Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Modernisierung des hylOs-Reasoners

Nicolas With

Projekt 2

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein Learning Content Management System hat die Aufgabe Lernmaterialien aller Arten aufzunehmen, sie miteinander zu verknüpfen und diesen Inhalt passend für den spezifischen User zu präsentieren. Das hylOs-System hat viele Module, welche die unterschiedlichen Aufgaben übernehmen.
Der hylOs-Reasoner hat die Aufgabe den Lerninhalt zu verknüpfen und somit ein dichtes semantisches Netz aus Lernmaterialien zu knüpfen. Um dies zu ermöglichen, werden Lerninhalte als electronic Learning Objects (eLOs) verpackt, was die Möglichkeit mitbringt, Lernobjekte als Hierarchien und bestimmte Lerninhalte als Teil von anderen Inhalten darzustellen. Darüber hinaus können die eLOs mit Metadaten versehen werden, die wichtig sind, um Lerninhalte automatisch miteinander in Verbindung zu bringen und verknüpfen zu können.
Darauf aufbauend können dem User unterschiedliche, weiterführende Inhalte präsentiert werden, die seine momentanen Präferenzen widerspiegelt und ihn in seinen Studien weiterhelfen könnten.

1.1 Motivation

Die Diplomarbeit von Dagmar Lange über die Grundlagen und die Ausarbeitung des hylOs-Reasoners wurde 2006 geschrieben. Obwohl danach noch Änderungen gemacht worden sind, ist die Implementierung bis heute weitesgehend unverändert. Nicht nur die Implementierung, sondern auch die benutzten externen Bibliotheken sind auf dem Stand geblieben. Vor allem das Ontologie-Tool JENA hat seitdem signifikante Änderungen erfahren und die Aktualisierung dieses Tools, sowie die Überarbeitung des Codes ist für weitere zukünftige Aufgaben zwingend.
Nach dieser Arbeit wird es die Aufgabe sein, den Reasoner in ein neues System zu übertragen, welches mit dem freien sozialen Netzwerk Diaspora verknüpft wird. Daher muss erst eine moderne Überarbeitung des Reasoners erfolgen.

1.2 Aufgabenstellung

Im Rahmen dieses Berichts soll der hylOs-Reasoner, der von Dagmar Lange geschrieben wurde, auf den neuesten Stand gebracht werden. Dazu werden die Libraries aktualisiert und damit einher gehende Codeverbesserung hinzugefügt, auch mit dem Ziel die Leistung der Anwendung zu erhöhen.
In Kapitel 2 werden die Grundlage und der Ausgangszustand des Reasoners erläutert.
In Kapitel 3 werden die vorgenommenen Änderungen beschrieben und der Ist-Zustand dargestellt.
Abgeschlossen wird dieser Bericht mit der Zusammenfassung und einem Ausblick auf die Masterarbeit in Kapitel 4.

2 hylOs

Die Abkürzung hylOs steht für **hy**permedia **l**earning **O**bject **s**ystem. hylOs bietet eine Lernplattform, dass dem Benutzer hochqualitative, verknüpfte und personalisierte Lerninhalte anbietet.
Der Kern von hylOs ist ein eLearning Content Management System, in dem Lernobjekte erstellt und verwaltet werden. Der Aufbau der Lernobjekte orientiert sich an den Design-Prinzipien von David Wiley[1]. Die eLearning Objects (eLO) werden mit Metadata beschrieben, in Konformität zu dem IEEE LOM-Standard[2], die dazu verwendet wird, die eLOs semantisch zu beschreiben und mit Hyperreferenzen verknüpfen zu können. Daraus entsteht ein engmaschiges Wissensnetz, das dazu beiträgt, dass der Benutzer Lerninhalte vorgeschlagen bekommt basierend auf seinen vorherigen Lerninhalten.
Gespeichert werden die eLOs in einem Repository, dem Media Information Repository (MIR)[3].

