UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Erweiterung und Reparatur von Ontologien

Bachelorarbeit

Leipzig, 15.08.2008

Betreuer:
Prof. Dr. Klaus Peter Fähnrich
Dipl. Inf. Jens Lehmann

vorgelegt von

Lorenz Bühmann
geb. am: 15.11.1981

Studiengang Informatik

Zusammenfassung

OWL-Ontologien halten momentan Einzug in vielfältige Anwendungen im World Wide Web und innerhalb von Firmen. Dabei zeigen zunehmen leichtgewichtige, kollaborative Ansätze zur Erzeugung und Nutzung von Ontologien hohes Potential. In einigen Fällen wird dabei ein agiler Prozess verfolgt, d.h. ausgehend von Daten wird erst später ein ausdrucksstarkes Schema entwickelt. Das ist insbesondere dann der Fall, wenn Daten automatisch generiert werden oder Daten aus bereits existierenden Datenbanken übernommen werden. Ziel dieser Arbeit ist die Entwicklung eines Prototypen zur halbautomatischen Anreicherung solcher Ontologien um Axiome und damit verbunden Hilfswerkzeuge um entstehende Inkonsistenzen beseitigen können. Dadurch wird das Pflegen von Ontologien erleichtert. Der Prototyp ist dabei ein in Java geschriebenes Tool ORE, welches eine Ontologie einliest und dem Nutzer ermöglicht, Klassen (neu) zu lernen. Diese neuen Informationen können dann der Ontologie hinzugefügt werden (Erweiterung), und im Anschluss daran mögliche Inkonsistenzen beseitigt werden (Reparatur).

Inhaltsverzeichnis

1 Einleitung

Das World Wide Web (kurz Web) hat unsere Kommunikationsstrukturen, unsere Geschäftsprozesse, die Suche nach Informationen und Unterhaltung revolutioniert - es hat unser tägliches Leben tiefgreifend verändert. Mittlerweile bietet es Zugriff auf eine gigantische Informationsfülle mit Milliarden von Dokumenten.

Doch es hat eine große Einschränkung - es ist für die Nutzung durch den Menschen bestimmt. Das Web basiert technisch auf der Markupsprache HTML[1], die beschreibt wie Informationen dargestellt werden sollen (XHTML[2] + CSS[3]) und wie Informationen miteinander verknüpft werden können, aber nicht was diese Informationen bedeuten. Sucht man zum Beispiel mit einer Suchmaschine nach einem Begriff, ist es dem Computer nicht möglich Homonyme[4] auszuschließen und Synomyme[5] mit als Ergebnis der Suche anzugeben, obwohl das sicherlich wünschenswert wäre. Die Ursache hierfür ist allerdings nicht etwa ein schlechtes Datamining-Verfahren der Suchmaschine, sondern das fehlende Kontextwissen des Computers. Wir Menschen wissen wenn nach einem Begriff suchen, was sich hinter diesem eigentlich verbirgt. Der Computer hingegen kann einfach nur alle Einträge, die syntaktisch dem Suchbegriff entsprechen, als Information liefern. Würden aber zu den jeweiligen Informationen noch semantische Metadaten[6] existieren hätte auch der Computer eine Art 'Verständnis'.

Genau hier setzt die Entwicklung des Semantic Web an, dessen grundlegende Idee es ist, Inhalte im Web so anzureichern, dass sie nicht nur für Menschen verständlich sind. Diese Inhalte sollen auch von Computern zumindest soweit erfasst werden können, dass eine bedingte Automatisierung auf Ebene der Semantik möglich wird. Die dafür notwendige, formale Darstellung komplexer Wissenbeziehungen wird im Bereich Informatik als Ontologie bezeichnet, und soll im Rahmen dieser Arbeit äquivalent zum Begriff Wissensbasis verstanden werden.

