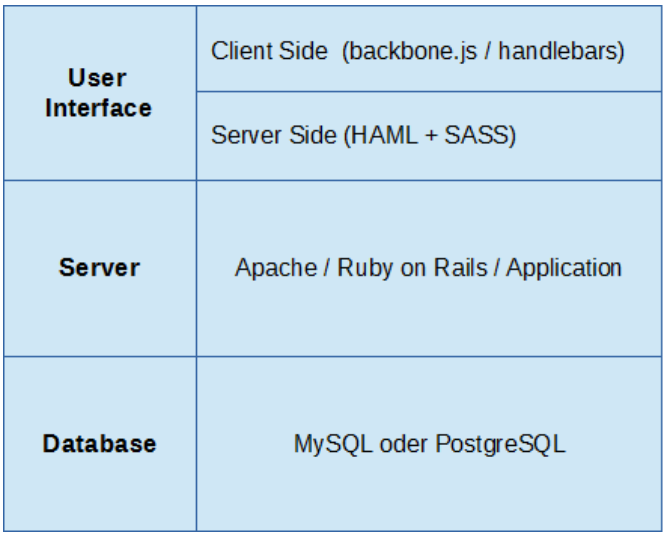

Abbildung 3.3: Überblick der Architektur der Diaspora-Software. (Quelle: eigene Darstellung in Anlehnung an (Haß, 2013))

Erwähnenswerte Features von Diaspora sind unter anderem:

- ein Aspektsystem, in dem Freunde verschiedenen Gruppen zugeordnet werden können
- ein Hashtag-, Follower- und Mention-System, ähnlich dem von Twitter
- ein Like-System und eine Profilseite, ähnlich der von Facebook
- ein Cross-post-System, mit dem Nachrichten direkt auf anderen Social-Media-Seiten gepostet werden

3.2.1 Backbone.js

Backbone.js[5] ist eine JavaScript-Bibliothek, welche mit dem Zweck entwickelt wurde, die Darstellung und die Programmlogik im Frontend stärker zu trennen und zu strukturieren. Erreicht wird dies durch die Benutzung eines MVC-Patterns auf der Client-Seite. Vor allem bei größeren Single-Page-Anwendungen kann es durch die komplexe Präsentationslogik schnell zu einer unübersichtlichen Menge an jQuery-Callbacks kommen, die alle an einzelne DOM-Elemente gebunden sind. Dies versucht Backbone.js zu verhindern.
Die erste Version von Backbone.js wurde 2010 von Jeremy Ashkenas veröffentlicht[6].
Das MVC-Pattern spiegelt sich im Aufbau von Backbone.js wider. Es gibt drei wichtige Module, die für die Benutzung von Backbone.js wichtig sind: das Modell (*Model*), die Ansicht (*View*) und der Router.
Modelle halten die interaktiven Daten und große Teile der umgebenden Logik, wie Validierungen, Konversionen und Zugriffsrechte. Das Modell interagiert mit dem Server über eine REST-Schnittstelle. Wenn eine Menge an Objekten gehalten werden sollen, können *Collections* benutzt werden.
Die **Anischt** ist dafür zuständig, wie die Daten präsentiert werden. Einerseits ist es für die Einbindung von Templates zuständig und andererseits für die Abhandlung von Events. Jede Ansicht ist an ein DOM-Element gebunden, so wird eine modulare Strukturierung in der Präsentationsebene erreicht.
Mit dem **Router** lässt sich definieren, was passieren soll, wenn eine bestimmte URL aufgerufen wird. An eine Route wird eine Funktion gebunden, die aufgerufen wird, wenn die URL aufgerufen wird. Der Router delegiert dann, welche Modelle benutzt und welche Ansichten angezeigt werden.

3.2.2 Handlebars.js

Handlebars.js[7] ist ein Template-System, womit Templates definiert werden können, die z.B. von Ansichten in Backbone.js eingebunden werden können. Handlebars.js ist eine Erweiterung der Template-Engine Mustache und fügt eine minimale Logik hinzu, wie *if-then-else*-Bedingungen und *each*-Schleifen.
Die Handlebars-Templates werden in normaler HTML-Syntax beschrieben. Mit doppelt geschweiften Klammern kann dynamisch Inhalt eingefügt werden. Ein Beispiel wäre die Ausgabe von Benutzerinformation:

[5] http://backbonejs.org/
[6] http://backbonejs.org/#changelog
[7] http://handlebarsjs.com/

```
<div id="user-informations">
  <h2>{{user.nickname}}</h2>
  <div class="information">
    <p>Vorname: {{user.name}}</p>
    <p>Nachname: {{user.surname}}</p>
        <p>Alter: {{user.age}}</p>
  </div>
</div>
```

Angenommen Handlebars würden mit Backbone.js benutzt werden, so kann es ein Modell *User* geben mit den Properties *name, surname* und *age*. Der Router initialisiert das Modell, welches die Daten vom Server holt, und die Ansicht, welche das Modell übergeben bekommt. Die Ansicht bindet das Template an das DOM-Element mit der ID *user-informations* und die Template-Expressions werden mit den Daten des Modells gefüllt.

3.3 Graphdatenbanken

Als Graphdatenbanken werden Datenbanksysteme bezeichnet, die auf einem graphenbasierten Datenbankmodell beruhen. Die Daten werden über ein Datenbankmanagementsystem (DBMS) in Form von Knoten und Kanten abgefragt, wobei sowohl Knoten als auch Kanten beschreibende Eigenschaften (*Properties*) besitzen können. Dies steht im Gegensatz zu den relationalen Datenbanken, dessen Datenbankmodell tabellenbasiert sind und in denen über die Referenzierung von Tabellenzeilen Daten abgefragt werden. Durch die Repräsentation der Daten als Graph können vor allem in Verbindung stehende Daten sehr gut dargestellt werden. Ein erläuterndes Beispiel wäre die Speicherung von Personen und Gruppen und deren Beziehungen zueinander, dargestellt in Abbildung 3.4. Um das gleiche Beispiel in einer relationalen Datenbank zu speichern, müssten mindestens drei Tabellen angelegt werden. Eine für Personen, eine zweite Tabelle für die Gruppen und eine dritte Tabelle für die Beziehungen. Aus diesem Grund ist es schwer die Beziehungen zwischen Daten aus der visuellen Abbildung eines relationalen Datenbankmodells zu sehen. Graphdatenbankenmodelle hingegen lassen sich sehr einfach visuell abbilden und können dazu benutzt werden bekannte Graphstrukturen zu identifizieren, wie zum Beispiel Cliquen und Hotspots. Darüber hinaus bieten Graphdatenbanken sehr schnelle und effiziente Algorithmen, wenn es darum geht, den Graphen zu traversieren. Relationale Datenbanken müssen hingegen auf die ressourcenhungrigen JOIN-Operationen zurückgreifen, um Datentabellen in Beziehung zu setzen. Daher eignen

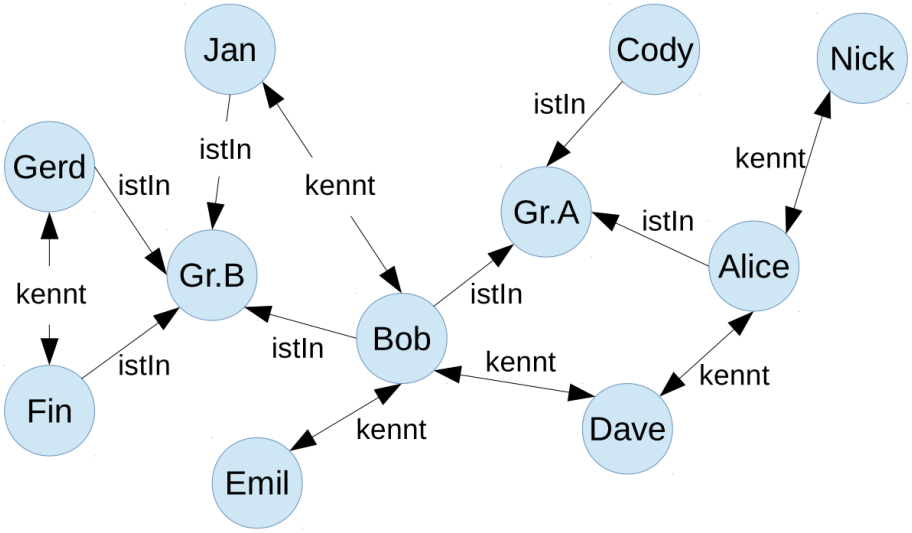

Abbildung 3.4: Graphenbeispiel mit Personenknoten und Gruppenknoten. Personen können in Gruppen sein und Personen können sich untereinander kennen.

sich Graphdatenbanken besonders für die Speicherung von Beziehungsdaten in *Social Web* Anwendungen. Auch im semantischen Web lassen sich Graphdatenbanken sehr gut benutzen. Jedes RDF-Tripel kann auch als gerichteter Graph dargestellt werden. Somit können alle semantischen Daten einfach in Graphdatenbanken übertragen werden.
In einem Vergleich zwischen relationalen und Graphdatenbanken lassen sich folgende Vorteile für Graphdatenbanken finden: (Robinson u. a., 2013; Vicknair u. a., 2010)

- **Performanz** – Vicknair u. a. (2010) haben in Tests gezeigt, dass Graphdatenbanken bei traversierenden Abfragen und Volltextsuchen vor allem in höheren Datensatzbereichen deutlich schneller sind.

- **Flexibilität** – Die Graphdatenbank ist einfach erweiterbar und es können einfach neue Beziehungen, Knoten und Typen hinzugefügt werden. Bei den relationalen Datenbanken muss am Anfang feststehen, was in der Datenbank stehen soll, da eine Erweiterung der Tabellen nur schwer möglich ist.

- **Agilität** – Aufgrund dieser Flexibilität eignen sich Graphdatenbanken gut für die agile und testgetriebene Softwareentwicklung von heute. Das heißt, die Datenbank wächst mit der Anwendung.

Relationale Datenbanken haben dagegen die Vorteile, dass sie erstens bei *set*-orientierten Abfragen, wie es zum Beispiel beim *Online Analytical Processing* benötigt wird , sehr schnell sind und zweitens das Datenbankmodell und die standardisierte Abfragesprache SQL weit verbreitet, gut dokumentiert und getestet sind. Dies macht den Wechsel auf andere Implementationen einfach, da SQL als weit verbreiteter Standard faktisch überall benutzt wird.
Eine bekannte API zum Zugriff auf Graphdatenbanken ist Blueprints, welche eine API in Java anbietet. Es gibt eine Reihe von zusätzlichen Werkzeugen, die auf Blueprints aufbauen, so auch die Traversiersprache *Gremlin*, mit der Graphen abgefragt werden können und die mit vielen JVM-basierten Sprachen funktioniert. Um in dem obigen Beispiel alle Mitglieder von Bobs Gruppen zu bekommen, würde die Abfrage und die korrespondierende Ausgabe in Gremlin die folgende sein:

```
1 > v.outE('istIn').inV.inE('istIn').outV.name
2 ==> Bob
3 ==> Cody
4 ==> Alice
5 ==> Bob
6 ==> Jan
7 ==> Gerd
8 ==> Fin
```

3.3.1 Rexster

Rexster[8] ist ein Graph-Server, der über ein JSON-basiertes REST-Interface oder einem binären Protokoll, genannt RexPro, auf jeden *Blueprints*-Graph zugreifen kann. Darüber hinaus Rexster über das Browser-basierte Interface *Dog House* Graphen visualisieren.
Auf der REST-Seite bietet Rexster eine Erweiterung mit der Travesiersprache **Gremlin** an. Dort können über das REST-Interface Gremlin-Skripte ausgeführt werden, um vom Client ad-hoc auf den Graphen im Backend zuzugreifen.
Das binäre Protokoll RexPro arbeitet nicht viel anders als die Gremlin-Erweiterung, hat jedoch eine bessere Performanz, da es ohne die REST HTTP Infrastruktur auskommt. Ein eingebetteter RexPro-Server wird mit dem Rexster-Server gestartet und hört auf einem TCP-Port auf Anfragen vom Client. Offiziell ist nur ein Java-Client für RexPro verfügbar, es existieren jedoch für viele andere Sprachen 3rd-Party Implementierungen.

[8] https://github.com/tinkerpop/rexster

4 Konzept eines sozialen Netzwerks mit E-Learning-Erweiterung

Dieses Kapitel legt das Konzept für ein System dar, welches in Englisch mit dem Begriff *e-learning enabled Online Social Network* bezeichnet wird. Auf Deutsch könnte es mit dem Begriff eines mit E-Learning Funktionen erweiterten sozialen Netzwerks beschrieben werden. Es umschreibt die Idee, ein offenes soziales Netzwerk mit den Funktionen einer Lernplattform zu verbinden. Das Konzept ermöglicht es, die hohe Kommunikationsbereitschaft der Teilnehmer von sozialen Netzwerken in ein Lernumfeld zu übertragen. Das Ziel ist es, eine Lernumgebung zu schaffen, die dem Anwender erlaubt, eigene Themen zu erarbeiten ohne dabei auf einen Dozenten angewiesen zu sein. Das System hilft dem Anwender, neue Themen zu entdecken, Inhalte zu finden und über die Aktivitäten anderer Nutzer Informationen zu erhalten. Der Anwender kann Tags folgen, um seine Interessen zu beschreiben. Die Lernumgebung hält den Anwender ständig mittels eines Neuigkeitenstroms über seine Interessen auf dem Laufenden. Es zeigt ihm weitere Inhalte, die ihn interessieren könnten, weitere Themen, die weiterführend oder auf dem Interessengebiet aufbauen, und Lerngruppen, denen er beitreten kann, um sein Wissen in einem sozialen Umfeld zu erweitern.

Es gibt zwei Komponenten der hier vorgestellten Lernumgebung. Einerseits die Lerngruppenkomponente, die ausführlich in den Berichten von Brauer und Schmidt (2012) und Brauer u. a. (2013) beschrieben wurde und in Abschnitt 4.2 für das Verständnis des Gesamtkonzepts erläutert wird. Andererseits das semantische Content-Netz, welches das Thema der vorliegenden Arbeit ist und von dem Konzept und Implementierung vorgestellt werden.

Die erste Komponente hilft dem Anwender, Beziehungen zu anderen Benutzern mit den selben Interessen aufzubauen und mit Ihnen zusammen in Gruppen an kollaborativen Aufgaben und Themen zu arbeiten. Das Konzept baut auf Personal Learning Environments (PLEs) auf, eine persönliche Lernumgebung, auf die in Kapitel 2.3.4 im Zuge des DEPTHS-Beispiels eingegangen wurde. PLEs stellen den Benutzer und seine sozialen Beziehungen in den Mittelpunkt und überlassen ihm die Kontrolle. Für

die sozialen Beziehungen, die in Personal Learning Environements entstehen, hat sich der Term Personal Learning Networks (PLN) entwickelt. Dabei geht es nicht nur um die Beziehungen zwischen den Lernenden, sondern um die Beziehungen zwischen den Lerngruppen, den zu bearbeitenden Themen und dem Inhalt.
Die andere Komponente und der Kern der vorliegenden Arbeit ist das semantische Content-Netz, welches das Verwalten und Verknüpfen der Inhalte übernimmt. Dabei werden Lerninhalte mit Metadaten in ein Lernobjekt gebündelt und in ein Netz mit anderen Lernobjekten hinzugefügt und mit diesen semantisch verknüpft. Die semantische Verknüpfung wird dabei von einem Verknüpfungsmodul übernommen. Dieses ist ein separat laufender Prozess, der Lernobjekte über Relationen miteinander verknüpft. Die Relationen werden einerseits aus den Metadaten der Objekte generiert und andererseits aus schon bestehenden Relationen geschlussfolgert. Die Objekte mit ihren Relationen werden anschließend in einer Graphdatenbank gespeichert, wobei die Objekte die Knoten und die Relationen die Kanten sind. Mit diesem Content-Netz ist es einem Anwender leicht möglich, zusammenhängende Daten zu finden und neue Daten einzupflegen. Darüber hinaus können dem Anwender aufgrund seiner Interessen weitere Content-Vorschläge unterbreitet werden.

4.1 Anwendungsszenarien

Nutzer interagiert mit Lerninhalten

Der Benutzer meldet sich mit seinem Benutzernamen und Passwort an und landet auf der Startseite der Anwendung, dem so genannten *Stream*. Der Stream ist eine Übersichtsseite für den Benutzer und zeigt ihm alle Neuigkeiten, chronologisch geordnet. Er sieht Posts von sich und anderen Nutzern und Neuigkeiten von beigetretenen Lerngruppen. Daneben gibt es die Aktivitätsseite, die dem Nutzer hilft, sich einen Überblick über seine Interessen und Aktivitäten zu machen. Sie ist nach Themengebiet geordnet, um eine überschaubare Darstellung der Inhalte zu erreichen. Der Benutzer sieht hier Lerninhalte, die er sich angeschaut und an denen er gearbeitet hat und andere, die ihn interessieren könnten, basierend auf seinen Präferenzen.
Der Benutzer will sich mit einem neuen Themengebiet befassen. Er öffnet das Suchfenster und gibt Stichworte für das neue Interesse ein (z.B. Pest, Europa, 16.Jhdt). Das System liefert ihm Lerninhalte zurück in Form von Artikeln, Abschlussarbeiten, Videos, Podcast, usw., die mit dem Suchbegriffen übereinstimmen. Des Weiteren werden Lerngruppen angezeigt, die sich mit diesem Thema befassen, dazu Posts sowie erfahrene Benutzer. Es

besteht die Möglichkeit, die Themen einzugrenzen oder zu erweitern und auch weiterführende Themen werden angezeigt (z.B. Lebensumstände im 16. Jhdt oder Hexenjagd in Europa usw.). Der Benutzer durchsucht die Ergebnisse und kann Einträge kommentieren und zu seinen Favoriten hinzufügen, um später gezielter darauf zugreifen zu können.
Der Benutzer will zum Themengebiet beitragen, indem er neuen Lerninhalt (*Content*) erstellt. Diesen Content kann er auf drei unterschiedliche Weisen erstellen. Entweder referenziert er eine Ressource per URL, er lädt eine Datei auf den Server hoch oder er erstellt lokal einen neuen „Artikel“, ähnlich einem Wiki-Artikel. Diesen kann er über einen Editor per Wiki-Markup erstellen.
Der Benutzer will eine entfernte Ressource verlinken und in das System einfügen. Über eine „Create Content“-Schaltfläche gelangt er auf die Erstellungsseite, wo er auswählt, ob er eine Referenz oder lokalen Content erstellen will. Er wählt Referenz und trägt nun die nötigen Daten ein, wie Titel, URL, Autor, Datum und weitere inhaltliche und technische Daten. Nach dem Erstellen des Contents wird der Benutzer auf die Detail-Seite des Lerninhalts geschickt, die auch andere Anwender sehen, wenn sie den Content aufrufen. Das Objekt kann nun auch von anderen Benutzern gefunden werden, die die gleichen Interessen haben.
Wenn der Nutzer den Lerninhalt mit anderen Benutzern teilen will, so kann er dies tun, indem er den Lerninhalt in eine Nachricht einbindet und ihn entweder auf seinem Stream postet, so dass seine Freunde es sehen können oder als private Nachricht verschickt.

Nutzer interagiert mit Lerngruppen

Der Benutzer kann Lerngruppen beitreten, mit deren Mitgliedern er gemeinsam an einer Aufgabenstellung, einem so genannten **Thema** (*Topic*) arbeitet. Um dies zu erstellen braucht er eine Beschreibung, die den Umfang und das Ziel der Aufgabenstellung umfasst. Daneben können Lernobjekte verlinkt werden, die zur Bearbeitung der Aufgabenstellung benötigt werden oder sie vereinfachen. Themen stellen den Start für kollaborative Tätigkeiten im System dar. Beispielsweise könnte gemeinsam ein Lernobjekt erstellt oder ein Thema diskutiert werden. Innerhalb der Gruppe können Dinge gepostet und kommentiert werden. Ein Benutzer kann einer öffentlichen Gruppe ohne Beschränkung beitreten oder er wird zu einer privaten Gruppe eingeladen. Der Benutzer hat das Lernziel der Aufgabenstellung erfüllt und kann diese markieren, so dass sie für ihn gesperrt wird und nicht mehr auf seiner Übersichtsseite auftaucht.
Der Benutzer möchte eine Lerngruppe erstellen, dazu muss ein Thema ausgewählt werden, welches bearbeitet werden soll. Außerdem muss ein Name und eine Beschreibung der

Gruppe angegeben werden. Der Benutzer wird automatisch Mitglied der Gruppe, nimmt aber keinen Sonderstatus als Admin oder ähnliches ein. Er hat nun die Möglichkeit weitere Benutzer einzuladen oder Anfragen von anderen Benutzern anzunehmen, die dann zusammen an den Themen arbeiten können. Der Benutzer möchte nun aus der Lerngruppe austreten. Die Lerngruppe wird für ihn gesperrt, die anderen Benutzer können jedoch noch weiter in der Lerngruppe arbeiten.

4.2 Lerngruppenkomponente

Zum Verständnis der Zusammenhänge in dem System zwischen den beiden E-Learning-Komponenten, wird in diesem Abschnitt der Aufbau und die Funktionsweise der Lerngruppenkomponente erläutert.
Zu der Lerngruppenkomponente gehören die Lerngruppen und die Themen. Diese sind mit den Lernobjekten die zentralen Komponenten für das kollaborative Arbeiten in der Lernumgebung. Das Konzept der Lerngruppen und Themen basiert auf Personal Learning Environments. Wobei die sozialen Beziehungen, die in einem PLE aufgebaut werden, mit dem Begriff Personal Learning Network umschrieben werden. Ziel der Lerngruppenkomponente ist es, den Benutzern eine Umgebung zu liefern, in der sie ihre sozialen Beziehungen beibehalten, gekoppelt mit formalen, offenen Lerngruppen. Der Lerner ist so imstande, Kontakt zu anderen Lernenden aufzubauen und mit ihnen zusammen in einer Gruppe kollaborativ an einem Thema zu arbeiten.