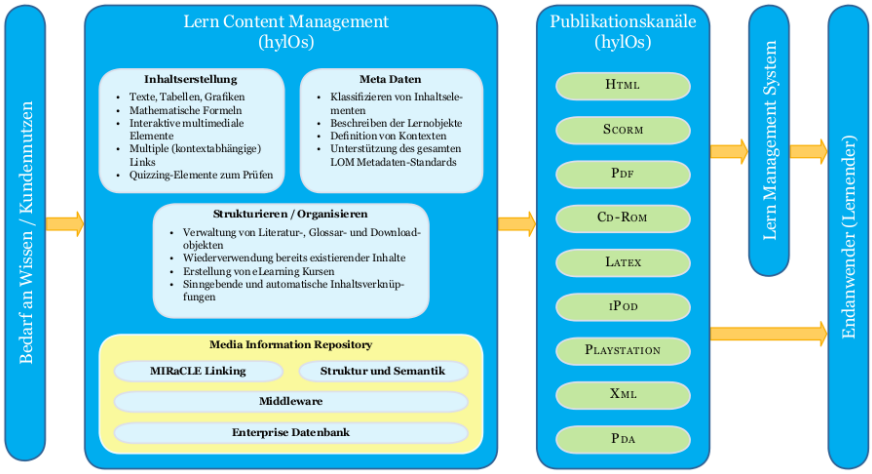

Abbildung 2.1: hylOs Architektur

[1] Wiley (2000)
[2] IEEE:LOM (2002)
[3] Feustel u. a. (2001)

Dort befindet sich auch das Linking System, welches MIR Adaptive Context Linking Environment (MIRaCLE)[4] genannt wird und mit Hilfe der XML-Komponenten XLink, XPath und XPointer Links verarbeitet.[5] [6] [7]

2.1 Ontology Evaluation Layer

Der Ontology Evaluation Layer (OEL)[8], übersetzt etwa die Ontologie Auswertungsschicht, ist die Schlussfolgerungsengine von hylOs und die Basis des hylOs-Reasoners. Die OEL ist dafür zuständig die Lernobjekte automatisch in Beziehung zueinander zu setzen, ausgehend von ihren Metadaten und ihren schon bestehenden Beziehungen. Der LOM-Standard lässt zu, dass Lernobjekte geschachtelt, referenziert und klassifiziert werden können. Diese Eigenschaften werden vom Reasoner benutzt, um Relationen zu setzen. Die Relationen sind eine Erweiterung des Dublin-Core-Metadatenstandard.[9] Die benutzten Relationen sind: (Lange, 2006, 24-26)

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

Das Kernstück des Reasoners ist die Verknüpfungsengine, welche aus vier verschiedenen *Relationship-Finder* aufgebaut ist. Der erste Teil der Verknüpfungsengine ist der *Structure-Relationship-Finder*, der dem Lernobjekt die Relation hasPart/isPartOf anhängt, wenn ein Lernobjekt verschachtelt ist. Der zweite Teil ist der *Taxonomy-Relationship-Finder*, welches aufbauend auf der Taxonomie die Relationen isNarrowerThan/isBroaderThan und references/isReferencedBy zuweist. Der dritte Teil ist der *Heuristics-Relationship-Finder*, welcher mit Hilfe weiterer Metadaten Wahrscheinlichkeiten berechnet, dass ein Lernobjekt die Relationen isFormatOf, isAlternativeTo, is-

[4] Engelhardt und Schmidt (2002)
[5] deRose u. a. (2010)
[6] Berglund u. a. (2011)
[7] Grosso u. a. (2003)
[8] Engelhardt u. a. (2006)
[9] DCMI Usage Board (2012)

LessSpecificThan/isMoreSpecificThan zu anderen Lernobjekten hat. Der letzte Teil ist der *JENA-Relationshi-Finder*, welcher den externen Schlussfolgerer JENA[10] benutzt, um mit Hilfe von Schlussfolgerungsregeln weitere Relationen zu erschließen. Diese Schlussfolgerungsregeln sind mit Hilfe von RDF bzw. die auf RDF aufbauende Sprache OWL beschrieben.

2.1.1 RDF

Das Resource Description Framework ist eine Spezifikation um Ressourcen im Internet zu beschreiben (Manola und Miller, 2004). RDF benutzt die XML-Syntax und wird aus Statements aufgebaut. Ein Statement ist ein Tripel der Form Subjekt-Prädikat-Objekt, wobei alle drei Teile mit einer URI oder als Zeichenkette gekennzeichnet werden. Das **Subjekt** ist die Ressource die beschrieben werden soll. Das **Prädikat** beschreibt die Beziehung des Subjekts zu dem Objekt, auch genannt **Property**. Das **Objekt** ist der Wert des Prädikats. Zur Verdeutlichung folgt ein kleines Beispiel in RDF-Syntax.