Eine so entstandene formale Festlegung von Begriffshierarchien, Relationen und Attributen erlaubt u.a. die Verwendung von Softwarewerkzeugen für Extraktion von Information und Erzeugung von Suchbaren, sie bietet eine gemeinsame Grundlage zum Wissensaustausch und nicht zuletzt auch eine einfachere Möglichkeit zur Bearbeitung.

Das Ziel unserer Arbeit ist es ein Tool ORE zu entwickeln, dass es Nutzern von Ontologien ermöglicht implizites, möglicherweise nützliches Wissen zu extrahieren und der Ontologie hinzuzufügen (Erweiterung). Wir verwenden dafür als Grundlage den DL-Learner, einer in Java geschriebenen Software, die mit Hilfe von Machine Learning Verfahren dieses Wissen extrahiert. Es soll dann mit

[1] http://www.w3.org/MarkUp/

[2] http://www.w3.org/TR/xhtml1/

[3] http://www.w3.org/Style/CSS/

[4] Als Homonym bezeichnet man ein Wort, das für verschiedene Begriffe oder unterschiedliche Einzeldinge steht

[5] Zwei Wörter sind synonym, wenn sie die gleiche (ähnliche) Bedeutung haben

[6] Metadaten sind Informationen die selbst Daten beschreiben

unserem Tool möglich sein dieses Wissen der Wissensbasis hinzuzufügen, und im nächsten Schritt damit möglicherweise entstandene Inkonsistenzen weitgehend zu beheben (Reparatur). Durch die damit erreichte Verbesserung der Ausdrucksstärke der Ontologien, ist eine einfachere Wartung dieser möglich wird.
Aktuell ist es so das zwar einige Software-Tools zum Erstellen und Betrachten von Ontologien existieren (z.B. Protégé[7], KAON[8]), aber das Reparieren von Ontologie noch Forschungsthema[3, 8] ist. Hier nimmt SWOOP[9] eine Art Vorreiterrolle ein, verfolgt dabei allerdings einen anderen Ansatz als unserer Projekt, denn es ist eher auf das Beseitigen von bestehenden Inkonsistenzen ausgerichtet und nutz keine Machine Learning Verfahren zum Erweitern von Ontologien. Zunküftig soll unser Tool sowohl Debuggen/Reparieren als auch das Erweitern als Funktionen integrieren und damit die Stärken von SWOOP mit den unseren kombinieren.

Ausblick

In Kapitel 2 werden wir eine Einführung in die Grundlagen dieser Arbeit geben, und kurz Begriffe und Technologien des Semantic Web beschreiben. Kapitel 3 bechäftigt sich dann mit den Fehlern in Ontologien, die nach dem Erweiterungsprozess entstehen und welche Möglichkeiten zur Behandlung dieser existieren bzw. welche unser Tool bereits elimieren kann. In Kapitel 4 werden wir dann auf die Anforderungen,Architektur und den Ablauf unseres Tools eingehen, bevor in Kapitel 5 eine Beschreibung der Benutzeroberfläche stattfindet. Zum Abschluss beschäftigen uns in Kapitel 6 mit einer Beispiel-Ontologie und zeigen den möglichen Verlauf von Erweiterung und anschließender Reparatur.

[7] http://protege.stanford.edu/
[8] http://kaon.semanticweb.org/
[9] http://code.google.com/p/swoop/

2 Grundlagen

In diesem Kapitel werden wir näher auf die Grundlagen und Basistechnolgien dieser Arbeit eingehen und dabei schon in der Einleitung erwähnte Begriffe genauer defnieren. Als erstes erklären wir den Begriff Semantic Web und gehen dabei auf die Entstehung und einige grundlegende Bestandteile der Architektur ein. Danach beschreiben wir was eine Ontologie ist, und beschränken uns im Rahmen dieser Arbeit auf die Beschreibungssprache OWL. Des weiteren werden wir noch den Zweck eines Reasoners beschreiben und beschreiben im Anschluss den DL-Learner, die Softwaregrundlage unseres Tools.