4.2.1 Lerngruppen

Eine Lerngruppe ist eine Gruppe von Anwendern, welche zusammen an einer bestimmten Aufgabenstellung arbeiten. Eine Lerngruppe kann öffentlich oder privat sein. Wenn die Gruppe offen ist, können andere Benutzer der Gruppe jederzeit beitreten, solange sie aktiv ist. Wenn die Gruppe privat ist, können nur eingeladene Benutzer der Gruppe beitreten. Während die Gruppe aktiv ist, können die Mitglieder jederzeit austreten, auch wenn das Lernziel noch nicht erreicht ist. Die Mitglieder können für sich abstimmen, wenn sie der Meinung sind, dass das in dem Thema beschriebene Ziel erreicht oder ein weiteres Arbeiten daran nicht gewünscht ist. Wenn alle Mitglieder dafür gestimmt haben, kann die Gruppe geschlossen werden. Eine Lerngruppe kann drei verschiedene Zustände haben: **Nichtexistent**, **aktiv** oder **gesperrt**. Die Abbildung 4.1 zeigt die Zustände und deren Übergänge als Graph.
Ein Benutzer kann eine Gruppe erstellen mit einem Gruppennamen, einer Beschreibung

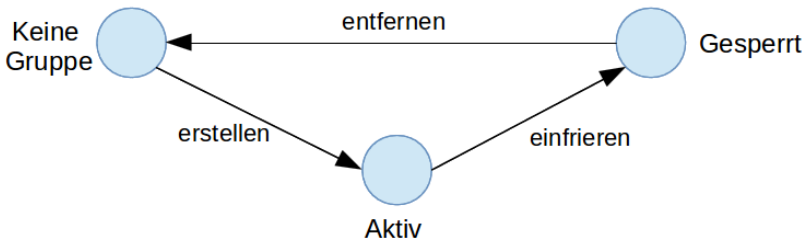

Abbildung 4.1: Darstellung der Zustände und Übergänge der Lerngruppen als Graph

und der Sichtbarkeit. Damit wechselt die Gruppe in den Zustand *aktiv*. Im aktiven Zustand kann Lerninhalt geteilt und kollaborativ an dem Thema gearbeitet werden. Wenn die Gruppe der Meinung ist, dass die Aufgabe erledigt ist und die Gruppe geschlossen werden kann, wird die Gruppe in den Zustand *Gesperrt* gesetzt. Nun können keine neuen Benutzer mehr der Gruppe beitreten, aber die Gruppe ist archiviert und noch für ehemalige Mitglieder sichtbar.
Eine wichtige Funktion des Lerngruppenmoduls ist es, die Lerngruppenbildung der Anwender aktiv zu unterstützen. Um das zu erreichen, stellt das System Vorschläge für mögliche Gruppenmitglieder bereit. Diese basieren auf den Tags der Themen und dem Verbindungen der Benutzer untereinander. Die Beiträge zu einer Gruppe können privat sein, das heißt, sie sind nur für Mitglieder innerhalb der Lerngruppe sichtbar oder sie können öffentlich sein, das heißt, dass auch Nicht-Mitglieder die Nachrichten innerhalb der Gruppe sehen kann. Gruppenmitglieder müssen nicht im System befreundet sein. Der Benutzer hat jedoch die Möglichkeit die Mitglieder einer Gruppe zu seinen Freunden hinzuzufügen.
Die Lerngruppenbildung innerhalb der Lernumgebung wird mittels folgender Kriterien bestimmt: (Brauer und Schmidt, 2012)

- Die Nutzer haben einen ausgeglichenen Lernstil
- Die Nutzer haben übereinstimmende Tags mit dem Thema
- Die Nutzer sind im Graph dicht beieinander

Der Lernstil wird anhand eines Fragebogens erkannt, den die Nutzer bei der Registrierung ausfüllen. Dieser baut auf dem Modell und den Theorien von Felder und Silverman (1988) auf.
Das zweite Kriterium kann mittels der Tags in der Lernumgebung erfüllt werden. Während der Lerngruppenbildung werden die gemeinsamen Tags zwischen den Nutzern

betrachtet, die für das Thema benötigt werden.
Um die Dichte der Gruppe zu bestimmen, wird die Entfernung im Graph zwischen den Benutzern gemessen.
Ausgehend von dem Benutzer, der die Gruppe gegründet hat, wird das Netzwerk durchsucht und die Nutzer anhand der oben genannten Kriterien bewertet. Für jedes Kriterium gibt es eine Formel, mit den die Nutzer einem Wert zugewiesen bekommen können. Anhand von genetischen Algorithmen wird die beste Kombination von Nutzern für die Gruppe gewählt. Die genaue Funktionsweise und der Aufbau der Formeln wird ausführlich in Brauer und Schmidt (2012) beschrieben.

4.2.2 Themen

Themen oder *Topics* sind der Ausgangspunkt zum kollaborativem Arbeiten im System. Sie haben einen Titel und eine Beschreibung. Es wird eine Tätigkeit beschrieben, um die sich eine Lerngruppe bilden kann. In der Beschreibung des Themas soll auch ein Ziel angegeben werden, was es von den Teilnehmenden einer Lerngruppe zu erreichen gilt. Eine automatisierte Auswertung dieses Ziels soll es nicht geben. Um das Finden eines Themas und das Einstufen ihrer Komplexität zu ermöglichen, können Themen mit Tags markiert werden. Die thematische Grundlage von Themen bilden Lernobjekte. Eine Lerngruppe wird auf einem Thema gestartet, das heißt, wenn eine Gruppe gebildet wird, muss ein Thema angegeben werden. Die Abbildung 4.2 verdeutlicht die Beziehungen zwischen den Komponenten.

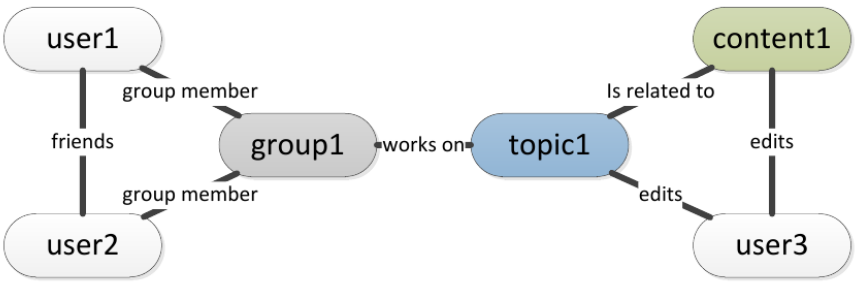

Abbildung 4.2: Die Beziehungsstruktur der E-Learning-Komponenten (Brauer und Schmidt, 2012)

4.2.3 Tags

Ein Tag ist ein Schlüsselwort oder Begriff, welches an eine Entität angehängt werden kann, um so diese „getaggten“ Objekte ordnen und gruppieren zu können. Das Prinzip des „Taggings“ ist somit eine Art Indizierung über Schlagwörter. Nachrichten und Posts können Tags enthalten und so einer oder mehreren Kategorien zugeordnet werden. So ist es möglich, Posts nach Tags zu filtern, um dann nur die Posts zu sehen, die den entsprechenden Tag aufweisen.
So können nun auch Lerninhalte, Lernthemen und Lerngruppen mit Tags versehen werden. Dies hilft mehr Struktur in das Lernumfeld des Anwenders zu bringen.

4.3 Semantisches Content-Netz

Die andere Erweiterung des sozialen Netzwerks bildet das semantische Content-Netz. Dieses orientiert sich am Konzept eines Learning Content Management System (LCMS), vorgestellt in Kapitel 2.3.1. Das semantische Content-Netz erstellt, verwaltet und organisiert die Lernobjekte.
Für die Erstellung wird dem Benutzer eine Oberfläche gegeben, in der er Lerninhalte hochladen oder verlinken kann und die Möglichkeit hat, dieses mit Metadaten zu bestücken. Die Verwaltung der Lernobjekte wird von einer relationalen und einer Graphdatenbank und einem serverseitigen Repository übernommen. Die relationale Datenbank hält dabei die Attribute und Metadaten des Lernobjekts. Die Graphdatenbank speichert die semantischen Beziehungen der Lernobjekte und das Repository speichert den Content. Die Organisation der Lernobjekte wird von einem Verknüpfungsmodul übernommen. Dabei werden die Lernobjekte aufgrund ihrer Metadaten und aus Schlussfolgerungen bestehender Verknüpfungen miteinander in Beziehung gesetzt.
Das Ziel ist es, dem Benutzer eine Lernumgebung zu geben, die auf seine Interessen eingeht und ihm auf seine Lernpräferenzen und seine Themenkenntnis zugeschnittene Inhalte liefert. Eine weitere Anforderung an das System ist, dass es modular gestaltet wird. Das bedeutet, dass semantische Content-Netz soll das soziale Netzwerk erweitern und nicht in es integriert werden. So ist es möglich diese Erweiterung schnell und einfach an weitere Server anzubauen.

4.3.1 Lernobjekte

Als Content wird der gesammelte Lerninhalt innerhalb des Systems benannt. Lernobjekte umhüllen den Content und enthalten Metadaten über den Content. Das Prinzip des Lernobjekts wurde in Kapitel 2.3.2 erläutert. Wie sich das in dieses System übertragen lässt, wird im Folgenden erörtert.
Bei der Struktur der Lernobjekte wird sich an hylOs orientiert. Dort ist der Aufbau der Lernobjekte nach dem LOM-Standard entwickelt worden. Wie in Absschnitt 2.3.2 erwähnt, definiert das LOM-Modell zahlreiche Metadaten für Lernobjekte, die in Kategorien aufgeteilt sind. Es wurde gezeigt, dass viele der Metadaten nur spärlich benutzt werden. Daher sollten einerseits die Metadaten automatisch eingetragen werden und andererseits dem Benutzer geholfen werden, die Daten einzutragen. Die Metadaten aus allgemeinen und technischen Kategorie können weitestgehend über einen Metadatenextraktor eingetragen werden. Dabei werden entweder aus Webseiten die <meta>-Tags oder aus Mediendaten die technischen Metadaten ausgelesen. Bei den verbliebenen Metadaten kann dem Benutzer über *Auto-Complete-* oder *Snap-to-Grid-*Techniken geholfen werden. Bei diesen Techniken wird dem Benutzer eine Auswahl an Eingabemöglichkeiten gegeben, die auf schon vergangenen, gespeicherten Eingaben beruhen. Bei den didaktischen Angaben wird dies jedoch schwierig umzusetzen sein. Hier wird die Eingabe der didaktischen Angaben damit erleichtert, dass dem Benutzer eine Auswahl an beschränkten Möglichkeiten gegeben wird. So wird zum Beispiel der Schwierigkeitsgrad in 5 Stufen unterteilt.
Die Lernobjekte dienen keinen konkreten Lernzielen. Mit den didaktischen Angaben lässt sich dem Lernobjekt ein Lernkontext geben. Das Lernziel wird im Thema von dem Benutzern vorgegeben. Sie entscheiden, ob der mit dem Lernobjekt angegebene Lernkontext zu dem Lernziel des Themas und den Ansprüchen der Lerngruppe passt.
Zur Erstellung der Lernobjekte gibt es drei Möglichkeiten. Erstens eine entfernte Ressource zu verlinken, zweitens eine Mediendatei hochzuladen und drittens einen lokalen Artikel zu erstellen. Wenn das Lernobjekt über eine Referenz erstellt wird, muss eine URL angegeben werden, worüber die entfernte Ressource zu erreichen ist. Somit ist der Content des Lernobjekts nicht lokal gespeichert und unveränderbar. Die Mediendaten können in bestimmten Formaten hochgeladen werden. Die Daten werden auf dem Server gespeichert und können über eingebettete Player und Betrachter angesehen werden. Die dritte Möglichkeit ist es, einen lokalen Artikel zu erstellen, ähnlich einem Wiki-Eintrag, der vom User erstellt und verändert werden kann. Dies kann von Definitionen eines Begriffs, über eine Zusammenfassung eines bestimmten Themas, bis hin zu einer Beant-

wortung häufig gestellter Fragen gehen. Der Artikel wird über einen Editor erstellt und auf dem Server gespeichert. Er kann Links zu anderen Lerninhalten aufweisen, im Text oder als Quellenangaben.
Der Anwender, der das Lernobjekt erstellt hat, ist Eigentümer des Objekts und hat die volle Verfügungsgewalt über das Objekt. Er kann das Objekt löschen oder verändern, wobei alle Veränderungen protokolliert werden. Er hat auch die Möglichkeit das Objekt zur Bearbeitung an andere User oder Gruppen freizugeben, wodurch zum Beispiel bei einem Artikel dieser von mehreren Leuten bearbeitet werden kann.
Lernobjekte können auch mit Tags versehen werden, dabei werden die Schlüsselworte automatisch als Tags des Lernobjekts übernommen. Weitere Tags können auch manuell vom Benutzer eingetragen werden. Darüber hinaus bilden Lernobjekte die Grundlage von Themen, an denen in Lerngruppen kollaborativ und zu einem Lernziel hin gearbeitet werden kann. Lernobjekte dienen dazu, Themen eine inhaltliche Grundlage zu liefern, die den Anwendern hilft, das Lernziel zu erreichen.
Der Aufbau der Lernobjekte lässt sich in einem Content-Modell darstellen. Abbildung 4.3 skizziert, wie die Lernobjekte im System strukturiert werden.

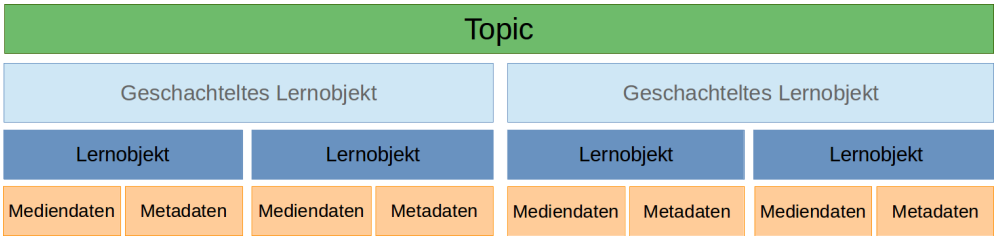

Abbildung 4.3: Darstellung des Aufbaus der Lernobjekte. Lernobjekte bestehen aus einen oder mehreren Mediadaten, die ineinander geschachtelt werden können. Mehrere Lernobjekte können als Grundlage für Topics von Usern bearbeitet werden.

Grundlage bilden die Mediendaten und die Metadaten. Die Mediendaten können Videos, Bilder, Audiodateien, Dokumente, Webseiten oder andere Daten in diversen Formaten sein. Die Metadaten helfen einerseits die Mediendaten zu beschreiben, andererseits dem Lernobjekt einen Kontext zu geben, in welchem Umfeld es eingesetzt werden kann. Diese Daten werden von einem Lernobjekt umschlossen. Lernobjekte können geschachtelt werden in andere Lernobjekte, welche dann ein größeres Sinngefüge ergeben, das heißt, ein Lernobjekt ist die Summe seiner Kindobjekte. Die Lernobjekte bilden die Grundlage des Arbeitens in dem System und bilden die Grundlage von Topics, welche ein oder

meherere Lernobjekte mit einem Lernziel versehen, so dass diese von den Benutzern in einer Lerngruppe bearbeitet werden können.
Die Schachtelung der Lernobjekte wird dabei über die Eltern-Kind-Beziehung bewerkstelligt. Um ein Kindsobjekt zu erstellen, wird das Elternobjekt bei der Erstellung ausgewählt. Da Eltern- und Kindobjekte ein Sinngefüge ergeben, werden Elternobjekte zusammen mit ihren direkten Kindobjekten angezeigt. Wenn ein Elternobjekt gelöscht wird, werden die Kindsobjekte nicht automatisch mitgelöscht. Dies hat den Grund, dass ein Kindobjekt auch mehrere Elternobjekte haben könnte, also mehrere Objekte von dem Kindobjekt strukturell abhängig sind.
Lernobjekte und der referenzierte Content dienen den Lernenden als Anhaltspunkt und Grundlage bei der Bearbeitung von Themen. Der Lerninhalt kann dem Lerner dabei sowohl als Information, als auch als Notwendigkeit zur Erfüllung des Themenziels helfen.

Löschen von Lernobjekten

Lernobjekte können mit anderen Komponenten des Lernsystems in Beziehung stehen. Themen benutzen die Lernobjekte als inhaltliche Grundlage, Nutzer können Lernobjekte als Favoriten hinzufügen und Lernobjekte können mit anderen Lernobjekten in Beziehung stehen. Aus diesem Grund dürfen Lernobjekte nicht einfach aus dem System gelöscht werden, wenn der Ersteller der Lernobjekte dies zur Löschung beantragt hat. Es muss eine Möglichkeit gefunden werden, wie das Lernobjekt weiter von den referenzierten Komponenten benutzt werden kann, ohne das formell gelöschte Lernobjekte das System zumüllen.
Daher müssen eine Reihe von Maßnahmen ergriffen werden, um diese Anforderungen zu erfüllen. Wenn ein Lernobjekt zur Löschung beantragt wurde, wird dieses Objekt zuerst **gesperrt**. Eine Sperrung heißt, dass das Lernobjekt nicht mehr über eine Suche gefunden und nicht mehr neu referenziert werden kann, auch nicht von semantischen Verknüpfungen. Damit wird verhindert, dass weitere Beziehungen zu dem Objekt aufgebaut werden. Zweitens wird das Objekt als **deprecated** angezeigt, in den Fällen, wo es noch referenziert ist. Dies ist ein Hinweis für die verbleibenden Nutzer des Objekts, dass das Objekt möglicherweise veraltet oder falsch ist und dient als Aufforderung das Objekt zu entfernen oder zu ersetzen.
Sobald die Beziehungen von den Themen zu den Objekten entfernt wurden, kann das Objekt aus der Datenbank gelöscht werden. Zugleich wird das Objekt auch aus dem semantischen Content-Netz entfernt. Weiteres dazu in Abschnitt 4.3.3.

Speichern von Lernobjekten

Alle Relationen werden einer Graphdatenbank gespeichert. Die Objekte stellen dabei die Knoten dar, die Verknüpfungen sind die Kanten zwischen den Knoten. Neben der ID wird bei einem Knoten ein Erstellungsdatum und zwei Markierungen (*Flags*) gesetzt, die einerseits angeben, ob das Objekt schon verknüpft wurde oder nicht. Dies ist wichtig, um sicherzugehen, dass der Verknüpfungsprozess komplett durchgelaufen ist. Andererseits ob das Objekt gelöscht werden soll. In der Kante wird der Relationsname und die Ableitungen gespeichert, falls es sich um eine geschlussfolgerte Relation handelt, sonst wird das Feld leergelassen. So lässt sich auch einfach nachprüfen, ob es sich bei der Relation um eine geschlussfolgerte Relation handelt oder nicht.

4.3.2 Relationen

Die Relationen basieren auf den Arbeiten von Engelhardt u. a. (2006a). Dort wurde sich an den Relationen des DublinCore-Metadatenstandards orientiert, ein bibliografisches Metadatenformat. Im Zuge dieses Metadatensatzes wurden Relationen definiert, um sowohl digitale, als auch physische Mediendaten zu beschreiben und in Beziehung zu setzen. Diese Relationen wurden für die Lernumgebung hylOs in das E-Learning-Umfeld übertragen, um Lernobjekte semantisch zu verknüpfen. Dabei wurden die Relationen angepasst und weitere Relationen hinzugefügt. Zusätzlich wurden die Eigenschaften der Relationen in einer Ontologie und es wurden Schlussfolgerungsregeln definiert. Dieses Konzept und die Implementierung in hylOs hat Lange (2006) in ihrer Diplomarbeit nochmals vertieft. Unter anderem hat sie ausgearbeitet, aus welchen Metadaten die Relationen erkannt werden können.
Für die Verknüpfung der Lernobjekte in dem semantischen Content-Netz wurden diese Relationsdefinitionen verwendet. Dabei wurden zwei Relationen nicht implementiert. Dies wird in den nächsten Abschnitten aufgegriffen, wo tiefer auf die Relationen eingegangen wird.
In der Ontologie gibt es die Klasse *LearningObject* mit den Relationen als *Properties*. Die Relationen können verschiedene Eigenschaften aufweisen: **transitiv**, **symmetrisch** und **invers**.
Transitivität bedeutet, wenn es eine Relation X von Objekt A nach Objekt B und von Objekt B zu Objekt C gibt, so gibt es auch die Relation X von Objekt A nach Objekt C (siehe Abb. 4.4).

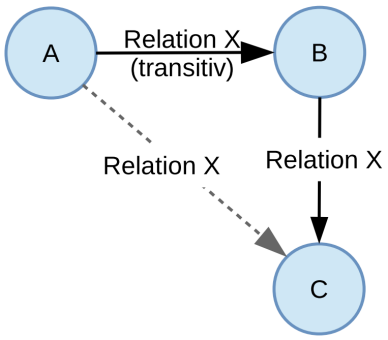

Abbildung 4.4: Die Relationseigenschaft der **Transitivität**

Symmetrie bezeichnet es, wenn es eine Relation X von Objekt A nach Objekt B gibt, so besteht die gleiche Relation X auch von Objekt B nach Objekt A (siehe Abb. 4.5). Eine inverse Relation ist die Umkehrung einer Relation, das heißt, wenn es eine Relation X von Objekt A nach Objekt B gibt, so besteht die inverse Relation von X von Objekt B nach Objekt A (siehe Abb. 4.5).

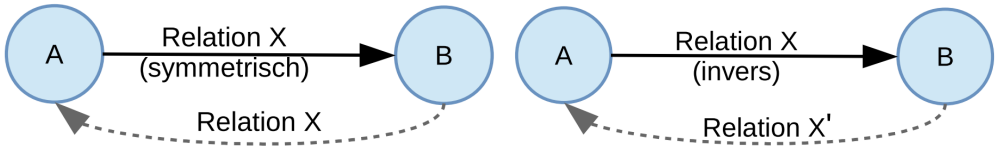

Abbildung 4.5: Die beiden Relationseigenschaften: **Symmetrie** und **Inversität**.

Inversität und Symmetrie sind redundante Eigenschaften, die nur vollständigkeitshalber angegeben werden. Transitivität dagegen kann bei Schlussfolgerungen zu neuen Erkenntnissen führen. Die Relationen werden in drei Unterkategorieren aufgeführt: Abhängigkeitsbeziehungen, Taxonomiebeziehungen und Inhaltsbeziehungen.

Abhängigkeitsbeziehungen

Unter Abhängigkeitsbeziehungen fallen Relationen, die eine Abhängigkeit von zwei Objekten ausdrücken, entweder eine strukturelle Abhängigkeit oder eine inhaltliche Abhängigkeit. Im LOM-Standard können Lernobjekte geschachtelt werden. Diese Schachtelung kann als Elternbeziehung ausgedrückt werden. Ein Objekt A kann aus mehreren Objekten aufgebaut sein, dieses Objekt A wäre der Elternteil der anderen Objekte. Ein Elternobjekt ist somit die Summe seiner Kindobjekte. Die Relationen, die benutzt werden, um dies auszudrücken sind **isPartOf** und **hasPart**. Diese Relationen sind invers

zueinander und transitiv.
Daneben wird noch überprüft, ob bei einem Objekt die Kenntnis über den Inhalt eines anderen Objekts vorausgesetzt wird. Dies wird mit den Relationen **requires** und **isRequiredBy** gekennzeichnet. Diese Relationen sind *invers* zueinander und *transitiv*.
Bei dem Relationspaar *requires/isRequiredBy* handelt es sich um eine starke inhaltliche Abhängigkeit. Eine schwächere inhaltliche Abhängigkeit drücken die Relationen *isBasedOn/isBasisFor* und *references/isReferencedBy* aus, wobei letztere die schwächste inhaltliche Abhängigkeit angeben. *References/isReferencedBy* werden über die Taxonomie erkannt und werden im nächsten Abschnitt erläutert. *IsBasedOn/isBasisFor* wurde hingegen nicht implementiert. Der Grund ist, dass *isBasedOn/isBasisFor* über die inhaltlichen Metadaten erkannt werden müssten. Es ist jedoch schwierig auf eine Abhängigkeit zweier Objekte nur über die Analyse der Metadaten zu schließen. Die anderen beiden Möglichkeiten wären, dies entweder über die Taxonomie zu machen, jedoch gibt es keine SKOS-Relationseigenschaften, die die *isBasedOn/isBasisFor*-Beziehung beschreiben würden. Oder dies explizit von dem Benutzer angeben zu lassen. Das würde dazu führen, dass der Benutzer die *requires/isRequiredBy*- und die *isBasedOn/isBasisFor*-Beziehung angeben müsste. Der Unterschied der beiden Relationen ist jedoch zu marginal, um diesen dem Benutzer leicht verständlich machen zu können.
In Engelhardt u. a. (2006a) wurde noch die Relation **hasVersion/isVersionOf** definiert. Sie beschreibt, dass ein Lernobjekt eine überarbeitete Version eines anderen Lernobjekts darstellt. Diese Beziehung ist vor allem nützlich, wenn eine Versionverwaltung im System implementiert ist. Aber auch für Objekte, die zwei verschiedene Versionen eines Dokuments referenzieren ist diese Beziehung nützlich. Dies müsste dann allerdings vom Nutzer explizit angegeben werden, da eine automatische Versionserkennung schwierig ist. Da im System noch keine Versionsverwaltung implementiert ist, wird diese Beziehung nicht verwendet.