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3            xmlns:format="www.dataformat.de/format#">
4   <format rdf:about="www.example.org/datei.doc">
5     <format:open>Microsoft Word</format:open>
6   </format>
7 </rdf:RDF>
```

Die ersten drei Zeilen dienen zur Deklaration der Tags <**rdf**> und <**format**>, die in diesem Beispiel benötigt werden. Die vierte Zeile definiert das Subjekt (*www.example.org/datei.doc*) und in der fünften Zeile wird das Prädikat (*www.dataformat.de/format#open*) und das Objekt (*Microsoft Word*) definiert. Das Objekt muss nicht zwingend eine URI als Bezeichner haben, sondern kann auch mit einem String identifiziert werden. Abbildung 2.2 zeigt das Beispiel nochmal als Graph.

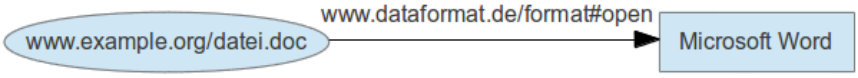

Abbildung 2.2: Darstellung des RDF-Beispiels als Graph

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

Wichtig zum Verständnis der RDF-Syntax ist, dass es sich bei der Kennzeichnung des Tripels nicht um URLs handelt, die Adressen im World Wide Web identifizieren, sondern um URIs, welche zur Kennzeichnung von beliebigen Ressourcen benutzt werden. In diesem Beispiel hat das Subjekt den Identifier *www.example.org/datei.doc*. Dies ist praktisch, denn so können die URIs gegebenenfalls als URL benutzt werden und Informationen zu dem Subjekt liefern oder in diesem Fall direkt zu der Datei verweisen.

RDFS und OWL

RDF ist ausreichend, um Ressourcen im Internet zu beschreiben. Um jedoch Ontologien passend skizzieren zu können, reichen die Möglichkeiten, die RDF bietet, nicht aus. Daher wurde RDFS (Resource Description Framework Schema) entwickelt[11], welches RDF um einfache Klassenkonzepte erweitert. Es können Klassenhierarchien definiert werden und einer Klasse können Eigenschaften zugewiesen werden.
OWL (Web Ontology Language)[12] baut auf RDFS auf, aber erweitert dieses noch um weitere Konzepte. Zum Beispiel können Klassen als *disjoint* gekennzeichnet werden, was bedeutet, dass eine Unterklasse nicht von zwei disjoint-Klassen erben kann. (Apache, 2013)

2.1.2 JENA

JENA 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. Es erlaubt Graphen aus aus RDF- oder OWL-Skripten zu erstellen und umgekehrt. JENA bietet Klassen an, um die Sprachelemente eines RDF-Skriptes als Objekte darzustellen. So kann der RDF-Graph als ein objektorientiertes Modell dargestellt werden. Es können aber auch direkt RDF-Modelle mit JENA erstellt werden:

```
1 Model model = ModelFactory.createDefaultModel();
2 Resource s = model.createResource("www.example.org/datei.dpc");
3 Property p = model.createProperty("www.dataformat.de/format#open");
4 Resource o = model.createResource("Microsoft Word");
5
6 model.add(s, p, o);
7 Statement statement = model.createStatement(s, p, o);
```

[11] Brickley und Guha (2004)
[12] W3C OWL Working Group (2012)

In dem Code-Abschnitt wird der RDF-Graph aus Abbildung 2.2 in JENA implementiert.
Mittels einer Abfragesprache kann das Modell abgefragt werden. Dazu steht die auch von den HP Labs entwickelte Sprache RDQL (Resource Description Query Language), aber seit Version 2.3 auch die von dem W3C entwickelte Sprache SPARQL (SPARQL Protocol And RDF Query Language)[13] zur Verfügung. Die Benutzung der Abfragesprachen ist sehr ähnlich der Benutzung von SQL, mit der sich Datenbanken abfragen lassen. Das nachfolgende Code-Beispiel liefert den Namen des Autors aus dem eben erstellten Modell:

```
1 String queryString = "SELECT ?prog WHERE {<www.example.org/datei.doc>
2   <www.dataformat.de/format#open> ?prog}";
3 Query query = new Query(queryString);
4 query.setSource(model);
5 QueryEngine qe = new QueryEngine(query);
6 QueryResults results = qe.exec();
```

Ähnlich einer SQL-Abfrage wird eine *Query* erstellt, ausgewertet und das Ergebnis als Instanz der Klasse *QueryResults* zurückgeliefert.

Reasoning in JENA

Ein für diese Arbeit sehr wichtiger Aspekt ist die Möglichkeit mit der JENA API Ontologien zu erstellen und durch aufgestellte Regeln Schlussfolgerungen anzustellen, genannt **reasoning**. Es können über die RDF verwandte Sprache OWL Ontologien erstellt werden und über eine Abfragesprache wie SPARQL können Regeln aufgestellt werden, die die aufgestellten Statements gegen die Regel prüfen, um neue Relationen zwischen Ressourcen zu etablieren.

Zur Veranschaulichung kann das Beispiel aus Kapitel 2.1.1 erweitert werden, dargestellt in Abbildung 2.3.
Angenommen es gibt eine zweite Datei, die eine Version der ersten Datei ist, die über die Property *isVersionOf* miteinander verknüpft sind. Nun kann eine Regel aufgestellt werden, die besagt, wenn eine Datei B eine Version von Datei A ist, so kann sie mit dem selben Programm geöffnet werden. Die Beschreibung der Regel könnte folgendermaßen aussehen:

```
[rule1: (?B opensWith ?C) <- (?A isVersionOf ?B) (?A opensWith ?C)]
```

[13] The W3C SPARQL Working Group (2013)

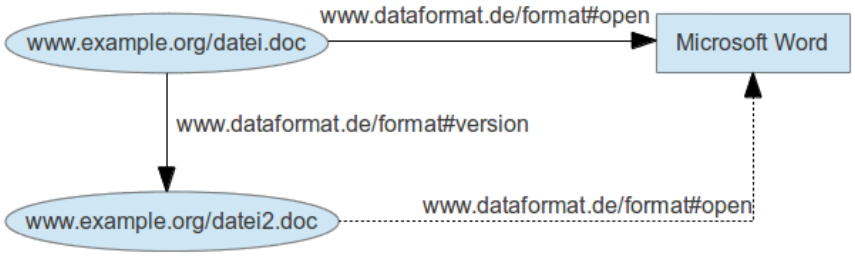

Abbildung 2.3: Darstellung des Ontologie-Beispiels als Graph

In diesem Fall würde der Reasoner anhand der Regel die Verbindung erkennen und eine neue Relation zwischen *datei2.doc* und *Microsoft Word* erstellen, zu sehen in Abbildung 2.3 als gestrichelte Linie.

3 Analyse

Dieser Abschnitt ist in drei Teile unterteilt. Im ersten Teil wird ein genauer Blick auf die verwendeten Bibliotheken des Reasoners geworfen. Zuerst wird skizziert, welche Nutzen die einzelnen Libraries haben und wo sie eingesetzt werden. Danach wird erläutert, wie die Bibliotheken auf den neusten Stand gebracht wurden.
Im zweiten Teil werden wichtige Code-Abschnitte aufgezeigt, um ein Einblick in die Funktionsweise des Reasoners zu bekommen.
Im dritten Teil werden vorgenommene Testläufe, die die Performance des Reasoners testen sollen, präsentiert und evaluiert.

3.1 Bibliotheken

Der hylOs-Reasoner benutzt eine Vielzahl von externen Bibliotheken, die benötigte Funktionen zur Verfügung stellen. Eine genaue Auflistung der Bibliotheken kann Tab. 3.1 entnommen werden.

JENA wurde von Version 2.3 auf 2.10.1 aktualisiert. Viele Bibliotheken wurden nicht mehr benötigt, andere sind dazugekommen. Gelöscht wurden:

- **ANTLR** - ein Analysetool zum lesen, ausführen, schreiben und übersetzen von strukturiertem Text oder Binärfiles