2.1 Semantic Web

Das Semantic Web ist eine Erweiterung des World Wide Web (WWW), wobei die Begriffsbildung auf den Erfinder des WWW, Tim Berners-Lee, zurückzuführen ist. Dieser sprach von einer Vision

> I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web the content, links, and transactions between people and computers. A Semantic Web, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines...

und veröffentlichte 1998 ein Dokument mit dem Namen 'Semantic Web Road map'[10], in dem er die notwendigen Schritte für die Entwicklung vom WWW zum Semantic Web nannte.
Das Hauptziel des Semantic Web ist, dass Daten sowohl von Programmen als auch von Menschen ausgetauscht und verarbeitet werden können. Es baut dabei auf mittlerweile bestehende Standards auf und nutzt außerdem etablierte Methoden in der Wissensrepräsentation, wobei wir im folgenden auf die wichtigsten Bestandteile aus dem W3C Semantic Web Schichtenmodell(Abb. 1) eingehen, für umfangreiche Informationen aber auf die jeweiligen W3C Empfehlungen verweisen möchten.

Uniform Resource Identifier (URI)[12] URIs dienen zur weltweit eindeutigen Bezeichnung von Ressourcen. Eine Ressource kann dabei jedes Objekt sein, was im Kontext der gegebenen Anwendung eine klare Identität besitzt (z.B. Bücher, Orte, Menschen, Verlage, Beziehungen zwischen diesen Dingen, abstrakte Konzepte usw.).

[10] http://www.w3.org/DesignIssues/Semantic.html
[11] Quelle: http://www.w3.org/2007/03/layerCake.png, März 2007
[12] http://gbiv.com/protocols/uri/rfc/rfc3986.html

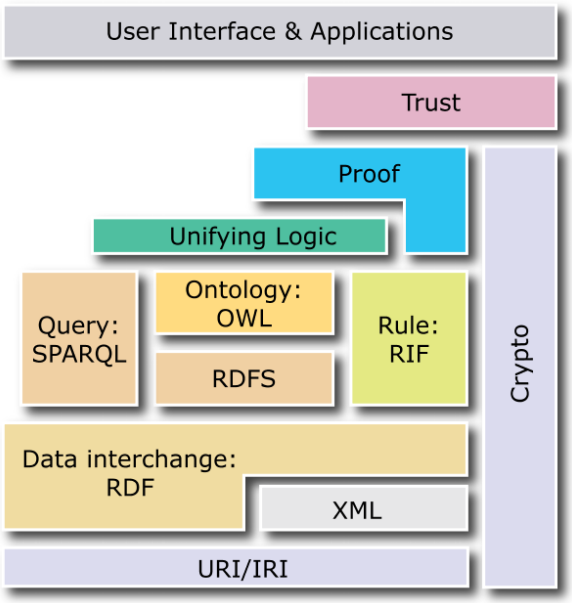

Abbildung 1: Semantic Web Layer Cake, März 2007[11]

XML Die XML Technologiefamilie bildet das Syntaktische Grundgerüst des Semantic Web und besteht im Wesentlichen aus XML[13], XML-Schema[14] und Namespaces[15]. Diese Technolgien ermöglichen, aufbauend auf dem Unicode-Standard, ein von der internationalen Standardisierungs-Organisation ISO genormtes System zur Kodierung von Textzeichen, plattform-unabhängige und selbstbeschreibende Dokumente. Der Vorteil von XML ist die Möglichkeit, basierend auf einem einheitlichen Metamodell, relativ frei Inhalte formulieren zu können.