Relation	**Definition**	**Metadaten**
isPartOf/hasPart	Ein Objekt kann aus mehreren anderen Objekten bestehen.	*Structure*
isRequiredBy/ requires	Das Wissen über den Inhalt eines anderen Objekts wird vorausgesetzt.	*Requirement*

Tabelle 4.1: Übersicht der Abhängigkeitsbeziehungen. Die Spalte *Metadaten* gibt an, welche Metadaten zur Erstellung der Relation überprüft werden.

Taxonomiebeziehungen

Lernobjekte können innerhalb einer Taxonomie geordnet werden. Dabei gibt es hierarchische Beziehungen, das heißt, ein Lernobjekt ist einem direkt höher- oder tieferklassigeren Knoten eines anderen Lernobjekts innerhalb der Taxonomie zugeordnet. Und es gibt assoziative Beziehungen, das heißt, ein Lernobjekt referenziert ein anderes Lernobjekt. Diese Beziehung ist nicht an die taxonomische Baumstruktur gebunden.
Wenn das Objekt A einem direkt verbundenen, höherklassigerem Knoten zugewiesen wird als dem Objekt B, so besteht die Relation **isBroaderThan** von Objekt A zu Objekt B. Genauso wird bei einem tieferklassigerem Knoten die Relation **isNarrowerThan** vergeben. Diese Relationen sind invers zueinander und transitiv.
Bei horizontalen Beziehungen wird von einer Referenz von Objekt A zu Objekt B gesprochen. Dies wird in der Relation **references** bzw. **isReferencedBy** ausgedrückt. Diese Relationen sind auch invers zueinander und transitiv.

Relation	**Definition**	**Metadaten**
isNarrowerThan/ isBroaderThan	Ein Objekt ist einem direkt höheren oder niedrigen taxonomischen Knoten zugeordnet als ein anderes Objekt.	*Taxon Path*
isReferencedBy/ referenced	Ein Objekt weist auf ein anderes Objekt innerhalb des taxonomischen Gefüges hin.	*Taxon:related*

Tabelle 4.2: Übersicht der Taxonomiebeziehungen

Inhaltsbeziehungen

Inhaltsbeziehungen spiegeln eine inhaltliche oder technische Äquivalenz zwischen den Lernobjekten wider. Die vorhandenen Metadaten wurden in Kapitel 4.3.1 vorgestellt. Es können bei der Analyse der Metadaten jedoch keine hundertprozentig richtigen Schlüsse in Bezug auf mögliche Relation gemacht werden. Daher wird von einer Heuristik gesprochen, da nur vermutet wird, dass aufgrund der Vergleiche von den Metadaten eine Relation zwischen den zwei Objekten besteht. Der Heuristik-Reasoner behandelt die Relationen **isFormatOf**, **isAlternativeTo**, **isMoreSpecificThan/isLessSpecificThan** und **isIllustratedBy/illustrates**.
Eine wichtige Rolle bei den Inhaltsbeziehungen spielt die Überprüfung und der Vergleich

des Inhalts (*Content*) zweier Lernobjekte. Um zu ermitteln, ob die Semantik des Contents ähnlich ist, werden 4 Eigenschaften der Lernobjekte verglichen.

1. **Schlüsselwörter** – Haben die Lernobjekte überschneidende Schlüsselwörter?
2. **Sprache** – Benutzen die Lernobjekte die gleiche Sprache?
3. **Klassifikationspfad** – Sind die Lernobjekte denselben Klassifikationen untergeordnet?
4. **Beziehungen** – Besitzen die Lernobjekte die gleichen Relationen zu denselben Lernobjekten?

Die Einschätzung, wieviele Schlüsselwörter, Klassifikationen und Beziehungen zwischen den Objekten gleich sein müssen, muss anhand von Erfahrungswerten gemacht werden. Die Überprüfung von gleichen Relationen beim Ermitteln von gleichem Inhalt ist ein zusätzlicher Schritt, um sicherzugehen, dass zwei Lernobjekte einen ähnlichen Inhalt haben.
Die Relation **isFormatOf** drückt aus, dass zwei Objekte ungefähr den gleichen Inhalt behandeln, jedoch ein unterschiedliches Format besitzen. Die Metadaten *Format*, *Coverage*, *Difficulty*, *Age Range* und der *Content* spielen dabei eine Rolle. **IsFormatOf** ist eine symmetrische Relation.
Die Relation **isAlternativeTo** wird verwendet, wenn zwei Objekte anstatt des jeweils anderen verwendet werden können, also Alternativen zueinander darstellen. Dafür werden der *Content*, *Interactivity Type*, *Learning Resource Type*, *Interactivity Level*, *Semantic Density*, *Intended Enduser Role*, *Context*, *Age Range* und *Difficulty* verglichen. Die Relation **isAlternativeTo** ist *transitiv* und *symmetrisch*.
Die Relationen **isMoreSpecificThan/isLessSpecificThan** bezeichnen, dass ein Objekt einen Sachverhalt konkreter oder weniger konkret beschreibt als das andere Objekt. Die dafür zu vergleichenden Metadaten sind der *Content* und die *Semantic Density*. Die beiden Relationen sind *invers* zueinander und *transitiv*.
Die Relationen **isIllustratedBy/illustrates** geben an, dass ein Objekt den Sachverhalt eines anderen Objekts veranschaulicht oder erläutert. Dies kann in der Form einer grafischen Darstellung sein oder einer textuellen Erklärung. Die Metadaten, die hierfür in Betracht gezogen werden sind der *Content*, *Age Range*, *Resource Type*, *Context* und die *Difficulty*. Die beiden Relationen sind *invers* zueinander und *transitiv*.
Eine Zusammenfassung der Relation, die von dem Heuristik-Reasoner erstellt werden, wird in Tabelle 4.3 gegeben.

Relation	Definition	Metadaten
isFormatOf	Zwei Objekte haben ungefähr den gleichen Inhalt, aber ein unterschiedliches Format.	*Format* *Coverage* *Difficulty* *Age Range* *Content*
isAlternativeTo	Zwei Objekte können als Alternative zueinander verwendet werden.	*Difficulty* *Context* *Age Range* *Interactivity Type* *Interactivity Level* *Learning Resource Type* *Semantic Density* *Intended Enduser Role* *Content*
isLessSpecificThan/ isMoreSpecificThan	Ein Objekt beschriebt einen Sachverhalt konkreter oder abstrakter als ein anderes Objekt.	*Content* *Semantic Density*
isIllustratedBy/ illustrates	Ein Objekt wird durch ein anderes veranschaulicht.	*Content* *Age Range* *Resource Type* *Context* *Difficulty*

Tabelle 4.3: Zusammenfassung der Relationen des heuristischen Reasoners

4.3.3 Schlussfolgerungen

Schlussfolgerungen sind Relationen, die anhand von schon bestehenden und erstellten Verknüpfungen gemacht werden. Dies wird durch eine Schlussfolgerungsengine (*Reasoner*) bewerkstelligt. Die Basis der Engine ist ein Ontologiemodell, in der die Ontologie und Schlussfolgerungsregeln festgehalten werden. Da die Schlussfolgerungen mit schon bestehenden Relationen gemacht werden, müssen die nötigen Relationen dem Ontologiemodell hinzugefügt werden, bevor der Schlussfolgerungsprozess gestartet werden kann.
Dabei müssen nicht alle Relationen dem Ontologiemodell hinzugefügt werden, sondern nur so viele, wie für den Schlussfolgerungsprozess des neu hinzugefügten Objekts benötigt werden. Welche und wieviele Relationen das sind, hängt von dem Aufbau der Regeln ab. Die Schlussfolgerungsregeln sind der Implementierung von hylOs entnommen und

umfassen 61 Regeln. Die meisten Regeln referenzieren nicht mehr als 3 Objekte, das heißt, die meisten Regeln sind in der Form $A \rightarrow C \; if \; A \rightarrow B \wedge B \rightarrow C$ gehalten. In dem Fall reicht es, wenn alle Relationen, die **einen Schritt** von dem neu hinzugefügten Objekt entfernt sind, in das Modell eingegliedert werden. Die Abbildung 4.6 veranschaulicht dieses Konzept.

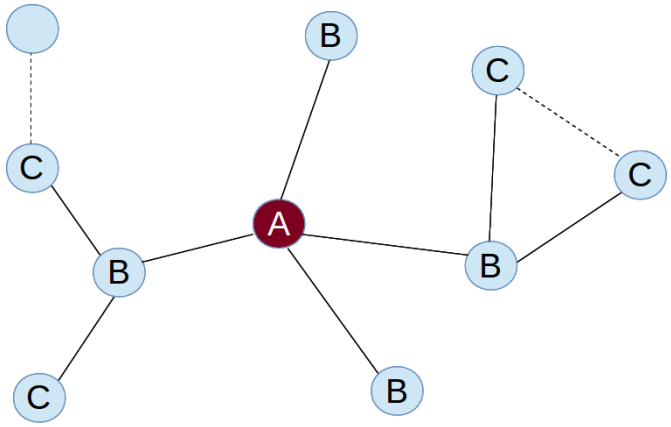

Abbildung 4.6: A ist das neu hinzugefügte Objekt. Bei dem 1-Schritt-Konzept reicht es, die Verknüpfungen von A nach B und B nach C in das Ontologiemodell des Schlussfolgerers hinzuzufügen, da die meisten Regeln nicht über A, B und C hinausgehen.

In dem dargestellten Beispiel ist A das neu hinzugefügte Objekt, dessen neue Relationen geschlussfolgert werden sollen. B sind alle Objekte, welche einen Schritt von A entfernt sind. C sind alle Objekte die zwei Schritte von A entfernt sind. Um die Regeln zu überprüfen reicht es alle Relationen von A nach B und von B nach C in das Modell einzugliedern, um den Schlussfolgerungsprozess auszuführen.
Eine konkrete Implementierung einer Schlussfolgerungsregel wird in Listing 4.1 gezeigt.

```
1 [rule30: (?A lom:IsNarrowerThan ?C) <- (?A lom:IsAlternativeTo ?B)
2         (?B lom:IsNarrowerThan ?C) notEqual(?A, ?C)]
3
4 [rule31: (?A lom:IsBroaderThan ?C) <- (?A lom:IsAlternativeTo ?B)
5         (?B lom:IsBroaderThan ?C) notEqual(?A, ?C)]
```

Listing 4.1: Zwei Schlussfolgerungsregeln

Regel 30 besagt, dass wenn Objekt A eine Alternative zu B ist und B taxonomisch niedriger angeordnet ist, so ist auch anzunehmen, dass Objekt A niedriger angeordnet ist als Objekt C. Regel 31 besagt das Gleiche, nur unter Berücksichtigung taxonomisch

höhergestellten Knoten.
Abbildung 4.7 zeigt ein Beispiel, welches die Anwendung der Regel veranschaulicht.

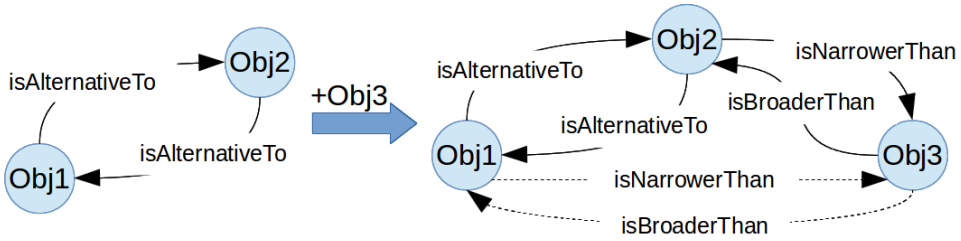

Abbildung 4.7: Beispiel einer Schlussfolgerung nach Regel 30. Obj3 wird zu den bestehenden Objekten hinzugefügt. Die Relationen mit durchgezogenen Linien wurden über Metadaten erstellt, die Relationen mit gestrichelter Linie über Schlussfolgerungen.

Obj1 ist mit Obj2 über die Relation *isAlternativeTo* verbunden. Obj3 wird hinzugefügt und ist einem direkt höher liegenden Taxonomieknoten als Obj2 zugeordnet. Daher wird Obj3 über die Relation *isBroaderThan* mit Obj2 verbunden. Gleichzeitig wird auch Obj2 mit der dazu inversen Relation *isNarrowerThan* zu Obj3 verbunden. Dieser Zustand wird nun in das Ontologiemodell eingepflegt und der Schlussfolgerungsprozess gestartet. Der Reasoner erkennt, dass die Regel 30 angewendet werden kann, mit den Zuweisungen `?A = Obj1`, `?B = Obj2` und `?C = Obj3`. Die gestrichelten Linien in der Abbildung zeigen die vom Reasoner hinzugefügten Relationen an. Wieder wird gleichzeitig die inverse Relation mit hinzugefügt.
Nachdem eine korrekte Schlussfolgerung gemacht wurde, wird diese auch dem Ontologiemodell hinzugefügt und der Schlussfolgerungsprozess erneut gestartet, bis keine neue Schlussfolgerung mehr erkannt werden.
Während des Schlussfolgerungsprozesses müssen alle geschlussfolgerten Statements dahingehend überprüft werden, ob dies schon existierende Relationen sind, ob diese Relationen als *inkorrekt* gekennzeichnet sind und ob der Schlussfolgerungsweg korrekt war.

Inkorrekte Relationen

Als inkorrekt werden Relationen bezeichnet, die Fehlschlüsse in der Semantik aufweisen. Es handelt sich dabei meistens um Relationspaare, die sich gegenseitig ausschließen. Ein Beispiel wären die folgenden beiden inkorrekten Relationen in Listing 4.2.

```
1 [IncorrectRelation43: (?A lom:IncorrectRelation43 ?B) <-
2         (?A lom:IsAlternativeTo ?B) (?A lom:IsRequiredBy ?B)]
```

```
[IncorrectRelation44: (?A lom:IncorrectRelation44 ?B) <-
        (?A lom:IsAlternativeTo ?B) (?A lom:Requires ?B)]
```

Listing 4.2: Zwei Schlussfolgerungsregeln für inkorrekte Relationen

Diese beiden Statements sagen aus, dass Objekt A und Objekt B nicht gleichzeitig Alternativen und Voraussetzungen füreinander sein können. Wenn A von B vorausgesetzt wird, kann A nicht als Alternative für B eingesetzt werden.
Um zu verhindern, dass solche semantisch inkorrekten Schlüsse in die Datenbank geschrieben werden, müssen sie explizit als solche gekennzeichnet und vorher erkannt werden.

Ableitungen

Als Ableitungen werden die Schlussfolgerungswege bezeichnet, die angeben, wie eine Schlussfolgerung zustande gekommen ist. Eine Ableitung besteht aus der Regel, die zur Schlussfolgerung geführt hat und den Objekten, die diese Regel erfüllt haben.
Bei dem Beispiel, welches in Abschnitt 4.3.3 gegeben wurde, würde die Regel, die zu dieser Schlussfolgerung geführt hat, als Ableitung gespeichert werden. In diesem Fall würde die Ableitung folgendermaßen aussehen:

Rule 30: ({Obj1} lom:isAlternativeTo {Obj2}) ({Obj2} lom:isNarrowerThan {Obj3})

Gespeichert wird die Ableitung im Graphen als eine Property der Kante, die die Relation darstellt. Somit kann einerseits schnell gesehen werden, welche Relationen geschlussfolgert wurden und welche aufgrund von Metadaten erstellt wurden und zum anderen kann so schnell auf die Ableitung zugegriffen werden.
Diese Ableitungen helfen es nachzuprüfen, ob eine Schlussfolgerung korrekt geschlussfolgert wurde. Dazu ist es nötig alle Schlussfolgerungsschritte nochmal nachzuvollziehen und sicherzugehen, dass es alle Relationen, auf die die Schlussfolgerungen beruhen, auch gibt. Zum anderen spielen sie eine große Rolle, wenn es um das Löschen von Relationen geht. Wenn eine Relation gelöscht wird, die dazu führte, dass eine andere Relation geschlussfolgert wurde, muss dies festgestellt werden und entsprechend die Relation gelöscht oder zu mindestens die Ableitung mit der Relation entfernt werden. Sonst bleiben Relationen stehen, die unter Umständen auf falschen Daten beruhen. Dies wird im nächsten Abschnitt und in Kapitel 5.2.3 weiter erläutert.

Löschen von geschlussfolgerten Relationen

Geschlussfolgerte Relationen bauen auf anderen, schon existierenden Relationen auf. Wenn eine Relation gelöscht wird, entweder direkt oder indirekt durch die Löschung eines Objekts, muss daher überprüft werden, ob diese in einem Schlussfolgerungsweg verwendet wurde. Genauer muss überprüft werden, ob diese Relation in einer gespeicherten Ableitung vorkommt. Dabei gibt es zwei Möglichkeiten:

1. Ein Objekt wird gelöscht, wodurch alle Relationen gelöscht werden, die von diesem Objekt ausgehen oder die zu diesem Objekt hingehen.

2. Eine einzelne Relation wird gelöscht. Dies kann nur auftreten, wenn alle Ableitungen der Relation gelöscht wurden.

Wenn ein Objekt gelöscht wird und alle direkt damit zusammenhängenden Relationen, so werden alle Ableitungsregeln der Relationen gelöscht, in der die ID des Löschobjekts vorkommt. Wenn eine einzelne Relation gelöscht wird, so werden alle Ableitungen gelöscht, in der diese Relation mit dem genauen Ziel- und Quellobjekt vorkommen.

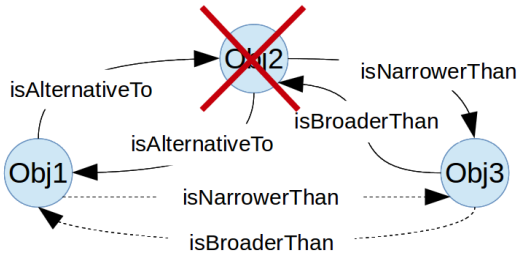

Abbildung 4.8: Obj2 wird gelöscht und damit auch alle angrenzenden Relationen und alle von Obj2 abhängigen, geschlussfolgerten Relationen

Abbildung 4.8 zeigt das Beispiel aus Sektion 4.3.3, in dem nun Obj2 gelöscht werden soll. Alle direkt angrenzenden Relationen von Obj2 werden gelöscht. Darüber hinaus werden die Ableitungen von allen Relationen durchsucht, ob in einer Ableitung Obj2 vorkommt. In diesem Fall würde die Kante `Obj1 isNarrowerThan Obj3` diese Ableitung gespeichert haben:

Rule 30: ({Obj1} lom:isAlternativeTo {**Obj2**}) ({**Obj2**} lom:isNarrowerThan {Obj3})

Da Obj2 in der Ableitung referenziert wird, würden diese Regeln aus der Ableitung gelöscht werden. Bleiben keine Ableitungen mehr übrig, so wird die komplette Relation und deren Inversion gelöscht. Der Hintergrund ist, dass solange die Ableitung auf

existierenden Relationen aufbaut, diese legitim ist und beibehalten werden kann. Sobald aber alle Relationen gelöscht sind, welche zur Schlussfolgerung der Relation geführt haben, muss angenommen werden, dass diese Relation auf falschen Informationen aufgebaut und nicht mehr legitim ist.
Als Alternative könnten die Ableitungen, die das gelöschte Objekt erzeugt hat, auch beibehalten werden. Es wird also gemäß dem Grundsatz gearbeitet, dass ein gelöschtes Objekt nicht unbedingt als falsch angesehen werden muss. Als Beispiel sei angebracht, dass ein User eine neue, bessere Version eines Objekts erstellen will. Nun werden alle Relationen und Ableitungen gelöscht, obwohl das Objekt nicht per se falsch war. Dies zu erkennen ist jedoch nicht einfach und die Ressourcen, die für das aufwendige Löschen der Relationen benutzt werden, werden bei der langsamen Arbeitsweise des Reasoners keinen großen Einfluss haben.
Darüber hinaus sollten die Schlussfolgerungsregeln dafür sorgen, wenn eine neue Version eines Objekts erstellt wird, dass am Ende die gleichen Relationen und die Ableitungen, die gelöscht worden sind, wieder erstellt werden.

4.3.4 Verknüpfungsmodul

Das Verknüpfungsmodul ist ein separat laufender Prozess, der benachrichtigt wird, wenn Lernobjekte modifiziert worden sind. Das Modul ist aus vier Sub-Prozessen aufgebaut, die für verschiedene Relationen verantwortlich sind. Der erste Prozess überprüft Struktur- und Abhängigkeitsbeziehungen. Der zweite Prozess ist zuständig für Beziehungen innerhalb einer Taxonomie. Der dritte Prozess überprüft anhand von vielen verschiedenen Metadaten inhaltliche Beziehungen. Der letzte Prozess ist eine Schlussfolgerungsengine, die aus den davor erstellten und schon bestehenden Relationen auf neue Relationen schließt.
Der Ablauf des Verknüpfungsprozesses wird als Aktivitätsdiagramm in Abbildung 4.9 illustiert.

Die ersten beiden Sub-Prozesse werden dabei einmalig ausgeführt, da die möglichen Verknüpfungen eindeutig aus den dazugehörigen Metadaten und Klassifikationen extrahiert werden können. Die letzten beiden Sub-Prozesse dagegen hängen von schon bestehenden Relationen ab. Die Inhaltsbeziehungen überprüfen unter anderem, ob zwei Objekte die gleichen Relationen aufweisen, um somit auf inhaltliche Äquivalenz zu schließen. Schlussfolgerungen werden auch mit schon bestehenden Relationen erstellt.
Daher müssen diese beiden Sub-Prozesse mehrfach ausgeführt werden, da bei einem Durchlauf neue Relationen erstellt werden könnten, die wiederum auf weitere Relationen

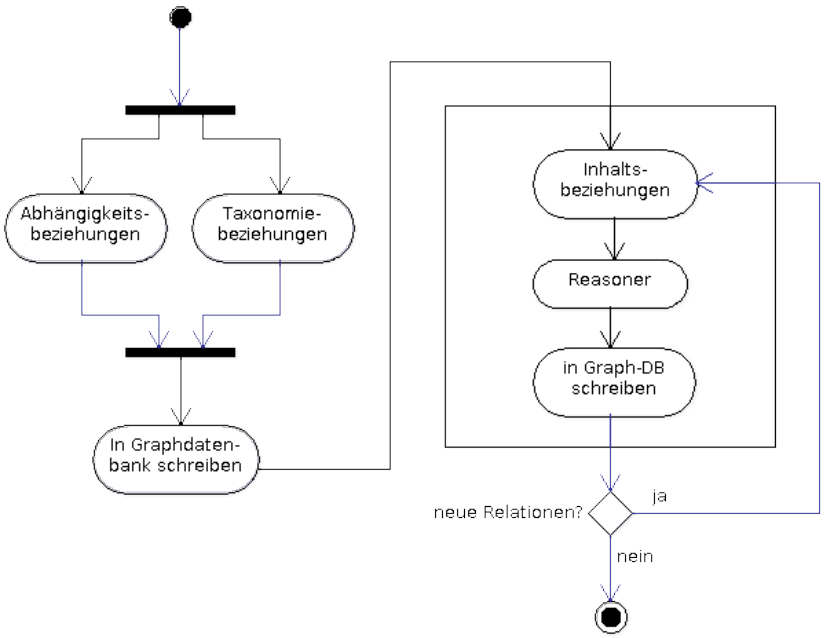

Abbildung 4.9: Ein Aktivitätsdiagramm der Verknüpfungsengine. Der Prozess startet mit der Überprüfung der Abhängigkeits- und Taxonomiebeziehungen. Danach werden solange die Inhalts- und geschlussfolgerten Beziehungen überprüft und die die Graphdatenbank geschrieben, bis keine neuen Relationen mehr gefunden worden sind.

schließen lassen. Dies muss solange gemacht werden, bis keine neuen Relationen bei einem Durchlauf erstellt wurden. Dann kann davon ausgegangen werden, dass alle möglichen Relationen basierend auf dem neu hinzugefügten Objekt erstellt wurden.