- **icu4j** - Unicode Support für Java
- **ORO** - Tool zum Text-Processing

Ein Großteil der verwendeten Bibliotheken wird für das Framework JENA benötigt, welches in Kapitel 2.1.2 erläutert wurde. Darunter fallen:

- **Xerces** - parsen, modifizieren und generieren von XML-Dokumenten
- **XML-API** - leichtgewichtige, standardisierte Java-XML-API zum validieren, parsen, generieren und transformieren von XML-Dokumenten

Bibliotheken	
alt	**neu**
antlr-2.7.5.jar	*(deleted)*
icu4j_3_4.jar	*(deleted)*
jakarta-oro-2.0.8.jar	*(deleted)*
xml-apis.jar	xml-apis-1.4.01.jar
commons-logging.jar	commons-logging-1.1.3.jar
iri.jar	jena-iri-0.9.6.jar
jena.jar	jena-core-2.10.1.jar jena-tdb-0.10.1.jar jena-arq-2.10.1.jar
junit.jar	junit-4.11.jar
xercesImpl_2.7.1.jar	xercesImpl-2.11.0.jar
slf4j-jdk14-1.7.5.jar	slf4j-api-1.7.5.jar slf4j-log4j13-1.7.5.jar log4j-1.2.16.jar
(nicht verfügbar)	commons-codec-1.6.jar
(nicht verfügbar)	httpclient-4.2.3.jar
(nicht verfügbar)	httpcore-4-2-2.jar

Tabelle 3.1: Unterschiede der benutzten Bibliotheken

- **IRI** (Internationalized Resource Identifier) - Erweiterte Form der URIs mit Unicode-Unterstützung
- **commons-logging** - bietet Logging-Unterstützung
- **commons-codec** - eine definitive Implementation des Base64-Encoders, der die verschiedenen Java-Codecs vereinen soll
- **httpclient** - Bietet mehr Funktionalität im Umgang mit HTTP als die java-seitigen HTTP-Funktionen
- **httpcore** - bietet Funktionen an zum Bauen von HTTP-Client/Server-seitigen Services

Erwähnenswert sind **jena-tdb**, eine Komponente für RDF-Speicherung und Anfragen, und **jena-arq** ein Modul zum Verarbeiten von SPARQL-Anfragen, welche ausgegliedert wurden in eine eigene Bibliothek. Daneben wird **JUnit**, ein Framework für Tests, verwendet und **slf4j**, eine Fassade die hinterliegende Logging-Bibliotheken maskiert, um eine einheitliche Schnittstelle zu bieten, unabhängig von dem eigentlich verwendeten

Logging-Framework. Diese beiden Module wurden auch aktualisiert.
Die Aktualisierung der JENA-Version verlief weitestgehend reibungslos. Der einzige Fehler der auftrat, war ein *NoSuchMethodError*, der im Zusammenhang mit den Logger-Modulen auftrat. Quelle des Problems war der Legacy-Binder **jcl-over-slf4j**, welcher Anfragen an den Logger abgefangen hatte und so den Fehler verursachte. Nach Entfernen des Moduls gab es keine weiteren Probleme mehr und der Reasoner konnte einwandfrei durchlaufen.

3.2 Code-Ausschnitte

Die Main-Klasse des Reasoners ist die LearningObjectLinker-Klasse. Hier werden die *RelationshipFinder* initialisiert und ausgeführt. Die Ausführung geschieht in der Methode **reason()**, hier werden alle Lernobjekte aus der Datenbank geholt und für jedes eLO werden alle *RelationshipFinder* in einer Iterationsschleife nacheinander aufgerufen.