Resource Description Framework (RDF)[16] RDF ist ein W3C Standard und dient zur Darstellung von Informationen und besitzt im Gegensatz zu XML eine formale Semantik. Die Basisbestandteile sind:

- Aussagen: Eine Aussage besteht aus drei Teilen: Subjekt, Prädikat und Objekt. Das Subjekt ist eine Ressource, über die die Aussage gemacht wird, das Prädikat ist eine bestimmte Eigenschaft und das Objekt der Wert dieser Eigenschaft. Das Objekt einer Aussage kann eine Ressource sein oder ein Literal.
- Ressourcen: Alle in RDF beschriebenen Dinge werden als Ressourcen bezeichnet. Eine Ressource wird durch ihre URI benannt um eine eindeutige Namensgebung zu gewährleisten.

[13] http://www.w3.org/TR/2004/REC-xml11-20040204/
[14] http://www.w3.org/TR/xmlschema-0/
[15] http://www.w3.org/TR/2006/REC-xml-names-20060816/
[16] http://www.w3.org/RDF/

- Eigenschaften: Eine Eigenschaft gibt eine Auskunft über die ihm zugeordnete Ressource. Desweiteren stellt eine Eigenschaft einen Bezug zum Objekt her, verbindet damit eine Ressource mit einem Objekt. Auch Eigenschaften sind Ressourcen und können somit wiederum beschrieben werden.

Es gibt für RDF unterschiedliche Darstellungsformen:
RDF kann z.B. als Graf (Abb. 2) oder in einer RDF/XML Syntax, die relevanter für die Verarbeitung und den Dokumentenaustausch von Computern ist, dargestellt werden.

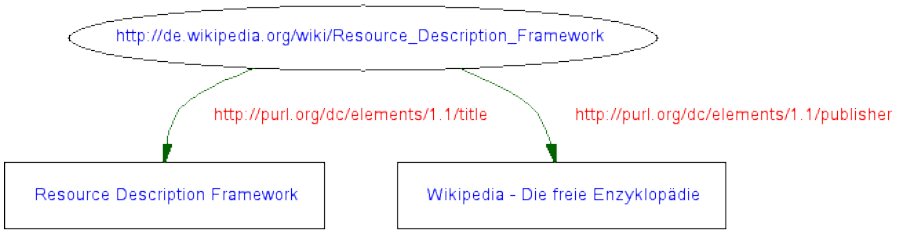

Abbildung 2: Beispiel eines RDF-Graph[17]

RDF-Schema (RDFS)
RDF-Schema ist ist eine semantische Erweiterung zu RDF. Mit dieser Erweiterung lassen sich Gruppen von Ressourcen, die Eigenschaften besitzen, und ihre Beziehungen beschreiben. RDFS ist ausdrucksstärker als RDF und eine Teilmenge von OWL (vgl. Kap.2.4).

2.2 Ontologie

Während der Begriff der Ontologie in der Philosophie für die Lehre vom Sein bzw. vom Seienden steht, wird er in der Informatik in einem etwas anderem Zusammenhang gesehen. Man könnte eine Ontologie kurz als formales Modell eines Wissenraumes bezeichnen, welches ein gemeinsames/geteiltes Verständnis ermöglicht. Die aber wohl bekannteste Definition zumindest unter Informatikern lautet:

> An Ontology is a formal and explicit specification of a shared conceptualisation of a domain of interest.
>
> (Tom Gruber, 1993[2])

[17] Quelle: http://de.wikipedia.org/wiki/Resource_Description_Framework

Eine Konzeptualisierung bezeichnet dabei eine abstraktes Modell von einem Teil der Welt. Unter einer expliziten, formalen Spezifikation versteht man in diesem Zusammenhang, dass die Bedeutung aller Begriffe definiert und damit sowohl für den Menschen als auch für Maschinen verständlich ist. Ontologien beschreiben Klassen, deren Eigenschaften (Attribute) und Beziehungen untereinander, sowie Elemente von Klassen, den Instanzen. Dazu besitzen Ontologien ähnlich wie bei der Objektorientierten Programmierung und dem ER- Schema aus dem Datenbankenbereich eine Klassenhierarchie und Konzepte wie Vererbung. Sie werden in vielen Anwendungsgebieten wie Knowledge Engineering, Integration, Sprachverarbeitung und eben auch dem Semantic Web verwendet, wobei ihnen im letzteren eine Art Schlüsselrolle zugesprochen wird.