4.4 Systemgraph

Das System ist aus vielen Komponenten aufgebaut, darunter Tags, Topics, Lernende, Lerngruppen und Lernobjekte, die miteinander in Beziehung stehen. Diese Beziehungen können Aufschluss geben über das Lernverhalten und Lernvorlieben der Anwender und eignen sich gut als eine Quelle für Empfehlungsalgorithmen. Um diese Beziehungen schnell zu überprüfen und visuell verständlich darstellen zu können, empfiehlt es sich

diese Komponenten mit deren Beziehungen in einer Graphdatenbank zu speichern. Die einzelnen Komponenten sind die Knoten und die Beziehungen die Kanten. Abbildung 4.10 illustriert dieses Konzept.

Dieses Prinzip ist eng verwandt mit dem Konzept von **Personal Learning Networks**.

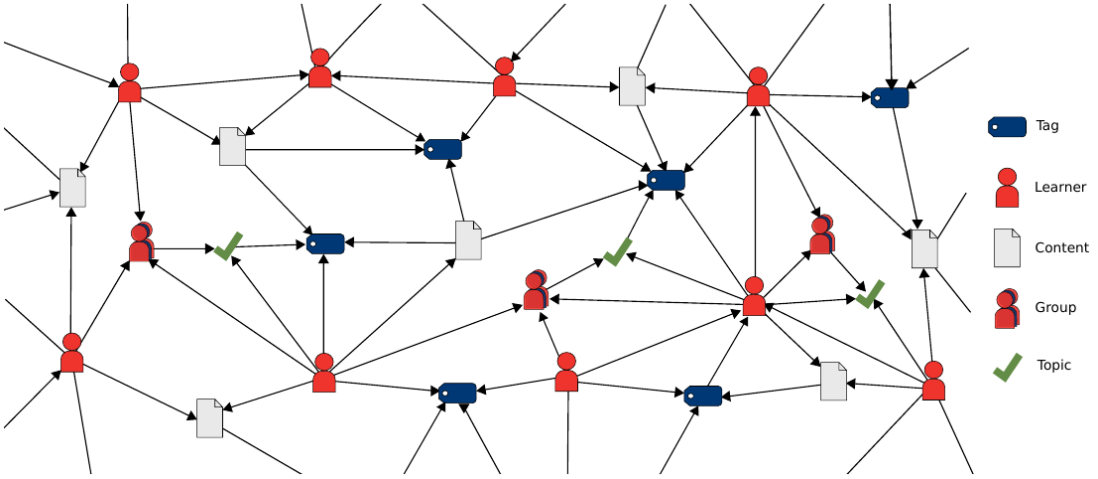

Abbildung 4.10: Eine Illustration des Systemgraphen. User, Tags, Topics, Groups und Content sind über Kanten miteinander verbunden. (Brauer u. a., 2013)

Couros (2010) definierte Personal Learning Networks als die Summe aller Beziehungen, welche in der Entwicklung und Förderung eines Personal Learning Environments resultiert. Zu den sozialen Beziehungen werden in diesem Graph auch die semantischen Verknüpfungen der Lernobjekte mit einfließen. Aus diesem reichhaltigen Netz an Beziehungen können einerseits viele implizite Verbindungen herausgelesen werden, beispielsweise dass zwei Nutzer an denselben Inhalten arbeiten, aber keine sozialen Bindungen zueinander haben. Daraus können Vorschläge für die Zusammenarbeit und Gruppenbildung entnommen werden. Oder an den semantischen Beziehungen der Lernobjekte und den Ähnlichkeiten der Beziehungen von mehreren Benutzern können Inhaltsvorschläge für andere Nutzer gemacht werden.

Die verschiedenen Knotentypen sind mittels gerichteter Kanten verbunden. Durch die Verbindung verschiedener Knotentypen entstehen andere implizite Verbindungen. So ist es möglich, Lernobjekte und Lerngruppenmitglieder zu finden, die keine direkte Verbindung über andere User oder Lernobjekte haben. Es ist außerdem möglich, die Stärke der Verbindung zwischen zwei Knoten über die Anzahl der gemeinsamen Nachbarn oder kanten-disjunkten Wege zu messen. Der User steht im Zentrum des Graphen. Er wird

durch sein Profil im Netzwerk repräsentiert. Er übernimmt die aktive Rolle im Netzwerk und erstellt oder bearbeitet andere Knoten und erstellt neue Verbindungen zwischen ihnen.
Die Kanten zwischen Benutzern und Lernobjekten entstehen durch das Konsumieren oder Editieren. Ihre kontextuellen Relationen werden über die gemeinsamen Verbindungen zu Tag-Knoten dargestellt. Um auch die Relevanz zwischen den verschiedenen Knotentypen zu modellieren, hat jede Verbindungsart ein spezifisches Gewicht. Diese ermöglichen beim Traversieren des Graphen Knoten auf Grund ihrer Relevanz als nächsten Knoten zu selektieren.
Folgend sind die möglichen Beziehungen in Bezug auf die Lernobjekte aufgelistet.

Lerner → Content Es gibt drei Möglichkeiten, wie eine Kante zwischen einem Lerner und dem Content entstehen kann. Erstens: Der Lerner erstellt oder bearbeitet den Content. Zweiten: Ein Benutzer schaut sich den Lerninhalt an, dabei kann als Kantengewicht angegeben werden, wie oft sich der Benutzer den Lerninhalt angesehen hat. Und Drittens: Der Lerner fügt den Content zu seinen Favoriten hinzu.

Thema → Content Diese Kante entsteht, wenn ein Lernobjekt zu einem Thema hinzugefügt wurde.

Tag → Content Die Kante wird erstellt, wenn ein Lernobjekt „getaggt“ wird. Schlüsselwörter eines Lernobjekts werden automatisch als Tag erstellt.

Content → Content Die Verbindung von Lernobjekt zu Lernobjekt wurde in Abschnitt 4.3.2 besprochen.

Die weiteren Beziehungen zwischen Gruppen, Themen und Benutzern ist Bestandteil der Arbeit von Brauer u. a. (2013) und werden dort ausführlich beschrieben.

5 Implementierung des semantischen Content-Netzes

In diesem Kapitel werden die Anforderungen und der konzeptionelle Entwurf aus dem vorigen Kapitel umgesetzt. Dabei geht es nicht nur um den Bau des semantischen Content-Netzes, sondern auch um die Einbindung des Content-Netzes in Diaspora. Dafür wurden zwei Module verwendet. Erstens das Verknüpfungsmodul, welches das semantische Content-Netz aufbaut und die Lernobjektverwaltung, die das semantische Content-Netz und Diaspora miteinander verbindet.
Um den Aufbau der Module einordnen zu können, wird in Abbildung 5.1 die Architektur des Gesamtsystems skizziert.

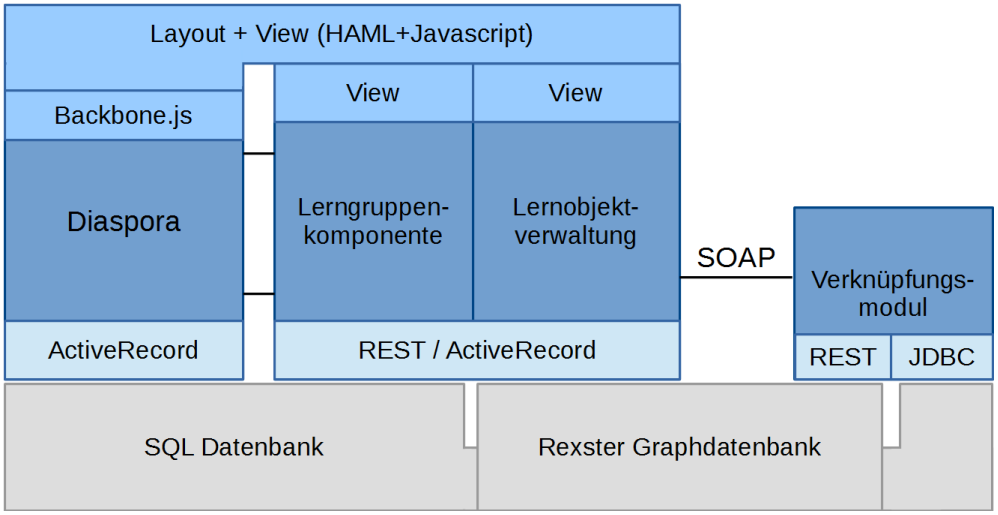

Abbildung 5.1: Darstellung der Architektur des gesamten Systems als Schichtenmodell.

Diaspora ist die Hauptapplikation und stellt das Layout und die ActiveRecord-Schnittstelle zur Verfügung. Die Lerngruppenkomponente und die Lernobjektverwaltung sind die beiden E-Learning Komponenten, die an die Hauptapplikation angeschlossen werden. Über eine SOAP-Schnittstelle kann die Lernobjektverwaltung mit dem Ver-

knüpfungsmodul kommunizieren. Es werden zwei Datenbanken benutzt: Eine relationale und eine Graphdatenbank. Die Graphdatenbank wird über eine REST-Schnittstelle angesprochen. Die SQL-Datenbank wird von den E-Learning-Komponenten über die ActiveRecord-Schnittstelle erreicht. Das Verknüpfungsmodul erreicht die Datenbank über die Java-Schnittstelle JDBC.
Die Implementierung des semantischen Content-Netzes soll die Anforderungen und Spezifikationen, die im vorigen Kapitel gestellt wurden, erfüllen. Diese sind Erstens: Eine modulare Implementierung der Module und eine lose Kopplung zu dem langsam arbeitenden Verknüpfungsmodul, um die Frontend-Anwendung nicht zu beeinträchtigen. Zweitens: Die korrekte Funktionsweise des Verknüpfungsmoduls und die Korrektheit der erstellten Relationen. Und Drittens: Erhaltung der Konsistenz innerhalb des semantischen Content-Netzes durch das korrekte Löschen von geschlussfolgerten Relationen.

5.1 Lernobjektverwaltung

Die Lernobjektverwaltung ist für die Erstellung und Verwaltung der Lernobjekte verantwortlich. Einerseits bietet sie den Anwendern eine Schnittstelle zu den Lernobjekten. Über diese Schnittstelle kann der Anwender Lernobjekte erstellen, sie bearbeiten und sie zur Löschung beantragen. Andererseits bildet es die Schnittstelle zum Verknüpfungsmodul. Es verbindet das System um Diaspora mit dem semantischen Content-Netz.

5.1.1 Modulare Anbindung

Die Lernobjektverwaltung wurde als Rails-Engine[1] konzipiert, um mit Diaspora verbunden zu werden. Rails-Engines sind weitgehend eigenständige Applikationen, die an eine Host-Anwendung „angehängt“ werden können, um diese mit mehr Funktionalität zu versorgen. Sie haben ihren eigenen Namensbereich, können aber auf alle Funktionen und Variablen der Host-Applikation zugreifen. In ihrem Umfang und ihrer Eigenständigkeit sind Engines zwischen vollwertigen Applikationen und Plugins anzusiedeln. Die Nutzung von Engines ermöglicht es, die vorzunehmenden Erweiterungen möglichst modular zu gestalten und hat den Vorteil, dass der bestehende Diaspora-Code nur minimal verändert werden muss. Dabei bleibt gleichzeitig ein einfacher Zugriff auf die Funktionen und das Layout des Hauptprogramms. Vor allem da Diaspora ein bereits komplettes, getestetes System ist, kann es so vermieden werden das funktionierende System zu modifizieren. Damit ist es möglich, über wohldefinierte Schnittstellen mit der Hauptapplikation zu

[1] http://guides.rubyonrails.org/engines.html

kommunizieren und so die Kernapplikation fehlerfrei und lauffähig zu halten. Darüber hinaus gibt es den Entwicklern die Möglichkeit unabhängig an dem gleichen System zu arbeiten und die anderen Module nicht in ihrer Funktionalität zu beeinflussen.

5.1.2 Lernobjekte

Aufbau

Die Klasse *LearningObject* hält alle Informationen über das Lernobjekt. Dabei sind die Attribute des Lernobjekts aus dem LOM-Metadatenschema übernommen. Abhängigkeiten zwischen Lernobjekten werden intern mit den Klassen *Structure* (strukturelle Abhängigkeit) und *Requirement* (inhaltliche Abhängigkeit) dargestellt. Die zum Lernobjekt gehörigen Schlüsselwörter und Taxonomieknoten werden mit den Klassen *Keywords* und *Taxons* festgehalten. Alle Klassen erben von der Oberklasse *ActiveRecord*. Eine Klasse, die im Hintergrund die Synchronisation mit einer relationalen Datenbank übernimmt. Zur Übersicht zeigt Abbildung 5.2 das Klassendiagramm um die Klasse *LearningObject*.

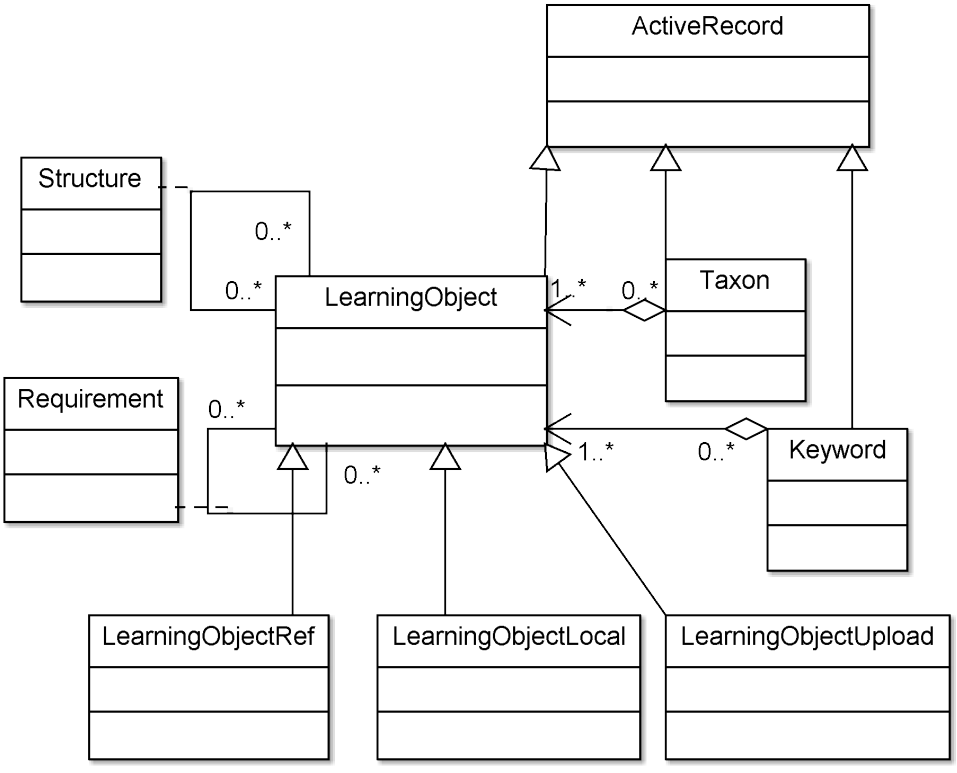

Abbildung 5.2: Klassendiagramm der Modelle der Lernobjektverwaltung

Wie die Verbindung zwischen den Klassen hergestellt wird, zeigt Listing 5.1 an der Klasse *Structure*. Diese definiert eine *belongs_to* Verknüpfung zu den beiden in Beziehung stehenden Objekten. Einerseits das Kindobjekt und andererseits das Elternobjekt. In *LearningObject* wird durch das Schlüsselwort *has_many* eine 1:n Beziehung zu den Kindobjekten durch die Klasse *Structure* aufgebaut.

```
1  class LearningObject < ActiveRecord::Base
2    ...
3    has_many :structures
4    has_many :parents, :through => :structures, dependent: :destroy
5    ...
6  end
7  class Structure < ActiveRecord::Base
8    belongs_to :learning_object
9    belongs_to :parent, :class_name => "LearningObject",
10             :foreign_key => "parent_id"
11 end
```

Listing 5.1: Die *Structure*-Klasse, um strukturelle Abhängigkeit abzubilden.

Die Entscheidung, diese Attribute des Lernobjekts in eigene Klassen auszugliedern, entspringt dem Design von ActiveRecord und dem konventionsorientierten Prinzip von Ruby on Rails, welche den Aufbau der Klasse direkt in das korrespondierende Datenbankschema abbildet. Somit wird direkt eine eigene Tabelle erstellt und verwaltet, die zum Beispiel die ID eines Elternobjekts mit der ID des Kindobjekts in Beziehung setzt. Die Klasse *LearningObject* teilt sich in drei Unterklassen auf, relativ zur Quelle des Contents. Lernobjekte, die eine Ressource aus dem Internet referenzieren, gehören der Klasse *LearningObjectRef* an. Hier wird nur die URL als Content gespeichert und mit den Metadaten wird die Ressource hinter der URI beschrieben. Hochgeladener Content, in Form von Mediendaten, wird mit der Klasse *LearningObjectUpload* gespeichert. Die Daten werden dabei serverseitig im Repository gespeichert. Als letztes können eigene Artikel erstellt werden über den Markup-Editor *CKEditor*[2]. Auch diese Artikel werden im serverseitigen Repository gespeichert. Repräsentiert werden diese Lernobjekte durch die Klasse *LearningObjectLocal*. Die Artikel können gut als Beschreibung von anderen Lernobjekten verwendet werden.
Die Unterklassen erweitern die Oberklasse um bestimmte Funktionen, die zum Beispiel für die Speicherung des jeweiligen Contents von Bedeutung sind. So fängt die Klasse *LearningObjectLocal* einen *before_save* Callback der Oberklasse auf und ruft eine Me-

[2] http://ckeditor.com/

thode auf, die den Inhalt im Repository speichert, bevor das Objekt in die Datenbank geschrieben wird.

Löschen von Lernobjekten

Nur der Benutzer mit der Verfügungsgewalt über das Objekt, kann dies löschen. Wenn der Nutzer den Löschauftrag gegeben hat, wird zuerst die Löschmarkierung in der Datenbank gesetzt. Damit ist dieses Objekt für den Benutzer gelöscht, kann aber für offenstehende Themen noch benutzt werden. Danach wird die *checkDeletion*-Methode in der Klasse *LearningObject* aufgerufen, die über den WebService die *delete*-Methode des Verknüpfungsmoduls aufruft, gezeigt in 5.2. Dort wird überprüft, ob noch eine Verbindung zu einem Thema besteht. Wenn ja, wird die Löschmarkierung in der Graphdatenbank im korrespondierenden Knoten gesetzt, wenn nicht, wird das Objekt mit allen verbliebenen Verknüpfungen gelöscht (siehe Abschnitt 5.2.3).

```
1  public boolean delete(int id){
2    System.out.print("Deleting "+id+"... ");
3    boolean ret = false;
4    boolean topic = _db.checkTopicRelations(id);
5    if(topic){
6      _db.setDeleteFlag(id);
7      System.out.println("still attached to topic. Flag has been set.");
8    } else {
9      ret = _db.removeLearningObject(id);
10     System.out.println(ret ? "Done." : "Aborted.");
11   }
12   return ret;
13 }
```

Listing 5.2: Der Löschvorgang im Verknüpfungsmodul

Durch die Markierung kann das Objekt für den Benutzer als veraltet angezeigt werden und ist eine Aufforderung das Objekt zu entfernen oder ersetzen. Durch das explizite Entfernen eines Objekts oder die Beendigung des Themas kann die Verbindung von einem Thema zu einem Objekt getrennt werden. In dem Fall wird die Löschmethode der entsprechenden Objekte erneut aufgerufen. Wenn das Objekt zum Löschen markiert ist, wird wiederum eine Nachricht an das Verknüpfungsmodul gesendet. Wenn das Objekt erfolgreich vom Verknüpfungsmodul gelöscht wurde, wird der Rückgabewert *true* zurückgegeben und die *destroy*-Methode des Objekts, welches den Aufruf an den WebService startete, ausgeführt. Dabei wird das Objekt vollständig aus der relationalen Datenbank gelöscht.

5.1.3 Schnittstelle zum Verknüpfungsmodul

Die Kommunikation zwischen der Lernobjektverwaltung und dem Verknüpfungsmodul findet über eine SOAP-Schnittstelle statt. Darüber hinaus übernimmt ein eigener Thread die Kommunikation mit dem Verknüpfungsmodul. Dies hat den Vorteil, dass das langsam arbeitende Verknüpfungsmodul somit den zeitkritischen Frontend-Teil nicht bremst. Alle Anfragen der Lernobjektverwaltung werden in einer *Queue* gespeichert und werden von dem SOAP-Client abgearbeitet. Die Anfrage besteht aus der ID des Lernobjekts und der Art des Befehls (Read, Create, Update, Delete). Der SOAP-Client schickt entsprechend eine Anfrage an den WebService, wo das Verknüpfungsmodul läuft und wartet auf eine Antwort. Wenn eine Antwort zurück kommt und der Vorgang erfolgreich war, wird eine entsprechende Meldung an die Lernobjektverwaltung gesendet. Wenn der Vorgang nicht erfolgreich war oder das Timeout der Anfrage des SOAP-Client überschritten wurde, wird die Anfrage verworfen und der nächste Befehl abgearbeitet. Ausgehend von der Lernobjektverwaltung muss der entsprechende Befehl noch einmal gesendet werden. Abbildung 5.3 zeigt den Kommunikationsverlauf der einzelnen Prozesse. Der Rails-Server ist dabei der Prozess, in dem die Lernobjektverwaltung läuft, der SOAP-Server ist der Prozess, in dem das Verknüpfungsmodul läuft. In diesem Beispiel sollen die Relationen eines neuen

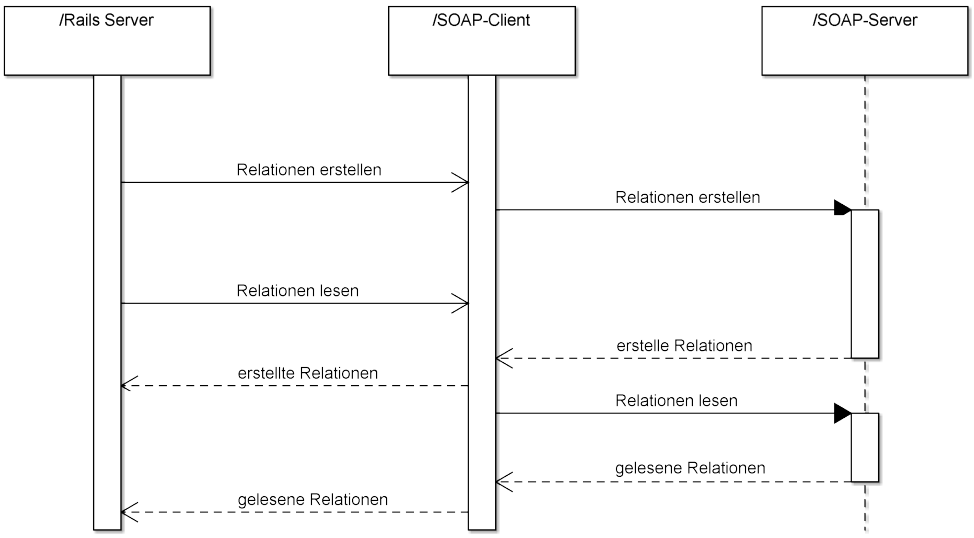

Abbildung 5.3: Die Kommunikation der Lernobjektverwaltung (Rails-Server) mit dem Verknüpfungsmodul (SOAP-Server) über den SOAP-Client als Sequenzdiagramm.