```
1  public void reason() {
2    //Objekte werden aus der Datenbank geholt
3    ...
4    for(Identifier id : los) {
5
6      List<Relation> structure_results;
7
8      /*Struktur- und Taxonomie-RelationshipFinder werden aufgerufen
9        und das Ergebnis in einer Liste gespeichert*/
10     structure_rf.init(id);
11     structure_rf.start(false);
12     taxonomy_rf.init(id);
13     taxonomy_rf.start(false);
14
15     //Heuristik- und JENA-RelationshipFinder
16     heuristics_rf.init(id);
17     heuristics_rf.start(false);
18     jena_rf.init(id);
19     jena_rf.start(true);
20   }
21   ...
22 }
```

In der **init()**-Methoden der jeweiligen *RelationsshipFinder*-Implementationen wird die ID des untersuchten eLOs übergeben. Die **start()**-Methode erzeugt Relationen zwischen dem aktuellen Lernobjekt und für die Beziehung relevante andere Lernobjekte. In der Klasse *RelationshipFinderStructureImpl* zum Beispiel werden die Kind- und Elternobjekte aus der Datenbank geholt und eine isPartOf/hasPart-Beziehung wird zwischen den Lernobjekten etabliert.

```
1  public List<Relation> start(boolean save){
2    ...
3    //Kindlernobjekte verarbeiten
4    List<Identifier> children = db.getChildren(this.id);
5    for (Identifier child : children) {
6      Relation r = new Relation(this.id, HylosLOMVocab.HasPart.
7        getURI().replaceFirst(Constants.DataModel.HYLOS_LOM_NS, ""), child);
8      if(super.isNewRelation(this.db, r)) {
9        relations.add(r);
10       this.derivationHelper.saveDerivationStep(r,
11         new Property(this.id, "parentOf", child));
12     }
13   }
14   //Elternlernobjekte werden verarbeitet
15   ...
16 }
```

In Zeile 6/7 wird die Relation als Tripel aufgestellt mit dem übergebenen Lernobjekt als Subjekt und dem Kindlernobjekt als Objekt. Für das Prädikat wird nur der Teil *HasPart* von der URI *http://hylos.fhtw-berlin.de/HylosLOM#HasPart* genommen.
Den größten Teil der erstellten Relationen werden von der JENA-Engine, aufgrund von Schlussfolgerungsregeln, erzeugt. In der **start()**-Methode der *RelationshipFinderJENAImpl*-Klasse werden alle Statements rausgesucht, die das untersuchte Lernobjekt als Subjekt in diesem Statement beinhaltet. Nachdem überprüft wurde, ob alles korrekt geschlussfolgert wurde, wird dieses Tripel als Schlussfolgerung gespeichert.

```
1  public List<Relation> start(boolean save){
2
3    // Liste aller neu identifizierten Verknüpfungen eines Lernobjektes
4    List<Relation> all_new_relations_for_id = new ArrayList<Relation>();
5
```