2.3 SPARQL

SPARQL ist eine Anfragesprache für RDF Dateien und seit Januar 2008 W3C Standard ([18]). Die Syntax ähnelt dabei der, aus dem Bereich Datenbanken bekannten Anfragesprache, SQL.

2.4 OWL

Zur Modellierung von Ontologien wird heute vermehrt die Web Ontology Language (OWL) verwendet. Zwar ist es auch mit RDF-Schema möglich einfache Ontologien zu erstellen, allerdings sind diese Sprachen in ihrer Ausdrucksstärke beschränkt. OWL ist seit 2004 W3C Standard([19]) und ist in 3 Varianten verfügbar: OWL-Lite, OWL-DL und OWL-Full die sich aufsteigend in ihrer Mächtigkeit unterscheiden. Dabei ist zu erwähnen, dass die Mächtigkeit von OWL-Full dazu führt, dass es nicht mehr entscheidbar ist und somit auch keine vollständigen und korrekten Reasoningverfahren existieren. Wir beschränken uns in dieser Arbeit ausschließlich auf OWL-DL. Sie baut syntaktisch auf RDF auf und basiert semantisch auf der Beschreibungslogik SHOIN(D). Beschreibungslogiken (engl. Description Logics) sind eine Familie von konzeptbasierten Formalismen zur Wissensrepresentation und besitzen eine abstrakte Syntax.
OWL benutzt im Gegensatz zu Beschreibunglogiken URIs als Namen, so wie dies in RDF geschieht. Eine Ontologie in OWL besteht aus einer Menge von Axiomen (Tab. 1) zur Modellierung von Klassen, Individuals und Properties. Eine Klasse ist ein Name und eine Sammlung von Eigenschaften, und beschreibt eine Menge von Individuals. Properties stellen binäre Relationen dar, und unterscheiden sich in ObjectProperties (Individuals mit Individuals) und DatatypeProperties (Individuals mit Datentypen). Auch Properties können Eigenschaften besitzen (z.B. Transitivität) und es ist möglich sie bezüglich Domain (Quellbereich) und Range (Zielbereich) einzuschränken. Dafür, und um auch

[18] http://www.w3.org/TR/rdf-sparql-query/
[19] http://www.w3.org/TR/owl-features/

Axiom	DL Syntax	Beispiel
thing	$\top$	
nothing	$\bot$	
classAssertion	$C(x)$	Mann(Hans)
propertyAssertion	$R(x, y)$	hatKind(Hans, Tina)
subClassOf	$C_1 \sqsubseteq C_2$	Männlich $\sqsubseteq$ Mensch
equivalentClass	$C_1 \equiv C_2$	Mann $\equiv$ Mensch $\sqcap$ Männlich
disjointWith	$C_1 \sqsubseteq \neg C_2$	Männlich $\sqsubseteq \neg$ Weiblich
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{Tim Burners Lee} $\equiv$ {T.B.Lee}
differentFrom	$\{x_1\} \sqsubseteq \neg \{x_2\}$	{Tim} $\sqsubseteq \neg$ {Bill}
subPropertyOf	$R_1 \sqsubseteq R_2$	hatTochter $\sqsubseteq$ hatKind
equivalentProperty	$R_1 \equiv R_2$	kostet $\equiv$ hatPreis
inverseOf	$R_1 \equiv R_2^-$	hatKind $\equiv$ hatEltern
transitiveProperty	$R^+ \sqsubseteq R$	Vorgänger$^+ \sqsubseteq$ Vorgänger
functionalProperty	$\top \sqsubseteq \leq 1R$	$\top \sqsubseteq \leq$ 1hatMutter
inverseFunctionalProperty	$\top \sqsubseteq \leq 1R^-$	$\top \sqsubseteq \leq$ 1hatTochter$^-$