Objekts erstellt werden. Die Lernobjektverwaltung schickt den Befehl *CREATE* mit dem Objekt als Argument. Dieser Befehl wird in der Queue gespeichert. Der SOAP-Client nimmt den Befehl aus der Queue und ruft über den WebService die entsprechende Methode des SOAP-Servers auf. Das Verknüpfungmodul wird gestartet und die erstellten Relationen zurück geliefert. Alle Befehle, die in der Zwischenzeit beim SOAP-Client ankommen, werden in die Queue gelegt. Nach der Antwort des SOAP-Servers wird vom SOAP-Client die *reasonDone*-Methode des entsprechenden Objekts aufgerufen und der nächste Befehl bearbeitet.

5.2 Verknüpfungsmodul

Das Verknüpfungsmodul verknüpft die Lernobjekte über Relationen. Die verwendeten Relationen wurden in Kapitel 4.3.2 vorgestellt. Dabei wurden die Relationen in drei Unterkategorien aufgeteilt: **Abhängigkeits-, Taxonomie-** und **Inhaltsbeziehungen**. Diese Aufteilung wird auch in der Implementierung beibehalten. Für jede der drei Kategorien ist eine Klasse verantwortlich.
Das Verknüpfungsmodul wird als WebService initialisiert. Der Webservice bietet drei Methoden an: Zum Erstellen, Auslesen und Löschen von Relationen. Der WebService leitet die Anfragen an die Implementierung weiter mit der ID des zu bearbeitenden Lernobjekts als Argument. Die ID wird in einer *Identifier*-Klasse gewrappt, die String-IDs oder Integer-IDs speichern kann.

Die Implementierung des Ontological Evaluation Layers in hylOs wurde als Startpunkt für die Implementierung des Verknüpfungsmoduls genommen. Davon ausgehend wurden jedoch viele Erweiterungen und Veränderungen gemacht. So wurde der komplette Verknüpfungsprozess umgestaltet, indem das Inhaltsbeziehungs- und Schlussfolgerungsmodul mehrmals durchlaufen wird, gezeigt in Kapitel 4.3.4. Vor allem der Schlussfolgerungsprozes wurde umgestaltet und effizienter gestaltet, indem nur noch die Knoten in einer bestimmten Entfernung zur Schlussfolgerung herangezogen werden. Auch werden die Ableitungen nun aktiv genutzt und zum korrekten Löschen der Relationen genutzt. Darüber hinaus werden die Taxonmiebeziehungen besser erkannt durch eine rekursives Auf- und Absteigen des Taxonomiepfads. Weitgehend gleich geblieben ist die Erkennung von Inhalts- und Abhängigkeitsbeziehungen über die Metadaten der Lernobjekte.

5.2.1 Ontologie

Wie in Kapitel 4.3.2 erwähnt wurde die Ontologie von Lange (2006) übernommen. Die Ontologie besteht aus der Klasse *LearningObject* und Klassen für die Eigenschaften der Verknüpfungen, wie symmetrisch, transitiv und invers. Die Verknüpfungen werden mit diesen Klassen definiert. Als Beispiel dient in Listing 5.3 die Verknüpfungen *isPartOf* und *hasPart*.

```
<owl:ObjectProperty rdf:ID="IsPartOf">
  <rdf:type rdf:resource="#TransitiveProperty"/>
  <inverseOf rdf:resource="#HasPart"/>
  <rdfs:range rdf:resource="#LearningObject"/>
  <rdfs:domain rdf:resource="#LearningObject"/>
  <rdfs:comment rdf:datatype="&xsd;string">
    LOM.Relations.kind isPartOf
  </rdfs:comment>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="HasPart">
  <rdf:type rdf:resource="#TransitiveProperty"/>
  <inverseOf rdf:resource="#IsPartOf" />
  <rdfs:range rdf:resource="#LearningObject"/>
  <rdfs:domain rdf:resource="#LearningObject"/>
  <rdfs:comment rdf:datatype="&xsd;string">
    LOM.Relations.kind hasPart
  </rdfs:comment>
</owl:ObjectProperty>
```

Listing 5.3: Die Definitionen der Verknüpfungen *isPartOf* und *hasPart* in der die Ontologie beschreibenden OWL-Datei

Im Fall von IsPartOf wird angegeben, dass sie transitiv ist, eine inverse Verknüpfung von hasPart und von einem Lernobjekt zu einem anderen Lernobjekt zeigt. Der RDF-Graph dieser Ontologie würde die Klasse *LearningObject* mit den Verknüpfungen als selbstreferenzierende Beziehung zeigen.

5.2.2 Relationen erstellen

Wenn Relationen für ein Lernobjekt erstellt werden sollen, wird die Methode *create(int id)* aus der Klasse *LinkingEngine* aufgerufen. Die Methode holt das Lernobjekt mit der ID *id* mit allen Attributen aus der SQL-Datenbank. Danach wird die *reason()*-Methode mit dem

Lernobjekt als Argument aufgerufen, die das Lernobjekt allen vier Relationsfinderklassen übergibt, die die Beziehungen erstellen und in die Graphdatenbank speichern. Der Aufbau der *reason()*-Methode folgt direkt dem Prozessablaufgraph, welcher in Kapitel 4.3.4 beschrieben wurde. Die komplette Methode wird in Listing 5.4 gezeigt.

```
1  public List<Relation> reason(LearningObject elo){
2    Set<Relation> relations = new HashSet<Relation>();
3    if(elo != null){
4      //Zuerst werden die Abhängigkeitsbeziehungen und die
5      //Taxonomiebeziehungen überprüft
6      relations.addAll(_struct.start(elo));
7      relations.addAll(_tax.start(elo));
8
9      List<Relation> new_relations = new ArrayList<Relation>();
10     new_relations.addAll(_heuristic.start(elo));
11     new_relations.addAll(_jena.start(elo.getIdentifier()));
12
13     //Solange es neue Relationen gibt, werden die Inhaltsbeziehungen
14     //und die Schlussfolgerungen überprüft
15     while(!relations.containsAll(new_relations)){
16       relations.addAll(new_relations);
17       new_relations = new ArrayList<Relation>();
18
19       new_relations.addAll(_heuristic.start(elo));
20       new_relations.addAll(_jena.start(elo.getIdentifier()));
21     }
22
23     relations.addAll(new_relations);
24   }
25   return new ArrayList<Relation>(relations); //return als Liste
26 }
```

Listing 5.4: Die *reason()*-Methode in der Klasse *LinkingEngine*

Nachdem überprüft wurde, dass das Lernobjekt nicht null ist, werden die Abhängigkeitsbeziehungen und die Taxonomiebeziehungen überprüft (Zeile 10,11). Danach werden die Inhaltsbeziehungen und die Schlussfolgerungen erstellt. Diese Beziehungen werden mehrmals in einer Schleife überprüft, da beide Prozesse die schon bestehenden Relationen überprüfen und daher bei neu hinzugefügten Relationen, diese wieder neu mit einlesen müssen. Die erstellten Relationen werden als Rückgabewert geliefert und in der aufrufenden Methode wird ein Flag in der Graphdatenbank gesetzt, dass das Objekt erfolgreich verknüpft wurde.

StructureReasoner

Der *StructureReasoner* überprüft die Beziehungen *isPartOf/HasPart* und *requires/isRequiredBy* des Lernobjekts. Dazu werden die ParentIDs und die RequirementIDs des Lernobjekts verwendet. Die Methode zur Erstellung der Eltern-/Kindbeziehungen ist in Listing 5.5 gezeigt.

```
1  private List<Relation> checkIsParentOf(LearningObject elo) {
2    List<Relation> new_relations = new ArrayList<Relation>();
3    //alle Objekte mit parent_id -> id
4    for (Identifier id : elo.getParents()){
5      if((long)id.getId() > 0 && (long)id.getId() != elo.getID()){
6        new_relations.add(new Relation(id, Relation.HAS_PART,
7                                       elo.getIdentifier()));
8        new_relations.add(new Relation(elo.getIdentifier(),
9                                       Relation.IS_PART_OF, id));
10     }
11   }
12   //alle Objekte mit id -> parent_id
13   for (Identifier id : elo.getChildren()){
14     if((long)id.getId() > 0 && (long)id.getId() != elo.getID()){
15       new_relations.add(new Relation(elo.getIdentifier(),
16                                      Relation.HAS_PART, id));
17       new_relations.add(new Relation(id, Relation.IS_PART_OF,
18                                      elo.getIdentifier()));
19     }
20   }
21   return new_relations;
22 }
```

Listing 5.5: *checkisParentOf(LearningObject elo)* in Klasse *StructureReasoner.java*

Zur Überprüfung dieser Beziehung werden die Parent-IDs aus der relationalen Datenbank gelesen. Wenn die ID des Lernobjekts eine Parent-ID ist, wird eine *HasPart*-Beziehung und die Inverse von dem Lernobjekt zum anderen Objekt gebildet. Wenn die ID des Lernobjekts eine Parent-ID hat, wird eine *IsPartOf*-Beziehung von dem anderen Objekt zum Lernobjekt mit der Inversen erstellt.
Die *Requirement*-Beziehungen werden nach dem gleichen Muster erstellt.

TaxonomyReasoner

Die Klasse *TaxonomyReasoner* überprüft die Taxonomiebeziehungen des Lernobjekts. Um die taxononmischen Beziehungen des Lernobjekts zu analysieren muss der Taxonomie-

baum eingelesen werden. Dafür wird das JENA-Framework benutzt. Der Taxonomiebaum ist mit SKOS beschrieben. Als Testtaxonomie wird das ACM Klassifizierungssystem von 1998 verwendet[3].
Die SKOS-Datei wird über ein *InputStream* geöffnet und von einem JENA-Model eingelesen.

```
InputStream in = FileManager.get().open(path_to_skos_file);
_model = ModelFactory.createDefaultModel();
_model.read(in, null);
```

Listing 5.6: Der Taxonomiebaum wird eingelesen

Nachdem das JENA-Modell mit dem Taxonomiebaum erstellt wurde, wird der Verknüpfungsprozess für die Taxonomiebeziehungen gestartet. Dabei gibt es die **IsBroaderThan-**/**IsNarrowerThan-** und die **references-**/**isReferencedBy-**Beziehungen.
Für die Abfragen werden SPARQL-Queries benutzt. Listing 5.7 zeigt die Abfrage, um alle Knoten zu bekommen, die unterhalb eines bestimmten Taxonomieknotens sind.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?each WHERE { ?each skos:narrower <taxon> }
```

Listing 5.7: SPARQL-Abfrage der unterhalb liegenden Taxonomieknoten

In diesem Codeabschnitt ist *<taxon>* ein Taxonomieknoten von dem zu überprüfenden Lernobjekt. Ein Problem ist jedoch, dass diese Abfrage nur die Knoten ausgibt, die genau eine Kante von dem angegebenen Taxonomieknoten entfernt sind. Dabei würde die Beziehung *IsNarrowerThan*/*isBroaderThan* auch bestehen, wenn die beiden Knoten über zwei Kanten hinweg verbunden sind. Daher musste ein Weg gefunden werden, wie die Taxonomiebeziehung über mehrere Kanten weg erkannt werden können. Eine Möglichkeit ist, dieses mit dem Reasoner durch Schlussfolgerungsregeln zu erledigen in Form der Schlussregel:

```
[Taxonomy1: (?A lom:IsNarrowerThan ?B) <- (?A lom:IsNarrowerThan ?B)
                        (?B lom:IsNarrowerThan ?C) notEqual(?A, ?B)]
```

Listing 5.8: Beispiel einer Schlussregel für die Erkennung von Taxonomiebeziehungen

Das Problem ist jedoch, dass JENA nicht die Taxonomie überprüft, sondern nur die Beziehungen der Lernobjekte. Das heißt, wenn es keine Lernobjekte gibt, die diesen Taxonomieknoten angehören, so kann auch keine Verbindung hergestellt werden. Abbildung 5.4 illustriert das Problem.

[3] http://www.acm.org/about/class/1998

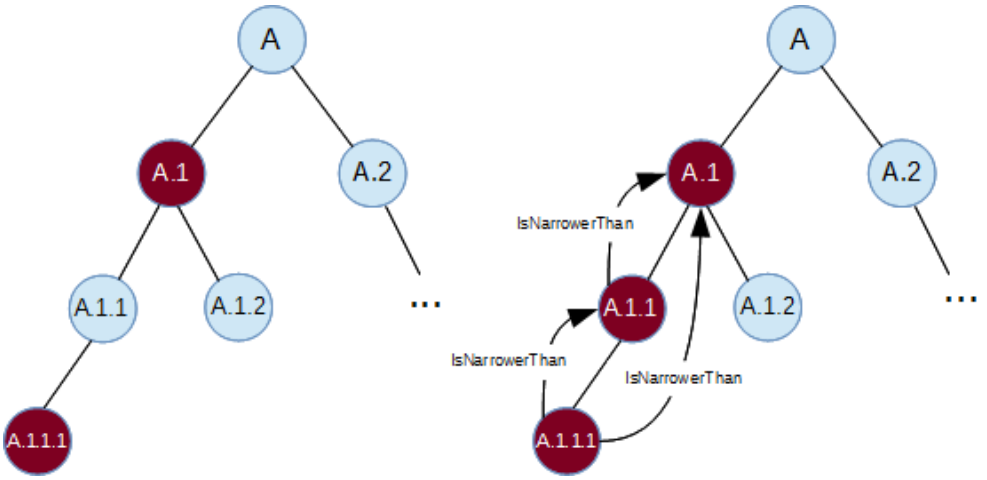

Abbildung 5.4: Darstellung einer Beispieltaxonomie. Für die hellblauen Knoten gibt es keine Lernobjekte, die diesen Taxonomieknoten zugewiesen wurden. Für die dunkelroten Knoten gibt es Lernobjekte. Erst wenn der Knoten A.1.1 einem Lernobjekt zugewiesen wurde, könnte JENA die Verbindung *IsNarrowerThan* herstellen.

Die zweite und auch implementierte Idee ist, die Taxonomie aufwärts und abwärts vom gegebenden Taxonomieknoten aus zu überprüfen, ob es auf dem Weg ein Lernobjekt gibt, der diesen Taxonomieknoten zugeordnet ist. Dies kann im TaxonmyReasoner rekursiv erledigt werden. Dazu wird einer Methode der Querystring, der Startknoten und die Richtung der Abfrage übergeben. Listing 5.9 zeigt den Code für diese Methode.

```
1  private List<Relation> recursiveQuery(String queryString,
2                                        String tax, boolean broader){
3    //Query wird ausgeführt
4    List<Relation> relations = new ArrayList<Relation>();
5    Query query = QueryFactory.create(queryString);
6    QueryExecution qe = QueryExecutionFactory.create(query, getModel());
7    ResultSet results = qe.execSelect();
8
9    while(results.hasNext()){
10     String subjectTax = results.next().get("each").toString();
11     List<Identifier> ids = _db.getIdFromTaxon(subjectTax);
12     for(Identifier id : ids){
13       if(broader){
14         relations.add(new Relation(_elo.getIdentifier(),
15                       Relation.IS_BROADER_THAN, id));
16         relations.add(new Relation(id, Relation.IS_NARROWER_THAN,
```

```
17                                      _elo.getIdentifier()));
18          } else {
19            relations.add(new Relation(_elo.getIdentifier(),
20                          Relation.IS_NARROWER_THAN, id));
21            relations.add(new Relation(id, Relation.IS_BROADER_THAN,
22                          _elo.getIdentifier()));
23          }
24        }
25        //Solange es results gibt, wird die Methode rekursiv aufgerufen
26        //und ein Knoten höher oder tiefer überprüft
27        relations.addAll(recursiveQuery(queryString.replace(tax, subjectTax),
28                                        subjectTax, broader));
29      }
30      qe.close();
31      return relations;
32    }
```

Listing 5.9: Eine rekursive Taxonomieüberprüfung

Der SPARQL-Querystring, der übergeben wird, ist in Listing 5.7 angegeben. In der while-Schleife werden alle Knoten zurückgegeben die unterhalb von *tax* liegen oder überhalb. Danach werden die IDs aller Lernobjekte aus der Datenbank geholt, die diesen Knoten angehören und mit den entsprechenden Relationen verknüpft. Danach wird für jeden Knoten die gleiche Methode aufgerufen, jedoch mit dem neuen Knoten als Bezugspunkt. Somit wird sichergestellt, dass alle Lernobjekte auf einer Taxonomielinie miteinander verknüpft sind.
Das zweite Beziehungspaar, das überprüft wird, ist **references**/**isReferencedBy**. In der Taxonomie können Referenzbeziehungen mit dem Label *skos:related* angegeben werden. Diese werden mit einer Abfrage, ähnlich der von *IsNarrowerThan*/*isBroaderThan*, überprüft und die damit verbundenen Lernobjekte verknüpft.

HeuristicReasoner

Die Klasse *HeuristicReasoner* überprüft die Inhaltsbeziehungen eines Lernobjekts. Diese sind **IsFormatOf**, **IsAlternativeTo**, **IsIllustratedBy** und **isMoreSpecificThan**/**isLessSpecificThan**. Eine genauere Definition der Beziehungen wurde in Kapitel 4.3.2 gegeben.
Die *HeuristicReasoner*-Klasse überprüft diese Beziehungen anhand von Metadaten und dem Inhalt der Lernobjekte. Die Ähnlichkeit des Lerninhalts wird überprüft durch den Vergleich von Schlüsselwörtern, der Sprache, den zugeordneten Taxonomieknoten und

den bestehenden Relationen.
Als erstes wird eine SQL-Abfrage aus den Schlüsselwörtern und der Sprache des Lernobjekts erstellt. Dazu werden die Schlüsselwörter über den *OR*-Ausdruck miteinander verknüpft und die Anfrage an die Datenbank geschickt, zu sehen in Listing 5.10

```
1  StringBuffer query =
2    new StringBuffer("SELECT learning_object_id
3                      FROM content_repo_keywords
4                      WHERE language="+_elo.getLanguage()+" AND ");
5  for(Iterator<String> it = keywords.iterator(); it.hasNext();){
6    query.append("keyword LIKE '%"+it.next()+"%'");
7    if(it.hasNext()) query.append(" OR ");
8  }
9  Set<Identifier> equiKeywords = new HashSet<Identifier>();
10 if(!keywords.isEmpty()) equiKeywords =
11   new HashSet<Identifier>(_db.getLOsWithQuery(query.toString()));
```

Listing 5.10: Überprüfung der Metadaten für Inhaltsbeziehungen

Zurück kommen alle Lernobjekte, mit gleicher Sprache und ähnlichen Schlüsselwörtern. Als Nächstes wird der Taxonomiepfad untersucht. Da für einen gleichwertigen Inhalt bei allen 4 Eigenschaften mindestens ein Element übereinstimmen muss, wird für die nächste Überprüfung die Lernobjekte analysiert, die bei der vorigen Anfrage zurückgeliefert wurden. Es wird untersucht, ob die Klassifkationen der Lernobjekte den gleichen Taxonomieknoten untergeordnet sind wie die Klassifikationen des zu überprüfenden Lernobjekts. Dafür wird eine rekursive Methode der TaxonomyReasoner-Klasse aufgerufen, die den Taxonomiebaum untersucht und für jeden Knoten, dem die Lernobjekte zugewiesen sind, den am weitesten oben liegenden Knoten zurückliefert. Abbildung 5.5 zeigt an einem Beispiel, wann zwei Lernobjekte den gleichen Taxonomiepfad haben und wann nicht.

Als Letztes werden die Beziehungen überprüft. Dafür werden die übrig bleibenden Lernobjekte aus der Taxonomieüberprüfung genommen und deren Beziehung gegen die Beziehung des zu überprüfenden Lernobjekts verglichen. Wenn das Prädikat und das Objekt in dem Relationstripel übereinstimmt, so hat das Subjekt eine gleiche Beziehung. Von allen Lernobjekten, die bei der letzten Überprüfung übrig geblieben sind, kann angenommen werden, dass sie einen ähnlichen Lernobjekt, wie das zu überprüfende Lernobjekte, besitzen.
Der Aufbau der Methoden zur Überprüfung der Inhaltsbeziehung sind alle ähnlich aufgebaut. Es wird aus den benötigten Metadaten eine SQL-Abfrage erstellt, mit der dann alle Lernobjekte zurückgegeben werden, die ähnlich oder genau gleiche Metadaten aufweisen. Als Beispiel wird die Überprüfung der Beziehung *IsFormatOf* beschrieben. Wie

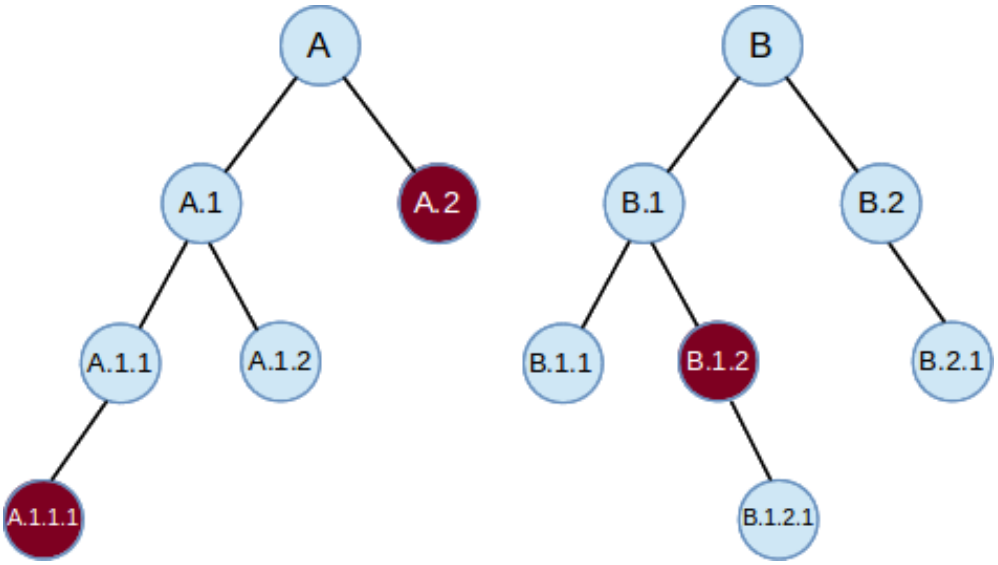

Abbildung 5.5: Zwei verschiedene Taxonomiepfade gegeben von dem gleichen Taxonomieschema. Den dunkelroten Knoten sind Lernobjekte zugewiesen, den anderen nicht.

in Kapitel 4.3.2 aufgeführt, werden neben dem Content, die Metadaten *format*, *context*, *age* und *difficulty* überprüft. Damit die Beziehung *IsFormatOf* erstellt werden kann, muss ein Lernobjekt ein anderes Format, gleichen Inhalt, ähnliche Alterseinstufung und einen ähnlichen Schwierigkeitsgrad als das neue Lernobjekt haben. Elemente, die gleich oder ungleich sein sollen, werden mit einer einfachen Gleichheits-/Ungleichheitsabfrage in der SQL-Query erhalten.