```
6   Individual individual =
        this.createLearningObjectIndividual(this.id.toString());
7
8   boolean continue_loop = true;
9   while (continue_loop) {
10    // Liste enthält neue Verknüpfungen, die innerhalb eines
          Schleifendurchlaufs gefunden werden
11    List<Relation> new_rel_in_cycle = new ArrayList<Relation>();
12
13    //es wird über alle Prädikate iteriert
14    for (int j = 0; j < Constants.Relations.RELATIONS.length; j++) {
15      ObjectProperty property = Constants.Relations.RELATIONS[j];
16
17      for (StmtIterator stmtIterator = this.listStatements(individual,
            property, null); stmtIterator.hasNext();) {
18        Statement stmt = (Statement) stmtIterator.next();
19
20        //Statement ist neu und wurde korrekt geschlussfolgert
21        if(!this.seen_rel.contains(this.triple2relation(stmt.asTriple()))){
22          if(!this.isIncorrectRelation(stmt.asTriple())) {
23            if(this.isCorrectlyInferred(stmt)) {
24              new_rel_in_cycle.add(this.triple2relation(stmt.asTriple()));
25              this.saveDerivationSteps(stmt); // Schlussfolgerungskette merken
26            }
27          }
28        } else { // Statement ist nicht geschlussfolgert worden ... }
29      } ...
30    } ...
31  } ...
32}
```

In Zeile 17 werden alle Statements, die mit dem übergebenen Lernobjekt zu tun haben, ausgewählt. Die Methode **listStatements(s,p,o)** greift auf das zugrunde liegende Modell zurück, welches bei dem **init()**-Aufruf aus der **reason()**-Methode aus einem OWL-Skript erbaut wurde.
Solange neue Relationen in einem *cycle* dazukommen, wird das Model aktualisiert und der Vorgang wird wiederholt, bis keine neuen Relationen mehr hinzu gekommen sind und die Schleife beendet wird.

3.3 Tests

Um die Performance und die Lauffähigkeit des Reasoners zu überprüfen wurden Tests laufen gelassen, einerseits noch von Dagmar Lange, andere neu geschrieben. Der Test von Dagmar war ein Durchlauf des Reasoners mit einer kleineren Datenmenge von 22 Elementen (normal: 251 Elemente), so dass der Durchlauf nicht so viel Zeit in Anspruch nahm. Dieser Test wurde vor und nach der Aktualisierung der Bibliotheken vorgenommen. Das Ergebnis kann Tabelle 3.2 entnommen werden, wobei das Ergebnis dem Mittelwert aus mehreren Durchläufen entspricht.

	Version alt	**Version neu**
Zeit in sek.	442	357

Tabelle 3.2: Ergebnis des ReasonerTests vor und nach Aktualisierung

Das Ergebnis zeigt, dass die Aktualisierung des Reasoners einen Performance-Schub von knapp 100 Sekunden bei 22 Elementen gegeben hat, was ungefähr 20 Prozent entspricht.
Als nächstes wurden mehrere Tests mit verschiedenen Objektmengen laufen gelassen, um zu sehen, wie der Reasoner skaliert. Es wurden vier Tests durchgeführt, einer mit 9 Elementen, einer mit 22 Elementen, einer mit 33 Elementen und einer mit 48 Elementen. Wie auch beim Test zuvor wurde der Mittelwert aus mehreren Durchläufen genommen.

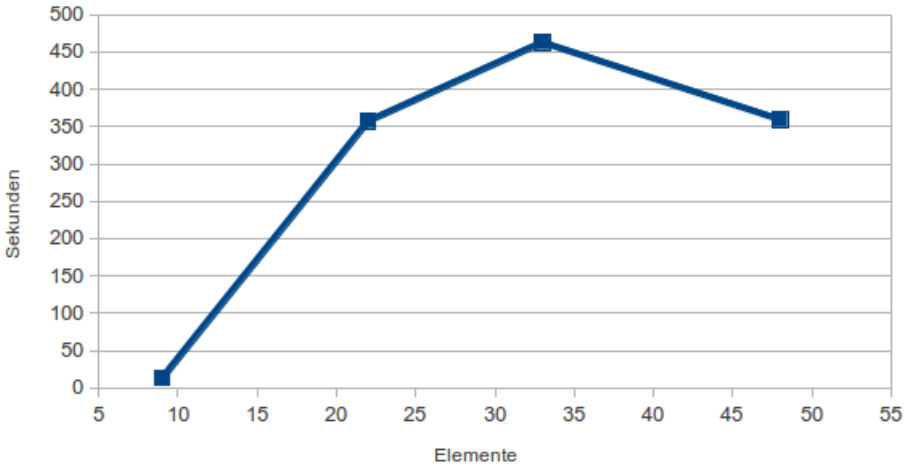

Abbildung 3.1: Testverlauf als Graph

Der Test zeigt, dass der Reasoner mit wenigen Objekten extrem schnell arbeiten kann. Alle Testläufe mit 9 Elementen waren unter 15 Sekunden. Bei steigender Objektmenge steigt die Dauer des Durchlaufs stark an, zum Beispiel dauert ein Durchlauf 30-mal länger bei einer vierfach größeren Objektmenge.
Interessant ist die Zeitmessung bei 22 und 48 Elementen. Bei beiden Durchläufen war die gemessene Zeit sehr ähnlich, obwohl die dritte Objektmenge mehr als doppelt so groß ist wie die zweite. Mögliche Ursachen hierfür sind, dass der eine Elementensatz wesentlich stärker verwoben ist als der andere, das heißt, dass mehr Objekte aufeinander verwiesen haben, so dass mehr Elemente aus der Datenbank geholt werden mussten. Ein weiterer Grund könnte die Menge an Metadaten sein, denn umso mehr Metadaten ein Objekt hat, umso mehr mögliche Verknüpfungen müssen vom Reasoner erzeugt, bzw. überprüft werden.
Dieser Test zeigt, dass die Dauer eines Durchlaufs nicht nur von der Menge der Objekte abhängt, sondern auch von dem Grad der Verknüpfung zwischen den Objekten.

4 Zusammenfassung & Ausblick

Im Rahmen dieser Arbeit wurde der HylOs-Reasoner näher betrachtet und überarbeitet. HylOs ist im Kern ein Learningobject Content Management System, welches nach außen personalisierte und hypermediale Inhalte anbietet. Der HylOs-Reasoner wird benutzt, um die große Menge an Lerninhalten automatisch und sinnvoll miteinander zu verknüpfen, so dass ein semantisches Netz mit Lernobjekten erstellt wird.