```
1 SELECT id FROM content_repo_learning_objects
2          WHERE mimeFormat != '"+_elo.getMimeFormat()+"'";
```

Listing 5.11: Aufbau der Datenbank-Abfrage für Metadaten

Wenn die Elemente einen ähnlichen Wert als Metadaten haben sollen, muss ein Bereich angegeben werden, in dem das Lernobjekt liegen soll. Beim Alter müssen sich die beiden Altersbereiche überschneiden und zwar so, dass das Durchschnittsalter eines Lernobjekts (L1) innerhalb des Bereichs des anderen Lernobjekts (L2) liegt:

$$L1_{min} < (L2_{min} + L2_{max})/2 < L1_{max} \vee L2_{min} < (L1_{min} + L1_{max})/2 < L2_{max}$$

Beim Schwierigskeitsgrad gibt es fünf Auswahlmöglichkeiten: *very easy*, *easy*, *medium*, *difficult* und *very difficult*. Ein ähnlicher Schwierigkeitsgrad heißt in dem Fall, dass die

Schwierigkeitsgrade nicht mehr als 1 Feld auseinanderliegen dürfen. Wenn ein Lernobjekt zum Beispiel den Schwierigkeitsgrad *easy* hat, so würde ein Lernobjekt mit dem Grad *medium* noch als ähnlich angesehen werden, mit *difficult* jedoch nicht mehr.
Sobald diese Metadaten abgefragt wurden, wird die Schnittmenge aller Abfrageergebnisse gebildet und die übrig gebliebenen Lernobjekte werden mit *isFormatOf* mit dem neuen Lernobjekt verbunden.
Die übrigen Methoden zur Verknüpfung von Inhaltsbeziehungen haben die gleiche Vorgehensweise nur mit anderen Metadaten.

JenaReasoner

Die Klasse *JenaReasoner* setzt Relationen anhand von Schlussfolgerungsregeln mit dem JENA-Framework. Um den Schlussfolgerungsprozess starten zu können, muss zuerst ein Ontologiemodell aufgebaut werden. Dazu muss die verwendete Ontologiesprache angegeben werden (siehe Kap. 3.1.3). In diesem System wird *OWL_DL* als Sprachversion benutzt. Danach muss die Ontologie und die Schlussregeln eingelesen und in das Modell geladen werden.
Anschließend kann das das Modell mit Daten gefüllt werden. Die Daten, die eingelesen werden sind Subjekt-Prädikat-Objekt-Statements. Listing 5.12 zeigt die Methode, mit der das Modell mit Daten gefüllt wird.

```
1  private void fillData(Identifier id){
2    initModel();                              //Model wird neu kreiert
3    _existingRelations = new ArrayList<Relation>();
4    Queue<Vertex> linked_ids = new LinkedList<Vertex>();
5    _used_ids = new ArrayList<Identifier>();
6    linked_ids.add(new Vertex(id, 0));
7
8    while(!linked_ids.isEmpty()){
9      Vertex next_id = linked_ids.poll();
10     _used_ids.add(next_id.id);
11     List<Relation> relations = _db.getRelations(next_id.id);
12     for(Relation r : relations){
13       if(!(linked_ids.contains(r.getObject()) || // Vertex überschreibt
14            _used_ids.contains(r.getObject()) ||  // equals()
15            next_id.depth+1 > MAX_RANGE))         //MAX_RANGE ist 2
16         linked_ids.add(new Vertex(r.getObject(), next_id.depth+1);
17       _data.add(createLearningIndividual(r.getSubject().getId()), //Subj
18                 _data.createProperty(LOM_NS + r.getRelation()),   //Präd
19                 createLearningIndividual(r.getObject().getId())); //Obj
```

```
20          _existingRelations.add(r);
21        }
22      }
23  }
```

Listing 5.12: Die Methode fillData() aus der Klasse *JenaReasoner*. Diese Methode lädt alle Daten aus dem Relationsgraph in das Modell, als Vorbereitung zum Schlussfolgerungsprozess.

In dieser Methode werden auch alle Relationen von den Objekten, die vom Ursprungssubjekt ausgehen, mit in das Modell geladen. So ist sichergestellt. dass das Modell genügend Daten für eine erfolgreiche Schlussfolgerung hat. Zuerst wird das Modell neu kreiert und dabei alle vorigen Daten aus dem Modell gelöscht. Dies hat den Grund, dass auf dem gleichen Modell mehrere Schlussfolgerungsdurchläufe ausgeführt werden und womöglich neue Relationen dazu kommen. Alle Daten neu zu laden ist ein einfacher Weg, um konsistente Daten zu haben. Es werden zwei Listen erstellt, eine für die noch hinzuzufügenden Objekte und eines für die schon hinzugefügten Objekte. Die innere Klasse *Vertex* speichert die Id der Objekte und die Entfernung zum Ursprungssubjekt. In einer Schleife werden solange die angrenzenden Objekte und deren Relationen zum Modell hinzugefügt bis die Entfernung zum Ursprungssubjekt größer als eine vordefinierte Größe ist, in diesem Fall 2. Dies ist in Konformität mit dem 1-Schritt-Konzept in Kapitel 4.3.3. Die Liste *_exisitingRelations* hält alle schon im Modell gespeicherten Relation, damit schon bekannte Relationen nicht doppelt in die Datenbank geschrieben werden.
Nachdem die Daten in das Ontologiemodell geladen wurden, kann der Schlussfolgerungsprozess gestartet werden. Dies passiert in der Methode *start()*. Dort wird an allen Lernobjekten, die in *_used_ids* gesammelt wurden, der Schlussfolgerungsprozess aufgerufen. Dies hat den Grund, dass somit auch neue Relationen erstellt werden, die nicht von dem neu hinzugefügten Objekt ausgehen. Beispielsweise wird an dem neuen Objekt B die Relation $B \xrightarrow{y} C$ erstellt. Dies führt dazu, dass an dem schon bestehenden und früher geschlussfolgerten Objekt A die Schlussregel $A \xrightarrow{x} B \wedge B \xrightarrow{y} C$ erfüllt wird und die Relation $A \xrightarrow{x} C$ erstellt werden kann. Um diese Relation erstellen zu können muss der Schlussfolgerungsprozess mit diesem Objekt aufgerufen werden.
Der Schlussfolgerungsprozess wird an dem Modell mit der Methode *listStatements()* aufgerufen, welche einen Iterator über alle geschlussfolgerten Statements zurückliefert. Die darunterliegende Liste wird der Methode startWithObject() übergeben, die in Listing 5.13 gezeigt wird.

```
1 private void start(List<Statement> stmtlist){
2   for(Iterator<Statement> iter = stmtlist.iterator(); iter.hasNext();){
```

```
3     Statement stmt = (Statement) iter.next();
4
5     //relationen herausfiltern
6     if(stmt.asTriple().getPredicate().getURI().contains(LOM_NS)){
7       Relation next_relation = stmtAsRelationWithDerivation(stmt);
8
9       //Überpüft, ob Statement schon geschlussfolgert wurde
10      if(!(_existingRelations.contains(next_relation) ||
11         _newRelations.contains(next_relation))){
12        //  Überprüfen, ob Relation zu den inkorrekten Relationen gehört
13        if(!next_relation.getRelation().contains("Incorrect")){
14          // Überprüfen, ob Statement korrekt geschlussfolgert wurde
15          if(isCorrectlyInferred(stmt)){
16            // Statement merken
17            _newRelations.add(next_relation);
18            _data.add(stmt);
19            //rekursives Reasoning bis keine neuen Statements
20            //mehr gefunden werden
21            start(_model.listStatements(_indi, null,
22                  (Resource)null).toList());
23          }
24        }
25      } else {
26         //Die Relation wird trotzdem hinzugefügt,
27         //da neue Ableitungen hinzugekommen sein können.
28         //Der Schlussfolgerungsprozess wird jedoch nicht neu gestartet,
29         //da keine "neue" Relation geschlussfolgert wurde
30        _newRelations.add(next_relation);
31      }
32    }
33  }
34 }
```

Listing 5.13: Die Methode *start()* aus der Klasse *JenaReasoner* überprüft alle Statements aus der Schlussfolgerung und fügt korrekte Statements zum Modell und zum Graphen hinzu.

Die Methode iteriert über alle Statements, die geschlussfolgert wurden. Zuerst werden alle Statements herausgefiltert, die keine LOM-Relationen beschreiben, zum Beispiel *rdf:type*-Relationen. Dann wird an den Relationen eine Reihe von Bedingungen geprüft.

1. Existiert die Relation schon?
2. Gehört die Relation zu den inkorrekten Relationen?

3. Ist die Relation korrekt geschlussfolgert worden?

Wenn die Relation schon existiert, kann trotzdem eine Ableitung hinzugekommen sein, die auf diese Relation schließen lässt. Um die Konsistenz des Content-Netzes zu bewahren, müssen daher neue Ableitungen an die schon bestehenden Ableitungen angefügt werden. Daher werden alle Relation den neuen Relationen hinzugefügt, jedoch ohne den Schlussfolgerungsprozess neu zu starten. Beim Speichern in die Datenbank wird dann überprüft, ob neue Ableitungen hinzugekommen sind, welche an die alten Ableitungen angefügt werden. Die entsprechende Speichermethode wird im nächsten Abschnitt gezeigt.
Das Konzept der inkorrekten Relationen wurde in Kapitel 4.3.3 erörtert. Wenn eine Verknüpfung erkannt wird, dessen Schlussfolgerungsregel als inkorrekte Schlussfolgerung gekennzeichnet ist, so wird diese Verknüpfung verworfen. Ob Relationen korrekt geschlussfolgert wurden, kann anhand einer Überprüfung der Ableitungen bestimmt werden. Dabei wird bei den einzelnen Ableitungsschritten überprüft, ob es diese Relationen auch wirklich gibt und ob es inkorrekte Schlussfolgerungsschritte gibt. Wenn alle Bedingungen korrekt durchlaufen wurden, wird die Relation dem Modell und den bekannten Relationen hinzugefügt und der Schlussfolgerungsprozess rekursiv neu gestartet, bis der Schlussfolgerer keine neuen, korrekten Schlussfolgerungen mehr ausgibt.

5.2.3 Ableitungen

JENA bietet die Möglichkeit an, aus dem geschlussfolgerten Statement die Ableitungen, also die zur Schlussfolgerungen führenden Regeln, herauszulesen. Die Ableitungen werden von der Klasse *RuleDerivation* gekapselt. Eine *RuleDerivation* enthält mehrere Tripel, je nachdem wie viele Terme als Bedingung zur Schlussfolgerung führen. Die Ableitung werden in die Form

$$Rule\, x : (\{id_{x1}\}\, rel_1\, \{id_{y1}\})\, (...)\, (\{id_{xn}\}\, rel_n\, \{id_{yn}\}) \backslash n$$

überführt und so mit der Relationen zusammen in der Graphdatenbank gespeichert. Der *líne-separator* wird dafür benutzt, die Ableitungen voneinander zu trennen. Durch die geschweiften Klammern kann gezielt auf die IDs zugegriffen werden. Die runden Klammern trennen die einzelnen Bedingungsterme voneinander.
Vor allem beim Löschen von Relationen spielen die Ableitungen eine wichtige Rolle. An den Ableitungen lässt sich prüfen, welche Relationen gelöscht werden können und welche noch gültig sind. Ein Beispiel wäre, dass ein Objekt B erstellt wird, welches die Relation *IsFormatOf* mit einem anderen Objekt C hat. Weiter wurde angegeben, dass das Objekt

B das Objekt A benötigt, was zu der Relation *A IsRequiredBy B* und *B Requires A* führt. Durch die Schlussfolgerungsregel 12[4] würde nun zwischen dem Objekt A und dem Objekt C die Relation *A IsRequiredBy C* entstehen. Angenommen der Benutzer hat beim Eintragen einen Fehler gemacht und das falsche Objekt als *Requirement* angegeben. Er korrigiert den Fehler, wodurch alle angrenzenden Kanten von B entfernt werden und die neuen, korrekten Relationen gesetzt werden. Ohne Überprüfung der Ableitungen könnte nicht darauf geschlossen werden, dass die Relation *A IsRequiredBy C* nicht mehr korrekt ist. Daher werden beim Löschen alle Kanten der Nachbarn vom gelöschten Element überprüft und gegebenenfalls gelöscht.
Listing 5.14 zeigt die Methode zum Löschen einer Relation aus der Datenbank und wie die Ableitungen dabei benutzt werden.

```
1  public boolean deleteVertex(long id){
2    Vertex v = getVertex(id);
3    if(v == null) return false;
4    List<Edge> inverse = new ArrayList<Edge>();
5
6    //Hole alle Vertices, die 2 Jumps entfernt sind
7    String sep = System.getProperty("line.separator");
8    for(Vertex vx : v.getVertices(Direction.OUT)){
9      for(Edge e : vx.getEdges(Direction.OUT)){
10       String s = e.getProperty(DERIVATION);
11       if(s != null){
12         String[] rules = s.split(sep);
13         boolean remove = true;
14         for(int i = 0; i < rules.length;i++){
15           if(!rules[i].contains("{"+id+"}")){
16           //Wenn die ID des Löschobjekts nicht gefunden wurde und
17           //es die letzte Relation war, wird die Relation nicht gelöscht,
18           //es sei denn, es handelt sich um eine inverse-Relation
19             if(rules.length == 1){
20               if(!rules[i].toLowerCase().contains("inverse"))
21                 remove = false;
22               else inverse.add(e);
23             } else remove = false;
24           } else rules[i] = "";
25         }
26         if(remove) e.remove();
27         else {
```

[4] [rule12: (?A lom:IsRequiredBy ?C) <- (?A lom:IsRequiredBy ?B) (?B lom:IsFormatOf ?C) notEqual(?A, ?C)]

```
28            s = join(rules, sep);
29            e.setProperty(DERIVATION, s);
30          }
31        }
32      }
33    }
34    //gucken, ob die inverse Relation existiert
35    for(Edge e : inverse){
36      boolean b  = true;
37      for(Edge s : e.getVertex(Direction.IN).getEdges(Direction.OUT,
38                        Relation.INVERSE.get(e.getLabel()))){
39        if(s.getVertex(Direction.IN).getProperty(ID).equals(
40                  e.getVertex(Direction.OUT).getProperty(ID))) b = false;
41      }
42      if(b) e.remove();
43    }
44    //Alle ausgehenden und eingehenden Kanten werden gelöscht, dann Vertex
45    for(Edge e : v.getEdges(Direction.BOTH)){
46      _graph.removeEdge(e);
47    }
48    v.remove();
49    return true;
50  }
```

Listing 5.14: Die Methode *deleteVertex(long id)* aus der Klasse *DatabaseGraph*

Es werden alle Knoten untersucht, die in unter drei Kanten mit dem gelöschten Knoten verbunden sind. Nun wird geschaut, ob das gelöschte Objekt an einer Schlussfolgerung beteiligt war, die zur Erstellung dieser Objekte geführt hat. Dafür werden die Ableitungen nach der Id des Objekts durchsucht. Wenn es eine Übereinstimmung gab, wird die Ableitung herausgelöscht. Wenn es eine Ableitung gibt, in der die Id des gelöschten Objekts nicht vorkommt, so kann die Relation erhalten bleiben - außer es handelt sich um eine Inversionsregel. Eine Inversionregel sagt aus, wenn A mit B über eine Relation X verbunden ist, so wird B mit A über die inverse Relation von X verbunden, zum Beispiel ist *IsBroaderThan* die inverse Relation von *IsNarrowerThan* und umgekehrt. Damit wird umgangen, dass zwei inverse Relationen sich gegenseitig referenzieren und daher nicht gelöscht werden können.
Wenn die Relation erhalten bleibt, werden die übrig gebliebenen Ableitungen mit der Relation gespeichert.

5.3 Datenbankkonfiguration

5.3.1 Relationale Datenbank

Die Metadaten der Lernobjekte werden in einer relationalen Datenbank gespeichert. Die verwendete Datenbank ist dieselbe, in der auch die Daten von Diaspora gespeichert werden. Diaspora bietet standardmäßig die Verwaltungssysteme PostgresSQL und MySQL an. Das Datenbanksystem wurde mit MySQL aufgesetzt.
Die Lernobjekttabelle enthält neben den LOM-Metadaten eine User-Id, welche die Id des Erstellers des Lernobjekts referenziert, Timestamps für Erstell- und Bearbeitungsdatum und zwei Flags. Das Eine zeigt an, ob eine Löschanfrage gestellt wurde, das Zweite gibt an, ob das Lernobjekt schon erfolgreich vom Verknüpfungsmodul bearbeitet wurde. Wenn das Verknüpfungsmodul keine erfolgreiche Rückmeldung geliefert hat (siehe 5.1.3), wird das Flag auf 0 gelassen und der Verknüpfungsprozess wird periodisch neu gestartet, bis eine erfolgreiche Antwort geliefert wurde.
Der Grund, warum eine relationale Datenbank für die Lernobjekteigenschaften verwendet wird und nicht die Eigenschaften mit in die Knoten der Graphdatenbank geschrieben werden, ist die gute Verbindung zwischen Rails-Applikationen und SQL-Datenbanken. Über die ActiveRecord-Schnittstelle kann in Rails eine Klasse direkt an die zugehörige Tabelle gelinkt werden, womit direkt auf der Tabelle gearbeitet werden kann. Dies führt zu einer einfachen und schnellen Implementierung der Lernobjekte.

5.3.2 Graphdatenbank

Für die Speicherung des Beziehungsgraphen, auf den in Abschnitt 4.4 eingegangen wurde, wird der Graphserver **Rexster** eingesetzt. Rexster bietet die Graphen über eine REST-Schnittstelle an. REST benutzt HTTP als Kommunikationsprotokoll zwischen dem Server und dem Client. Vorteilhaft ist, dass es sprachneutral ist und es einfach möglich ist, über eine HTTP-Schnittstelle ein eigenes Plugin für die Kommunikation mit dem Rexster-Server zu schreiben. Wie erwähnt wird der Graph aus der Hauptapplikation über Ruby und über das Verknüpfungsmodul über Java angesprochen. Für Java bringt Rexster eine eigene Implementierung mit, für die Rubyimplementierung wurde das HTTParty-Plugin[5] verwendet, um die Verbindung herzustellen. Ein weiterer Vorteil von Rexster ist, dass es sehr leichtgewichtig und schnell zu konfigurieren ist.

[5] https://github.com/jnunemaker/httparty

5.3.3 Synchronisation der Datenbanken

Ein Problem bei der Verwendung von zwei unterschiedlichen Datenbanken ist, wie die Daten zwischen den Datenbanken konsistent gehalten werden. Das Speichern von gleichen Daten in mehreren verteilten Datenbanken wird in der Datenbanktheorie als **Replikation** bezeichnet. Dabei handelt es sich in diesem Fall um eine symmetrische, asynchrone Replikation. Das heißt das Update der Daten wird auf allen Datenbanksystemen ausgeführt, jedoch nicht in einer Transaktion, sondern es wird nur sichergestellt, dass das replizierte Datenbankobjekt konvergiert ist.
Dabei gibt es verschiedene Vorgehensweisen, wie auftretende Konflikte zwischen den System gelöst werden können. Ein Verfahren ist das Master/Slave-Prinzip. Wenn es bestimmte Objekte nur in einer Datenbank gibt, das heißt ein Objekt gelöscht oder erstellt wurde und die Anfrage bei einer Datenbank nicht bearbeitet wurde, so gilt, was in Mastertabelle steht. In diesem Fall wäre das die relationale Datenbank. Um dies zu implementieren, müssen die IDs der Objekte, die aus der Graphdatenbank geholt werden mit den IDs der relationalen Datenbank abgeglichen werden. Gibt es das Objekt mit dieser ID nicht in der relationalen Datenbank, so wird das Objekt umgehend aus der Graphdatenbank entfernt.
Schwieriger gestaltet es sich, wenn ein Objekt bearbeitet wurde. Falls Metadaten verändert wurden, muss der Verknüpfungsprozess erneut gestartet werden und die Relationen neu gesetzt werden. Für dieses Problem lässt sich ein *Timestamp*-Verfahren benutzen, bei dem das Element gewählt wird, welches entweder den jüngeren oder den älteren *Timestamp* hat. In dieser Implementierung wird jedem Objekt eine Versionsnummer gegeben. Diese wird erhöht, wenn das Objekt verändert wurde. Wenn sich die Versionsnummer des Objekts zwischen der relationalen und der Graphdatenbank unterscheidet, wird entweder der Verknüpfungsprozess direkt neu gestartet (Graphdatenbank hat niedrigere Nummer) oder die Versionsnummer der SQL-Datenbank wird erhöht (relationale Datenbank hat niedrigere Nummer). Diese Versionsnummer wird parallel zur ID des Objekts überprüft, wenn Objekte aus der Graphdatenbank geholt werden.

5.3.4 Repository

Das Repository ist der lokale Content-Speicher auf dem Server. Dort werden die lokalen Artikel und die Mediendaten gespeichert. Alle Daten werden mit Titel und einem Timestamp als Namen gespeichert.
Die lokalen Artikel werden direkt als HTML gespeichert und können so einfach im

Frontend angezeigt werden. Die Mediendaten werden aufgrund des in der Datenbank gespeicherten Formats entsprechend wiedergegeben.

6 Auswertung

Nachdem in den vorigen Kapiteln die Komponenten konzipiert und implementiert wurden, geht es nun darum, das Ergebnis zu evaluieren. Für die Auswertung werden Funktionstests gemacht, die die korrekte Funktionalität der Implementierung zeigt. Dabei wird einerseits getestet, ob ein Lernobjekt, welches in das Content-Netz hinzugefügt wird, korrekt verknüpft wird. Andererseits wird getestet, ob bei der Löschung eines Lernobjekts alle damit zusammenhängenden Verknüpfungen korrekt gelöscht oder - im Falle von Ableitungen - korrekt modifiziert werden.
Danach werden die Ergebnisse und das implementierte System diskutiert und ein Fazit gezogen.

6.1 Funktionstests

Die Funktionstests zeigen die korrekte Arbeitsweise des implementierten Systems. Die Testumgebung ist ein lokaler Rechner mit Ubuntu 14.04. Das System wurde nach den Angaben des Mindstone-Wikis[1] aufgesetzt. Es wird Diaspora mit MySQL 5.5.41 benutzt und Rexster 2.6.0 als Graphserver mit RexsterGraph als Graphdatenbank. Die Verwendete Ruby-Version ist 2.1.5. Die Versionen der verwendeten Gems wird in den Gemfiles der Hauptapplikation und Engines spezifiziert. Das Verknüpfungsmodul verwendet das Apache JENA-Framework in der Version 2.11.1. Zur Veranschaulichung und zur Nachvollziehbarkeit des Tests wird der Graph grafisch dargestellt mit Hilfe der Javascript-Bibliothek d3.js[2].

6.1.1 Erstellung eines Lernobjekts

Zur Vorbereitung des Tests werden 5 Lernobjekte erstellt. Die Lernobjekte wurden dabei so erstellt, dass möglichst viele Beziehungen zwischen wenigen Lernobjekten erzeugt werden. Dadurch bleibt der Graph noch übersichtlich und es gibt genug Beziehungen,

[1] https://inet.cpt.haw-hamburg.de/trac-mindstone/wiki/diasetup
[2] http://d3js.org/

um die Funktionsweise des Content-Netzes zu veranschaulichen. Abbildung 6.1 zeigt den Graphen, bevor das Testobjekt hinzugefügt wird. Die Bezeichnung der Knoten ist

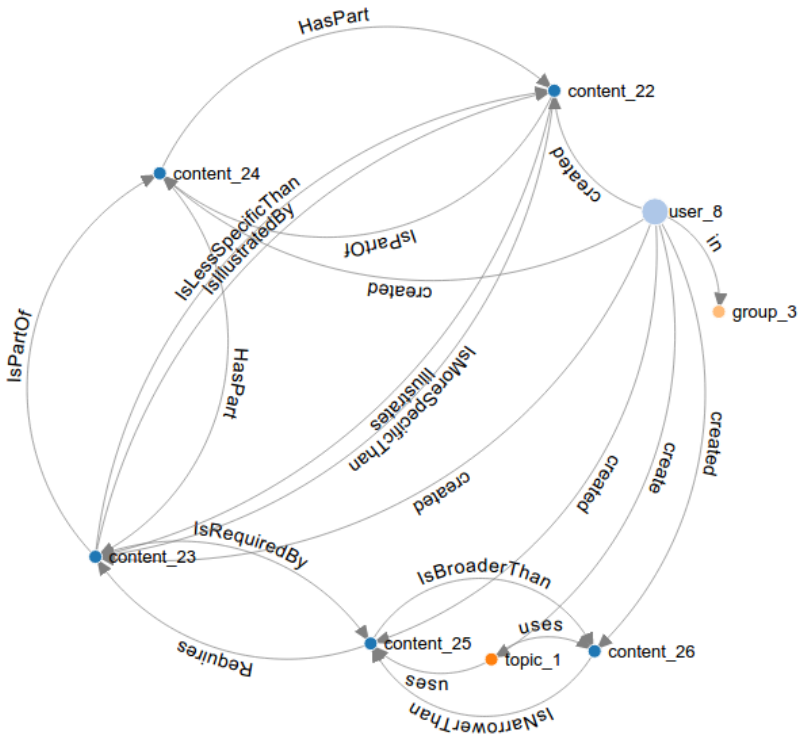

Abbildung 6.1: Content-Graph vor dem Test. Die hellblauen großen Kreise repräsentieren die Anwender. Die kleinen dunkelblauen Kreise sind die Lernobjektknoten. Die hellgelben Kreise sind Lerngruppen und die dunkelgelben Kreise repräsentieren die Themen.

zusammengesetzt aus dem Typ des Objekts und der ID aus der relationalen Datenbank. Zu sehen ist, dass die Lernobjekte mit der ID 23 und 22 Kindobjekte des Objekts 24 sind. Zwischen den beiden Objekten sind zwei Beziehungen erstellt worden, die angeben, dass die Objekte den gleichen Inhalt beschreiben. Objekt 25 benötigt Objekt 23 und hat mit Objekt 26 eine taxonomische Beziehung. Objekt 25 und Objekt 26 sind darüber hinaus Teil von *topic_ 1*.

Das neue Objekt wird so erstellt, dass es gleiche Schlüsselwörter und den gleichen Taxonomiepfad hat wie *content_ 25*. Außerdem wird angeben, dass *content_ 22* benötigt wird.

Ausgehend von diesen Attributen erwarten wir, dass bestimmte Beziehungen erstellt und

geschlussfolgert werden. Zum Überblick sind in Listing 6.1 die Schlussfolgerungsregeln angegeben, die in diesem Test benutzt werden.

```
[rule12: (?A lom:IsRequiredBy ?C) <- (?A lom:IsRequiredBy ?B)
            (?B lom:IsFormatOf ?C) notEqual(?A, ?C)]
[rule13: (?A lom:Requires ?C) <- (?A lom:Requires ?B)
            (?B lom:IsFormatOf ?C) notEqual(?A, ?C)]

[rule14: (?A lom:IsNarrowerThan ?C) <- (?A lom:IsNarrowerThan ?B)
            (?B lom:IsFormatOf ?C) notEqual(?A, ?C)]
[rule15: (?A lom:IsBroaderThan ?C) <- (?A lom:IsBroaderThan ?B)
            (?B lom:IsFormatOf ?C) notEqual(?A, ?C)]

[rule28: (?A lom:IsRequiredBy ?C) <- (?A lom:IsLessSpecificThan ?B)
            (?B lom:IsRequiredBy ?C) notEqual(?A, ?C)]
[rule29: (?A lom:Requires ?C) <- (?A lom:IsMoreSpecificThan ?B)
            (?B lom:Requires ?C) notEqual(?A, ?C)]
```

Listing 6.1: Die für diesen Test wichtigen Regeln im Überblick.

Durch die Angabe, dass *content_22* von dem neuen Objekt benötigt wird, müssen die Beziehungen `22 IsRequiredBy X` und `X Requires 22` erstellt werden (X bezeichnet das neue Objekt). Durch diese Beziehung muss unter Anwendung der Regel 28 die Schlussfolgerung `(23 isLessSpecificThan 22) (22 IsRequiredBy X)` → `(23 IsRequiredBy X)` erstellt werden.

Des Weiteren wurden die Metadaten von dem neuen Objekt so gewählt, dass eine *IsFormatOf*-Beziehung mit *content_25* erstellt wird. Durch diese Verknüpfung kann einerseits die Regel 12 angewendet werden, die zu `22 IsRequiredBy 25` führt und andererseits die Regel 14 aus der die Beziehung `26 IsNarrowerThan X` entsteht.

Nachdem das Objekt erstellt wurde, wird das Verknüpfungsmodul gestartet. Die Ausgabe für den Verknüpfungsprozess zeigt Listing 6.2.

```
Reasoning started...
Processing eLO << 30 >>
Reasoned: 23 IsRequiredBy 30
Reasoned: 30 Requires 23
Reasoned: 22 IsRequiredBy 25
Reasoned: 26 IsNarrowerThan 30
Reasoned: 30 IsBroaderThan 26
Reasoned: 25 Requires 22
Finished.
Relations: [22 IsRequiredBy 25, 30 IsBroaderThan 26, 25 Requires 22,
            30 IsFormatOf 25, 23 IsRequiredBy 30, 25 IsFormatOf 30,
```

```
12              30 Requires 23, 30 Requires 22, 26 IsNarrowerThan 30,
13              22 IsRequiredBy 30]
14 Required time: 3274ms
```

Listing 6.2: Ausgabe des Verknüpfungsmoduls, nachdem das Lernobjekt hinzugefügt wurde.

Die Ausgabe zeigt an, dass 6 Verknüpfungen geschlussfolgert und insgesamt 10 Relationen erstellt wurden. Zu beachten ist, dass nicht nur Relationen geschlussfolgert wurden, die von dem neu hinzugefügten Lernobjekt ausgehen. Jedes Objekt zu denen Relationen über die Schlussfolgerungsregeln aufgestellt werden können, werden selbst noch einmal geschlussfolgert. Somit können Relationen geschlussfolgert werden, die auf den neu hinzugefügten Relationen aufbauen. Als Beispiel sei hier dir Relation `22 IsRequiredBy 25` und `25 Requires 22` aufgeführt, die erstellt wurden aufgrund der vorher geschlussfolgerten Relationen `23 IsRequiredBy 30` und `30 Requires 23`. Zur Veranschaulichung werden in Listing 6.3 die Ableitungen, die als Eigenschaften der Relationen gespeichert sind als JSON-Objekte gezeigt.

```
1 { derivation: "inverse_A: ({23} IsRequiredBy {30})",
2 _id: "802",
3 _type: "edge",
4 _outV: "30",
5 _inV: "23",
6 _label: "Requires" },
7
8 { derivation: "rule28: ({23} IsLessSpecificThan {22})
9                         ({22} IsRequiredBy {30})
10                inverse_A: ({30} Requires {23})
11                rule12: ({23} IsRequiredBy {25}) ({25} IsFormatOf {30})",
12 _id: "801",
13 _type: "edge",
14 _outV: "23",
15 _inV: "30",
16 _label: "IsRequiredBy" }
```

Listing 6.3: Die drei geschlussfolgerten Beziehungen aus der Graphdatenbank als JSON-Objekte

Die Relation `23 IsRequiredBy 30` wurde aufgrund der Schlussfolgerungsregel `(?A IsLessSpecificThan ?B) (?B IsRequiredBy ?C)` erstellt. Nach einem zweiten Schlussfolgerungsdurchlauf, der im Verknüpfungsprozess gestartet wurde, weil neue Relationen gefunden wurden, wurde dieselbe Relation mit der Ableitung über die Regel 12 gefunden. Da es sich um eine neue Ableitung gehandelt hat, wurde diese an die alte

Ableitung angefügt.
Die erstellte Relation `23 IsRequiredBy 30` wurde als Term in der Schlussfolgerungsregel `(?A IsRequiredBy ?B) (?B IsFormatOf ?C)` benutzt, die zur Relation `22 IsRequiredBy 25` und respektive `25 Requires 22` geführt hat, zu sehen in Listing 6.4.

```
{ derivation: "rule12: ({22} IsRequiredBy {30}) ({30} IsFormatOf {25})
               inverse_A: ({25} Requires {22})",
_id: "805",
_type: "edge",
_outV: "22",
_inV: "25",
_label: "IsRequiredBy" },

{ derivation: "inverse_A: ({22} IsRequiredBy {25})",
_id: "808",
_type: "edge",
_outV: "25",
_inV: "22",
_label: "Requires" }
```

Listing 6.4: Die drei geschlussfolgerten Beziehungen aus der Graphdatenbank als JSON-Objekte

Die letzte Relation, die über Schlussfolgerungsregeln erstellt wurde, kann über die Regel `(?A IsNarrowerThan ?B) (?B IsFormatOf ?C)` zurückverfolgt werden.
Diese erstellten Verbindungen können nun auch im Graphen gesehen werden, gezeigt in Abbildung 6.2.

Neben den geschlussfolgerten Beziehungen, wurden noch andere Verknüpfungen aus den Metadaten hergeleitet. Die *Requires/IsRequiredBy*-Beziehung wurde durch die Angabe des *Requirements* erstellt. Die *IsFormatOf*-Beziehung wurde erstellt, weil die beiden Lernobjekte den gleichen Inhalt und Kontext haben, aber ein anderes Format.

6.1.2 Löschen eines Lernobjekts

In diesem Test soll aus dem Graphen ein Objekt gelöscht werden. Der Graph vor der Löschung entspricht der Darstellung in Abbildung 6.2. Aus diesem Graphen soll das Objekt mit der ID 25, welches im Graphen mit dem Titel *content_25* bezeichnet ist, gelöscht werden. Dieses Objekt wurde referenziert in der Ableitung für die Relation `23 IsRequiredBy 30`, `22 IsRequiredBy 25`, `25 Requires 22` und `26 IsNarrowerThan 30`.
Wenn dieses Lernobjekt zur Löschung freigegeben wird, müssen die Ableitungen, die

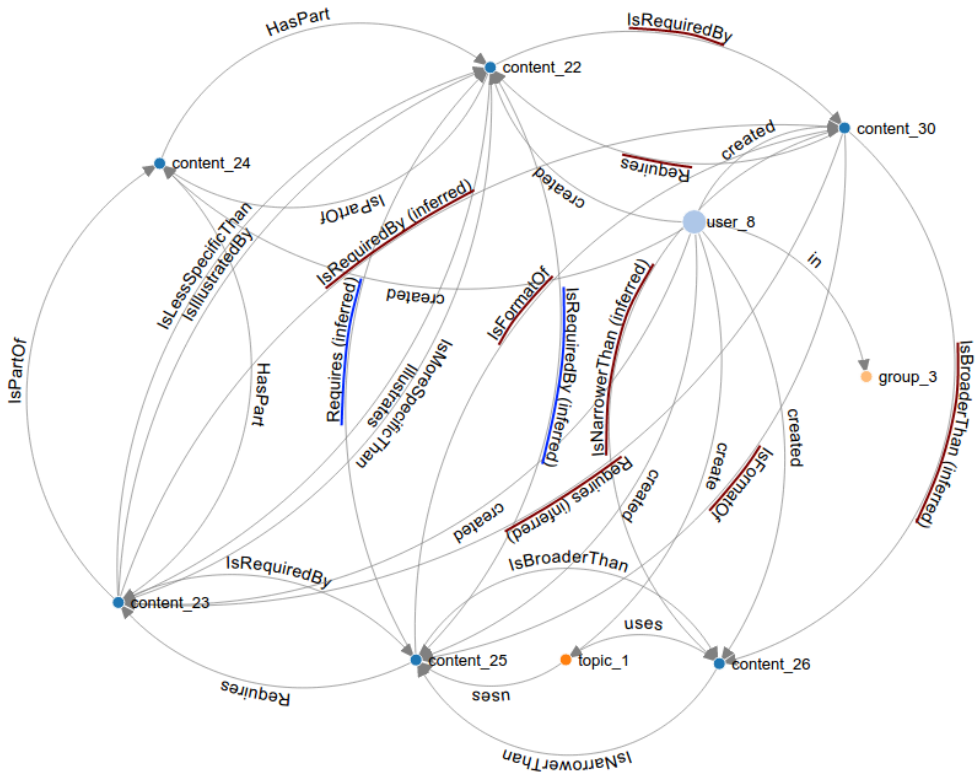

Abbildung 6.2: Der Graph, nachdem das Objekt *content_ 30* erstellt wurde.

mit dem gelöschten Lernobjekt in Verbindung stehen, entfernt werden und die Schlussfolgerungsrelationen dort gelöscht werden, wo keine Ableitungen mehr verfügbar sind. Nachdem das Objekt aus der Graphdatenbank gelöscht wurde, wird es auch aus der relationalen Datenbank entfernt. Solange wird das Objekt mit einer Löschmarkierung versehen.
Nachdem das Objekt zum Löschen freigegeben wurde, wird das Objekt gesperrt. Es kann somit nicht mehr neu referenziert und nicht mehr bearbeitet werden. Solange es noch von einem nicht abgeschlossenen Thema referenziert wird, bleibt das Objekt mit all seinen Beziehungen erhalten. Es wird dann von dem System als *deprecated* angezeigt, zu sehen in Abbildung 6.3.

Sobald eine Verknüpfung von einem Thema zu einem Lernobjekt gekappt wurde, wird erneut überprüft, ob das Objekt gelöscht werden kann. Wenn keine Verbindung mehr zu einem offenen Thema besteht wird die Löschfunktion des Verknüpfungsmoduls aufgerufen.

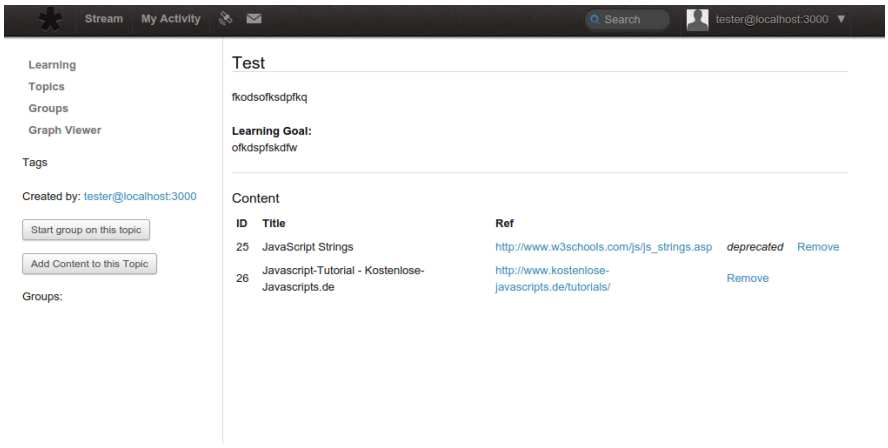

Abbildung 6.3: Die Themenseite mit der *deprecated*-Warnung

Durch die Funktion, die in Kapitel 5.2.3 erläutert wurde, werden alle Relationen, mit dem das Lernobjekt verbunden ist und alle Ableitungen, in dem das Lernobjekt referenziert wurde, gelöscht. Nachdem die Funktion durchgelaufen ist, bleiben 2 geschlussfolgerte Relationen übrig.

```
{ derivation: "inverse_A: ({23} IsRequiredBy {30})",
_id: "802",
_type: "edge",
_outV: "797",
_inV: "564",
_label: "Requires" },