Der Reasoner arbeitet mit vier *RelationshipFinder*, die Lernobjekte anhand ihrer Struktur (Kind-, Elternobjekte), der Taxonomie (Klassifizierungen), der Metadaten und anhand von Schlussfolgerungsregeln mit Hilfe des externen Frameworks JENA verknüpfen. Mit JENA können Ontologien, die z.B. mit der Sprache OWL beschrieben wurden, eingelesen und untersucht werden. Mit Abfragesprachen wie SPARQL können Regeln aufgestellt werden, nach denen neue Relationen gesetzt werden.
Im Zuge der Überarbeitung wurden die externen Bibliotheken aktualisiert und nicht mehr benötigte Bibliotheken entfernt oder neu benötigte hinzugefügt.
Zur Evaluation der neuen Version wurden zwei Tests laufen gelassen. Als erstes ein Vergleich der alten Version von Dagmar Lange mit der neuen aktualisierten Version, wobei die neue Version bis zu 20% schneller fertig geworden ist. Als zweites einen Stresstest mit steigender Objektmenge, bei der heraus kam, dass die Dauer des Durchlaufs stark zur Menge der Objekte steigt und die Dauer sehr von der Menge der Metadaten bzw. der Vernetzung der Daten abhängt.

In der folgenden Masterarbeit wird versucht den hylOs-Reasoner in dem Projekt Mindstone einzusetzen, um die content-Netze aufzubauen. Das Ziel von Mindstone ist es, ein *content-centric social network* zu entwickeln, um so ein Netz von semantisch verknüpften Lerninhalten mit den Vorzügen eines sozialen Netzwerks zu verbinden, welches unter anderem die schnelle und einfache soziale Interaktion zwischen Usern ist.
Um das zu erreichen wird das Open-Source Social Network Diaspora[1] eingesetzt, welches als Basis benutzt wird und ein frei einsetzbares, funktionsfähiges soziales Netzwerk mit-

[1] https://diasporafoundation.org/

bringt. Anders als zum Beispiel Facebook hat Diaspora eine **dezentrale** Serverstruktur, bei der jeder Nutzer seinen eigenen Server (genannt *„Pods“*) erstellen kann, auf dem er seine eigene Version von Diaspora laufen hat, so kann Diaspora nach eigenen Wünschen um Funktionen erweitert werden.
In diesem Umfeld soll der Reasoner benutzt werden, um die große Menge an Lerninhalten semantisch zu verknüpfen, die dann in einem *content repository* angeboten werden. Darüber hinaus wird ein Diaspora-Server aufgesetzt, der dann mit dem Repository verbunden wird und so modifiziert wird, dass die verknüpften Lerninhalte sinnvoll eingesetzt werden können.

Literaturverzeichnis

[IEEE:LOM 2002] *Draft Standard for Learning Object Metadata*. 3 Park Avenue, New York, NY 10016-5997, USA, 2002

[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)*. http://www.w3.org/TR/2010/REC-xpath20-20101214/. January 2011

[Brickley und Guha 2004] Brickley, Dan ; Guha, R.V.: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/. February 2004

[DCMI Usage Board 2012] DCMI Usage Board: *DCMI Metadata Terms*. http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements. June 2012

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

[Engelhardt und Schmidt 2002] Engelhardt, M. ; Schmidt, Thomas C.: MIRaCLE - The MIR adaptive Context Linking Environment. In: Auer, Michael E. (Hrsg.) ; Auer, Ursula (Hrsg.): *Interactive computer aided learning (ICL) 2002. International Workshop. Blended Learning*. Villach (Austria), September 2002. – Poster

[Engelhardt u. a. 2006] Engelhardt, Michael ; Hildebrand, Arne ; Lange, Dagmar ; Schmidt, Thomas C.: Reasoning about eLearning Multimedia Objects. In: Ossenbruggen, Jacco V. (Hrsg.) ; Stamou, Giorgos (Hrsg.) ; Troncy, Raphaël (Hrsg.) ; Tzouvaras, Vassilis (Hrsg.): *Proc. of WWW 2006, Intern. Workshop on Semantic Web Annotations for Multimedia (SWAMM)*, URL http://image.ntua.gr/swamm2006/resources/paper06.pdf, May 2006

[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

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

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

[Manola und Miller 2004] Manola, Frank ; Miller, Eric: *RDF Primer*. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. February 2004

[The Apache Software Foundation 2013] The Apache Software Foundation: *Jena Ontology API*. http://jena.apache.org/documentation/ontology/index.html. September 2013

[The W3C SPARQL Working Group 2013] The W3C SPARQL Working Group: *SPARQL 1.1 Overview*. http://www.w3.org/TR/sparql11-overview/. March 2013

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

[Wiley 2000] Wiley, David: *Learning Objects and Sequencing Theory*, Brigham Young University, Dissertation, 2000