{ derivation: "rule28: ({23} IsLessSpecificThan {22})
                      ({22} IsRequiredBy {30})
               inverse_A: ({30} Requires {23})",
_id: "801",
_type: "edge",
_outV: "564",
_inV: "797",
_label: "IsRequiredBy "},
```

Listing 6.5: Die beiden übrigen Relationen mit Ableitung nach der Löschung

Zu sehen ist, dass die Ableitung `(23 IsRequiredBy 25) (25 IsFormatOf 30)` aus der Relation entfernt wurden. Die Relation bleibt jedoch erhalten, da es noch eine weitere Ableitung gibt, die auf die Relation schließen lässt.
Nachdem das Objekt aus der Graphdatenbank gelöscht wurde, wird dem SOAP-Client

Bescheid gegeben, der die *destroy*-Funktion des Lernobjekts aufruft und damit das Objekt aus der relationalen Datenbank löscht. Abbildung 6.4. zeigt den Graphen nach der Löschung.

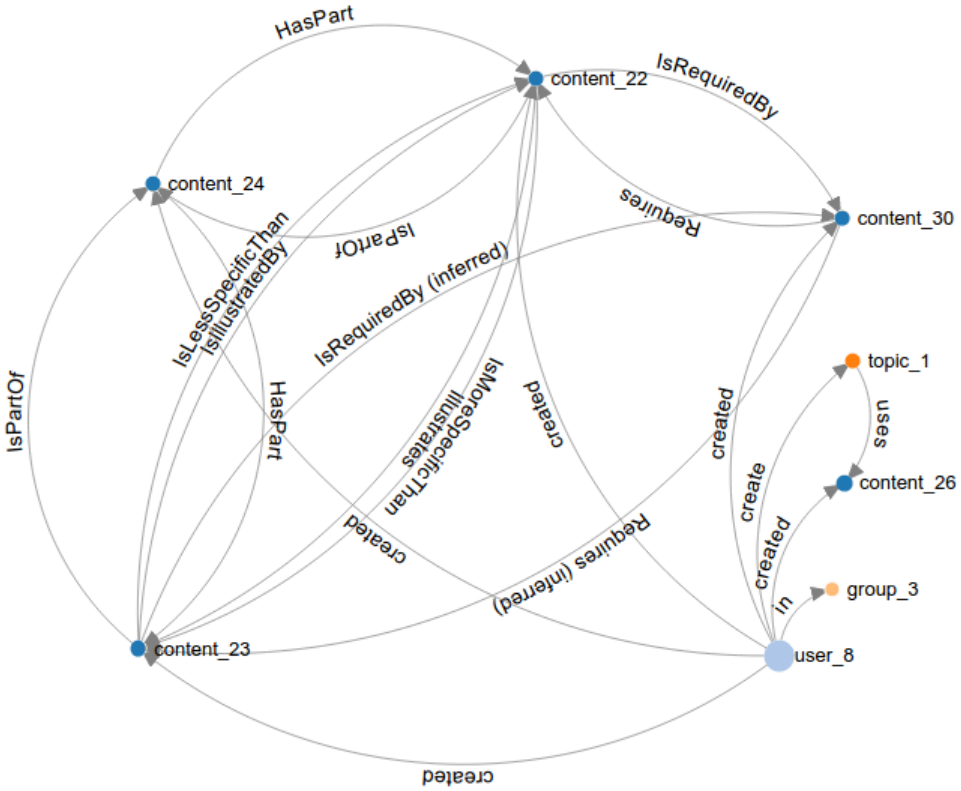

Abbildung 6.4: Der Graph nach der Löschung des Objekt *content_ 25*

6.2 Diskussion & Fazit

Nachdem das System implementiert wurde, kann ein Fazit gezogen werden. Das Ziel der Arbeit war es, ein semantisches Content-Netz zu entwickeln als E-Learning-Erweiterung des sozialen Netzwerks Diaspora. Die Anforderungen waren, eine Verknüpfungskomponente zu implementieren, mit denen Lernobjekte semantisch verknüpft werden können. Das Content-Netz sollte konsistent gehalten werden, so dass es keine ungültigen Relationen im Graphen gibt. Darüber hinaus sollte die gesamte Erweiterungskomponente modular gehalten werden und eine möglichst lose Kopplung zur Hauptapplikation haben, um

einfach angebunden und entfernt werden zu können.

Die Funktionalität des Verknüpfungsmoduls ist den Anforderungen entsprechend eingebaut worden. Die Lernobjekte werden korrekt verknüpft und können sauber gelöscht werden. Dabei werden durch die Nutzung der Ableitungen auch Relationen gelöscht die nicht direkt mit dem zu löschenden Objekt verbunden sind, sondern nur über Schlussfolgerungen erstellt wurden. Dies hilft das Content-Netz konsistent zu halten und falsche Daten zu vermeiden. Durch die Verwendung der SOAP-Schnittstelle kann der gesamte Verknüpfungsprozess asynchron im Hintergrund ausgeführt werden und stört somit nicht die Benutzeranwendung.
Die Lernobjektverwaltung wurde mit den nötigsten Funktionen ausgestattet, um eine reibungslose Funktionalität des semantischen Content-Netzes zu gewährleisten. Jedoch könnte noch weitere Funktionen implementiert werden. Vor allem in Bezug auf die Verbindung mit den Funktionen des sozialen Netzwerks kann die Komponente noch erweitert werden. Dies betrifft die Darstellung der Lernobjekte im sozialen Netzwerk und die Einbindung in die Streams und Aktivitätenseiten von Diaspora.
Des Weiteren wurde die gesamte E-Learning-Komponente lose an die Hauptapplikation gekoppelt und damit modular implementiert. Durch die Benutzung von Engines ist es möglich seinen kompletten Code aus der Hauptapplikation fernzuhalten. Die Einbindung in die Hauptanwendung funktioniert somit über ein paar Zeilen Code. An der Funktionalität der Hauptanwendung musste nichts geändert werden, dadurch kann die Erweiterung auch ohne Probleme entfernt werden, ohne das die Funktionalität des sozialen Netzwerks beeinträchtigt wird.
Die Benutzung einer Graphdatenbank für die Speicherung der Beziehungen ist ein hat gute und schlechte Seiten. Einerseits können die Verknüpfungen sehr einfach gespeichert und dynamisch erweitert werden mit weiteren Relationen. Darüber hinaus ist die Visualisierung des Graphen sehr hilfreich bei der Analyse von Graphmustern, wie Hotspots oder Clustern. Andererseits ist die Verwendung von zwei Datenbanken ein Konsistenzproblem. Da es immer passieren kann, dass die Verbindung unterbrochen wird, können Unstimmigkeiten zwischen den Datenbanken auftreten. In Kapitel 5.3.3 wurde diese Problematik schon diskutiert und Gegenmaßnahmen vorgeschlagen. Diese bringen jedoch einen erhöhten Implementierungs- und Ressourcenaufwand mit sich, da die Synchronizität ständig überprüft werden muss.
Ein weiteres Problem ergibt sich bei der Erstellung der Lernobjekte mit den didaktischen Metadaten. Dieses ist zum Teil ein grundlegendes Problem der Lernobjekte und

der LOM-Struktur, die seit 2002 weitgehend unverändert blieb. Schon 2004 kritisierte Baumgartner (2004) die didaktischen Metadatenstrukturen des LOM-Paradigmen, die wie er meint teilweise unverständlich in ihrer Bedeutung sind. Im Falle dieses Systems, wo es keine Dozenten gibt, die sich mit diesen didaktischen Elementen beschäftigen, fällt das schwierige Verständnis der didaktischen Angaben schwerer ins Gewicht. Entweder müssen jetzigen Angaben verständlicher für den allgemeinen Benutzer gemacht werden oder es müsste eine verständlichere Metadatenstruktur genommen werden. Das Problem dabei ist, dass LOM ein weitverbreiteter Standard ist. Eine Abweichung vom Standard bedeutet ein Kompabilitätsproblem, wenn es darum geht Content aus anderen System zu importieren oder exportieren.
Ungeachtet dessen liefert diese Implementierung des semantischen Content-Netzes eine Antwort auf die anfängliche Fragestellung: „Wie stellen wir relevanten Inhalt für Lerngruppen und Benutzer zur Verfügung?“. Durch die semantische Verknüpfung lässt sich der Inhalt strukturieren, wodurch auf den Benutzer zugeschnittener, relevanter Inhalt angeboten werden kann. Dieser kann durch die Einbindung in Themen von Lerngruppen bearbeitet werden.

7 Zusammenfassung

In der vorliegenden Arbeit wurde ein semantischen Content-Netz im Rahmen einer E-Learning-Erweiterung für das soziale Netzwerk Diaspora konzipiert und implementiert. Diaspora ist ein offenes, verteiltes soziales Netzwerk, bei dem Diaspora-Instanzen auf eigenen, privaten Servern, genannt Pods, installiert werden können und damit einen Teil des gesamten Diaspora-Netzwerks werden können.
Die Funktionsweise des semantischen Content-Netzes ist, dass Lernobjekte erstellt werden, die über ein Verknüpfungsmodul mit semantischen Beziehungen verbunden werden können. Dabei werden die Verknüpfungen einerseits über eine Analyse der Metadaten erstellt, die nach der LOM-Standard strukturiert sind und andererseits werden per Schlussfolgerungsregeln aufgrund von schon bestehenden Verknüpfungen neue Verbindungen erstellt. Dadurch entsteht ein engmaschiges semantisches Netz, in dem die Lernobjekte relativ zueinander eine Bedeutung haben.
Das Konzept des semantischen Content-Netzes baut auf den Arbeiten von hylOs und dem darin implementierten *„Ontological Evaluation Layer“* auf. In diesem Rahmen wurden semantische Relationen definiert und diverse Schlussfolgerungsregeln aufgestellt.
Die Anforderungen, die an das System gestellt wurden, waren eine korrekt arbeitende Verknüpfungskomponente zu implementieren, mit denen Lernobjekte automatisch semantisch in Beziehung gesetzt werden können. Diese sollte autonom arbeiten, so dass die Hauptapplikation nicht durch den Verknüpfungsprozess gebremst wird. Darüber hinaus sollte eine Möglichkeit gefunden werden, die Lernobjekte in das soziale Netzwerk zu integrieren. Die Komponenten sollten modulare Erweiterung sein, mit denen schon bestehende Diaspora-Instanzen einfach und reibungslos erweitert werden können. Des Weiteren sollte das Content-Netz mit den anderen E-Learning- und Systemkomponenten, wie Lerngruppen, Themen und Benutzern konsistent in einer Graphdatenbank gespeichert werden.
Für die Implementierung wurde zuerst eine Lernobjektverwaltung entwickelt, mit denen die Lernobjekte organisiert und zur Verfügung gestellt werden konnten. Daran wurde das Verknüpfungsmodul angeschlossen, mit dem über eine asynchrone Schnitt-

stelle kommuniziert werden kann. Das Verknüpfungsmodul arbeitet vier Phasen ab. Die Strukturbeziehungen, die Taxonomiebeziehungen, die Inhaltsbeziehungen und die Schlussfolgerungen. Die erstellten Relationen werden in einer Graphdatenbank gespeichert und mit den anderen Komponenten verbunden. Ein besonderes Augenmerk hing dabei auf den Ableitungen, die angeben nach welchen Regeln eine Relation geschlussfolgert wurde. Diese werden mit den Relationen gespeichert und werden dazu verwendet, zu erkennen, welche Relationen gelöscht werden können, wenn ein Objekt gelöscht wird.
Das Verknüpfungsmodul wurde vollständig implementiert und erfüllte in den Tests die erwarteten Funktionen. Bei der Lernobjektverwaltung wurden die nötigsten Funktionen implementiert, die für den reibungslosen Ablauf des semantischen Content-Netzes benötigt werden. Funktionen, die noch implementiert werden können, sind eine Versionsverwaltung für Lernobjekte und einen stärkeren Zusammenschluss mit den Funktionen des sozialen Netzwerks.

8 Ausblick

Zum Zeitpunkt der vorliegenden Arbeit wird an der Lerngruppenkomponente als Erweiterung gearbeitet. Dieses umschließt eine automatische Lerngruppenbildung und die Einbindung von Themen und deren Bearbeitung in einer Gruppe.
Als weitere Arbeit muss die dritte Frage in Angriff genommen werden, die in der Einleitung gestellt wurde: *„Wie bekommen wir einen konsistenten Lernprozess mit Feedback und Korrekturmaßnahmen?“*. Dabei geht es einerseits um die Benutzung der Lernobjekte im Kontext des sozialen Netzwerks, das heißt: Wie können Lernobjekte so eingegliedert werden, dass sie ein Teil des sozialen Netzwerksystems werden? Andererseits geht um die Verwendung der semantischen Beziehungen, mit denen zum Beispiel ein Content-Recommendation System aufgebaut werden könnte.
Ein anderes Aufgabengebiet ist die Verbindung von mehreren Pods mit der Lernplattformerweiterung. Diaspora ist als verteiltes soziales Netzwerk konzipiert und dieses Prinzip müsste auch mit der E-Learning-Komponente umgesetzt werden. Dazu zählt der Austausch von Lernobjekten zwischen mehreren Pods. Dies könnte zum Beispiel mit dem SCORM-Referenzmodell realisiert werden, welches ein Standard spezifiziert hat, um einfache Austauschbarkeit zu gewährleisten und eine Wiederverwendbarkeit in verschiedenen Umgebungen ermöglicht. Somit wäre es möglich, die Lernobjekte zwischen entfernten Pods auszutauschen und in das dortige Content-Netz einzugliedern.

Literaturverzeichnis

[Attwell u. a. 2008] Attwell, Graham ; Bimrose, Jenny ; Brown, Alan ; Barnes, Sally-Anne: Maturing learning: Mash Up Personal Learning Environments. In: Wild, Fridolin (Hrsg.) ; Kalz, Marco (Hrsg.) ; Palm, Matthias (Hrsg.): *Proceedings of the First International Workshop on Mashup Personal Learning Environments (MUPPLE08)*. Maastricht, Netherlands, September 2008, S. 78–86

[Baumgartner 2004] Baumgartner, P.: Didaktik und Reusable Learning Objects (RLOs). In: Carstensen, Doris (Hrsg.) ; Barrios, Beate (Hrsg.): *Campus 2004 – Kommen die digitalen Medien an den Hochschulen in die Jahre?* Waxmann, 2004 (Medien in der Wissenschaft), S. 309–325

[Baumgartner und Kalz 2004] Baumgartner, P. ; Kalz, M.: Content Management Systeme für den Bildungsbereich. In: Baumgartner, Peter (Hrsg.) ; Häfele, Hartmut (Hrsg.) ; Maier-Häfele, Kornelia (Hrsg.): *Content Management Systeme in e-Education: Auswahl, Potenziale und Einsatzmöglichkeiten*. 1. Aufl. Studien Verlag, 2004, S. 14–66

[Baumgartner und Kalz 2005] Baumgartner, P. ; Kalz, M.: Wiederverwendung von Lernobjekten aus didaktischer Sicht. In: Tavangarian, D. (Hrsg.) ; Nölting, K. (Hrsg.): *Auf zu neuen Ufern! E-Learning heute und morgen* Bd. 34. Waxmann, 2005, S. 97–106

[Bechhofer u. a. 2004] Bechhofer, Sean ; Harmelen, Frank van ; Hendler, J. ; Horrocks, Ian ; McGuinness, Deborah L. ; Patel-Schneider, Peter F. ; Stein, Lynn A.: *OWL Web Ontology Semantic Reference*. February 2004. – URL http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[Berglund u. a. 2011] Berglund, Andreas ; Boag, Scott ; Chamberlin, Don ; Fernandez, Mary F. ; Kay, Michael ; Robie, Jonathan ; Simeon, Jerome: *XML Path Language (XPath) 2.0 (Second Edition)*. January 2011. – URL http://www.w3.org/TR/2010/REC-xpath20-20101214/

[Berners-Lee 2000] Berners-Lee, T.: *Semantic Web - XML2000*. Presentation at XML 2000 Washington DC. December 2000. – URL http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[Berners-Lee u. a. 2001] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora u. a.: The Semantic Web. In: *Scientific American* 284 (2001), Nr. 5, S. 28–37. – URL http://dx.doi.org/10.1038/scientificamerican0501-34

[Bratt 2007] Bratt, Steve: *Semantic Web, and Other Technologies to Watch*. 2007. – URL http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#%2824%29

[Brauer und Schmidt 2012] Brauer, Steffen ; Schmidt, Thomas C.: Group Formation in eLearning-enabled Online Social Networks. In: Auer, Michael E. (Hrsg.): *Proc. of the International Conference Interactive Computer aided Learning (ICL'12)*. Piscataway, NJ, USA : IEEE Press, September 2012. – URL http://dx.doi.org/10.1109/ICL.2012.6402070

[Brauer u. a. 2013] Brauer, Steffen ; Schmidt, Thomas C. ; Winschu, Andreas: Personal Learning Networks with Open Learning Groups - a Formal Approach. In: Auer, Michael E. (Hrsg.): *Proc. of the International Conference Interactive Computer aided Learning (ICL'13)*. Piscataway, NJ, USA : IEEE Press, Sep. 2013. – URL http://dx.doi.org/10.1109/ICL.2013.6644588

[Brickley und Guha 2014] Brickley, Dan ; Guha, R. V.: *RDF Schema 1.1*. February 2014. – URL http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[Cahier und Zacklad 2004] Cahier, Jean-Pierre ; Zacklad, Manuel: "Socio-Semantic Web" applications: towards a methodology based on the Theory of the Communities of Action. In: *COOP'04 Workshop on Knowledge Interaction and Knowledge Management*, 2004

[Cahier u. a. 2007] Cahier, Jean-Pierre ; Zaher, L'Hédi ; Zacklad, Manuel: Information Seeking in a "Socio-semantic Web" Application. In: *Proceedings of the 2Nd International Conference on Pragmatic Web*. New York, NY, USA : ACM, 2007 (ICPW '07), S. 91–95. – URL http://doi.acm.org/10.1145/1324237.1324248. – ISBN 978-1-59593-859-6

[Couros 2010] Couros, Alec: Developing Personal Learning Networks for Open and Social Learning. In: Veletsianos, George (Hrsg.): *Emerging Technologies in Distance Education*. AU Press, 2010 (Issues in Distance Education), Kap. 6, S. 109–128

[Delphi Group 2000] Delphi Group: Need to know: Integrating e-learning with high velocity value chains. In: *A Delphi Group White Paper* (2000), S. 1–12. – URL http://www.delphigroup.com/whitepapers/pdf/20001213-e-learning-wp.pdf

[deRose u. a. 2010] deRose, Steve ; Maler, Eva ; Orchard, David ; Walsh, Norman: *XML Linking Language (XLink) Version 1.1*. May 2010. – URL http://www.w3.org/TR/xlink11/

[Ehlers 2011] Ehlers, Ulf-Daniel: *Medienbildung und Gesellschaft*. Bd. 15: *Qualität im E-Learning aus Lernersicht*. 2., überarb. u. akt. Aufl. Springer VS, 2011. – URL http://dx.doi.org/10.1007/978-3-531-93070-1

[Engelhardt u. a. 2006a] Engelhardt, M. ; Hildebrand, A. ; Lange, D. ; Schmidt, Thomas C.: Reasoning about eLearning Multimedia Objects. In: Van Ossenbruggen, Jacco (Hrsg.) ; Stamou, Giorgos (Hrsg.) ; Troncy, Raphaël (Hrsg.) ; Tzouvaras, Vassilis (Hrsg.): *Proc. of WWW 2006, Intern. Workshop on Semantic Web Annotations for Multimedia (SWAMM)*. Edinburgh, Scotland, May 2006

[Engelhardt u. a. 2006b] Engelhardt, Michael ; Hildebrand, Arne ; Lange, Dagmar ; Schmidt, Thomas C.: Semantic Overlays in Educational Content Networks – The hylOs Approach. In: *Campus-Wide Information Systems* 23 (2006), September, Nr. 4, S. 254–267. – URL http://dx.doi.org/10.1108/10650740610704126

[Engelhardt u. a. 2004] Engelhardt, Michael ; Hildebrand, Arne ; Schmidt, Thomas C. ; Werlitz, Mathias: The Hypermedia eLearning Object System: Exploiting Learning Objects in a Semantic Educational Web. In: *Proceedings of the 19th Codata Conference*. Berlin, November 2004

[Engelhardt und Schmidt 2003] 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*. Humboldt Universität zu Berlin, September 2003, S. 55–66. – URL http://arxiv.org/abs/cs/0408001v1

[Felder und Silverman 1988] Felder, Richard M. ; Silverman, Linda K.: Learning and Teaching Styles In Engineering Education. In: *Engineering Education* 78 (1988), Nr. 7, S. 674–681

[Feustel u. a. 2001] Feustel, Björn ; Karparti, Andreas ; Rack, Torsten ; Schmidt, Thomas C.: An Environment for Processing Compound Media Streams. In: *Informatica* 25 (2001), July, Nr. 2, S. 201 – 209

[Gamma u. a. 1994] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, October 1994

[Gandon und Schreiber 2014] Gandon, Fabien ; Schreiber, G.: *RDF 1.1 XML Syntax*. February 2014. – URL http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

[Golbeck u. a. 2003] Golbeck, Jennifer ; Parsia, Bijan ; Hendler, James: Trust Networks on the Semantic Web. In: Klusch, Matthias (Hrsg.) ; Omicini, Andrea (Hrsg.) ; Ossowski, Sascha (Hrsg.) ; Laamanen, Heimo (Hrsg.): *Cooperative Information Agents VII* Bd. 2782. Springer Berlin Heidelberg, 2003, S. 238–249. – URL http://dx.doi.org/10.1007/978-3-540-45217-1_18

[Grau u. a. 2008] Grau, Bernardo C. ; Horrocks, Ian ; Motik, Boris ; Parsia, B. ; Patel-Schneider, P ; Sattler, Ulrike: OWL 2: The Next Step for OWL. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6 (2008), November, Nr. 4, S. 309–322. – URL http://dx.doi.org/10.1016/j.websem.2008.05.001

[Grosso u. a. 2003] Grosso, Paul ; Maler, Eva ; Marsh, Jonathan ; Walsh, Norman: *XPointer Framework*. March 2003. – URL http://www.w3.org/TR/xptr-framework/

[Gruber 1993] Gruber, T.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: *International Journal Human-Computer Studies* 43 (1993), November, Nr. 5–6, S. 907–928. – URL http://dx.doi.org/10.1006/ijhc.1995.1081

[Gruber 2007] Gruber, Tom: Collective knowledge systems: Where the Social Web meets the Semantic Web. In: *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 6 (2007), Nr. 1, S. 4–13. – URL http://dx.doi.org/10.1016/j.websem.2007.11.011

[Guarino 1998] Guarino, N.: Formal Ontology and Information Systems. In: Guarino, N. (Hrsg.): *Formal Ontology in Information Systems: Proceedings of the First International Conference (FOIS'98), June 6-8, Trento, Italy*. IOS Presss, 1998, S. 3–15

[Guarino und Giaretta 1995] Guarino, N. ; Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. In: Mars, N. J. I. (Hrsg.): *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing 1995*. Amsterdam : IOS Press, 1995, S. 25–32

[Guarino u. a. 2009] Guarino, Nicola ; Oberle, Daniel ; Staab, Steffen: What is an Ontology? In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer Berlin Heidelberg, May 2009 (International Handbooks in Information Systems), S. 1–17. – URL http://dx.doi.org/10.1007/978-3-540-92673-3_0

[Haß 2013] Hass, Jonne: *Architecture Overview*. June 2013. – URL https://wiki.diasporafoundation.org/index.php?title=Architecture_overview&oldid=770

[Harris und Seaborne 2013] Harris, Steve ; Seaborne, Andy: *SPARQL 1.1 Query Language*. March 2013. – URL http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[Harth u. a. 2011] Harth, Andreas ; Janik, Maciej ; Staab, Steffen: Semantic Web Architecture. In: Domingue, John (Hrsg.) ; Fensel, Dieter (Hrsg.) ; Hendler, James A. (Hrsg.): *Handbook of Semantic Web Technologies*. Springer Berlin Heidelberg, 2011. – URL http://dx.doi.org/10.1007/978-3-540-92913-0_2

[Hayes und Patel-Schneider 2014] Hayes, Patrick J. ; Patel-Schneider, Peter F.: *RDF 1.1 Semantics*. February 2014. – URL http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/

[Jeremić u. a. 2011] Jeremić, Zoran ; Jovanović, Jelena ; Gašević, Dragan: Personal Learning Environments on the Social Semantic Web. In: *Semantic Web* 4 (2011), Nr. 1, S. 23–51

[Lange 2006] Lange, Dagmar: *Entwicklung einer semantischen Verknüpfungs- und Retrievalengine für IEEE-LOM-konforme Lernobjekte innerhalb des Hypermedia Learning Object Systems HyLOS*, Technische Fachhochschule Berlin, Diplomarbeit, January 2006

[Lassila und Swick 1999] Lassila, Ora ; Swick, Ralph R.: *Resource Description Framework (RDF) Model and Syntax Specification*. February 1999. – URL http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[Learning Object Metadata working group 2002] Learning Object Metadata working group: *Draft Standard for Learning Object Metadata*. New York, NY 10016-5997, USA: IEEE Learning Technology Standards Committee (Veranst.), July 2002

[Martindale und Dowdy 2010] Martindale, Trey ; Dowdy, Michael: Personal Learning Environments. In: Veletsianos, George (Hrsg.): *Emerging Technologies in Distance Education*. AU Press, July 2010 (Issues in Distance Education), Kap. 9, S. 177–193

[McCathie Nevile 2014] McCathie Nevile, Charles: *World Wide Web Consortium Process Document*. August 2014. – URL http://www.w3.org/2014/Process-20140801/

[McGuinness und van Harmelen 2004] McGuinness, D. L. ; Harmelen, F. van: *OWL Web Ontology Language Overview*. February 2004. – URL http://www.w3.org/TR/2004/REC-owl-features-20040210/

[Miles und Bechhofer 2009] Miles, Alistair ; Bechhofer, S.: *SKOS Simple Knowledge Organization System Reference*. August 2009. – URL http://www.w3.org/TR/2009/REC-skos-reference-20090818/

[Motik u. a. 2012a] Motik, B. ; Grau, B. C. ; Horrocks, I. ; Wu, Zhe ; Fokoue, Achille ; Lutz, Carsten: *OWL 2 Web Ontology Language Profiles (Second Edition)*. December 2012. – URL http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/

[Motik u. a. 2012b] Motik, B. ; Patel-Schneider, P. F. ; Grau, B. C.: *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. December 2012. – URL http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/

[Nilsson u. a. 2002] Nilsson, Mikael ; Palmér, Matthias ; Naeve, Ambjörn: Semantic Web Meta-data for e-Learning – Some Architectural Guidelines. In: *Proceedings of the 11th World Wide Web Conference*. Honululu, Hawaii, USA, May 2002

[Ochoa u. a. 2011] Ochoa, Xavier ; Klerkx, Joris ; Vandeputte, Bram ; Duval, Eric: On the Use of Learning Object Metadata: The GLOBE Experience. In: Kloos, Carlos D. (Hrsg.) ; Gillet, Denis (Hrsg.) ; Garcia, Racquel M. C. (Hrsg.) ; Wild, Fridolin (Hrsg.) ; Wolpers, Martin (Hrsg.): *Towards Ubiquitous Learning* Bd. 6964. Springer Berlin Heidelberg, 2011, S. 271–284. – URL http://dx.doi.org/10.1007/978-3-642-23985-4_22

[Patel-Schneider und Motik 2012] Patel-Schneider, P. F. ; Motik, B.: *OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)*. December 2012. – URL http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/

[Robinson u. a. 2013] Robinson, Ian ; Webber, Jim ; Eifrem, Emil ; Loukides, Mike (Hrsg.) ; Jepson, Nathan (Hrsg.): *Graph Databases*. First Edition. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., June 2013

[Roreger und Schmidt 2012] Roreger, Hendrik ; Schmidt, Thomas C.: Socialize Online Learning: Why we should Integrate Learning Content Management with Online Social Networks. In: *Proc. of IEEE Intern. Conf. on Pervasive Computing and Communication (PerCom), Workshop PerEL*. Piscataway, NJ, USA : IEEE Press, March 2012, S. 685–690. – URL http://dx.doi.org/10.1109/PerComW.2012.6197601

[S3 Working Group 2002] S3 Working Group: Making Sense of Learning Specifications & Standard: A Decision Maker's Guide to their Adoption / The Maisie Center. Saratoga Springs NY 12866 USA, March 2002. – Forschungsbericht

[Schaffert und Hilzensauer 2008] Schaffert, Sandra ; Hilzensauer, Wolf: On the way towards Personal Learning Evnvironments: Seven crucial aspects. In: *eLearning Papers* (2008), July, Nr. 9

[Schmidt u. a. 2009] Schmidt, Thomas C. ; Hildebrand, Arne ; Engelhardt, Michael ; Lange, Dagmar: From a Link Semantic to Semantic Links - Building Context in Educational Hypermedia / Open Archive: arXiv.org. URL http://arxiv.org/abs/0912.5456v1, 2009. – Technical Report

[Schmidt u. a. 2007] Schmidt, Thomas C. ; Wählisch, Matthias ; Richter, Fritz: hylOs – Hypermedia Learning Object System / link-lab. Berlin, 2007. – White Paper

[Schneider 2012] Schneider, Michael: *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. December 2012. – URL http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/

[Schoop u. a. 2006] Schoop, Mareike ; de Moor, Aldo ; Dietz, Jan L. G.: The Pragmatic Web: A Manifesto. In: *Communications of the ACM 2006* 49 (2006), May, Nr. 5, S. 75–76. – URL http://dx.doi.org/10.1145/1125944.1125979

[Schreiber und Raimond 2014] Schreiber, Guus ; Raimond, Yves: *RDF 1.1 Primer*. June 2014. – URL http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/

[Schulmeister 2005] Schulmeister, Rolf: *Lernplattform für das virtuelle Lernen; Evaluation und Didaktik*. 2. Auflage. Oldenbourg Wissenschaftsverlag GmbH, 2005

[Shadbolt u. a. 2006] Shadbolt, N. ; Hall, W. ; Berners-Lee, T.: The Semantic Web Revisited. In: *Intelligent Systems, IEEE* 21 (2006), Jan, Nr. 3, S. 96–101. – URL http://dx.doi.org/10.1109/MIS.2006.62. – ISSN 1541-1672

[Swartz und Hendler 2001] Swartz, A ; Hendler, J.: The Semantic Web: A Network of Content for the Digital City. In: *Proceedings Second Annual Digital Cities Workshop*. Kyoto, Japan, October 2001

[Torniai u. a. 2008] Torniai, C. ; Jovanovic, J. ; Gasevic, D. ; Bateman, S. ; Hatala, M.: E-Learning meets the Social Semantic Web. In: *Eighth IEEE International Conference on Advanced Learning Technologies, 2008. ICALT '08* (2008), July, S. 389–393. – URL http://dx.doi.org/10.1109/ICALT.2008.20

[Vicknair u. a. 2010] Vicknair, Chad ; Macias, Michael ; Zhao, Zhendong ; Nan, Xiaofei ; Chen, Yixin ; Wilkins, Dawn: A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In: *Proceedings of the 48th Annual Southeast Regional Conference*. New York, NY, USA : ACM, 2010 (ACM SE '10), S. 42:1–42:6. – URL http://doi.acm.org/10.1145/1900008.1900067. – ISBN 978-1-4503-0064-3

[W3C OWL Working Group 2012] W3C OWL Working Group: *OWL 2 Web Ontology Language Document Overview (Second Edition)*. December 2012. – URL http://www.w3.org/TR/2012/REC-owl2-overview-20121211/

[Wiley 2000a] Wiley, David A.: Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. In: Wiley, David A. (Hrsg.): *The Instructional Use of Learning Objects*. Bloomington, 2000, Kap. 1.1

[Wiley 2000b] Wiley, David A.: *Learning Object Design and Sequencing Theory*, Brigham Young University, Dissertation, 2000

[Wilson u. a. 2007] Wilson, Scott ; Liber, Oleg ; Johnson, Mark ; Beauvoir, Phil ; Sharples, Paul ; Milligan, Colin: Personal Learning Environments: Challenging

the dominant design of educational systems. In: *Journal of e-Learning and Knowledge Society – Focus on e-learning 2.0* 3 (2007), June, Nr. 2, S. 27–38

[With 2013] With, Nicolas: Modernisierung des hylOs-Reasoners / HAW Hamburg. November 2013. – Forschungsbericht

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 02.04.2015 Nicolas With