Tabelle 1: Übersicht über die OWL-Axiome

komplexe Klassenbeschreibungen zu ermöglichen, bietet OWL außerdem eine Menge von Konstruktoren an, welche beliebig geschachtelt werden können (Tab. 2)
Für OWL gilt die Open World Assumption und es gilt keine Unique Name Assumption. Open World Assumption bedeutet, dass das Fehlen von Informationen wird nicht als negative Information gewertet wird. Die fehlende Unique Name Assumption führt dazu, dass man Verschiedenheit explizit ausdrücken muss.

Konstruktor	DL Syntax	Beispiel
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Mensch $\sqcap$ Mann
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Frau $\sqcup$ Mann
complementOf	$\neg C$	$\neg$ Weiblich
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{Tim} $\sqcup$ {Bill}
allValuesFrom	$\forall R.C$	$\forall$ hatKind.Arzt
someValuesFrom	$\exists R.C$	$\exists$ hatTochter.Lehrer
maxCardinality	$\leq nR$	$\leq$ 2hatKind
minCardinality	$\geq nR$	$\geq$ 1hatKind

Tabelle 2: Übersicht über die OWL-Konstruktoren

2.5 Reasoner

Ist das Wissen maschinenlesbar vorhanden, benötigt man Software, die diese Daten verarbeitet. Diese Funktion wird durch Reasoner übernommen. Sie können über gegebene Wissensbasen schlussfolgern und damit Wissen ableiten, das nicht explizit beschrieben ist. Dadurch können Reasoner z.B. Anfragen beantworten, die Daten auf Konsistenz prüfen und vollständig klassifizieren. Grundlage für Reasoner ist das Vorliegen des Wissens in einer formalen Sprache, die Inferenz ermöglicht (Beschreibungslogik).
Anforderungen an einen Reasoner sind z.B. Korrektheit, Vollständigkeit, Entscheidbarkeit und Effizienz, wobei die Entscheidbarkeit hauptsächlich von der Ausdrucksmächtigkeit der gewählten Beschreibungssprache abhängt.
Wir verwenden in unserem Projekt vorwiegend Pellet[20], einem Open Source Reasoner auf Java Basis, und FaCT++[21], es existieren aber unter anderem auch andere bekannte Reasoner wie Racer Pro[22] und KAON2[23] für OWL-DL.

2.6 DL-Learner

Der DL-Learner[24] ist ein in Java geschriebenes Tool für das maschinelle Lernen von Konzepten in Beschreibungslogiken. Es lernt dabei Klassendefinitionen in einer vorgegebenen Wissensbasis, indem der Nutzer zusätzlich zum Hintergrundwissen positive und negative Beispiele vorgibt. Positive Beispiele sind dabei Instanzen, die zu der (neu) zu lernenden Klasse gehören sollen, und negative

[20] http://pellet.owldl.com/
[21] http://owl.man.ac.uk/factplusplus/
[22] http://kaon.semanticweb.org/
[23] http://www.racer-systems.com/products/racerpro/index.phtml
[24] http://dl-learner.org

Beispiele Instanzen, die nicht dazugehören sollen. Auf Basis dieser Informationen besteht mit verschiedenen Lernalgorithmen, wie beispielsweise auf Refinement-Operatoren[7, 6] oder genetischen Algorithmen[5] basierenden, die Möglichkeit Klassen zu lernen. Beim Lernen werden dabei unterschiedliche Klassen generiert, die mit Hilfe heuristischer Methoden in ihrer Qualität bewertet werden (Abb. 3). Notwendig für diese Bewertung sind Reasoner, welche über ein DIG-Interface oder die OWL-API angesprochen werden.

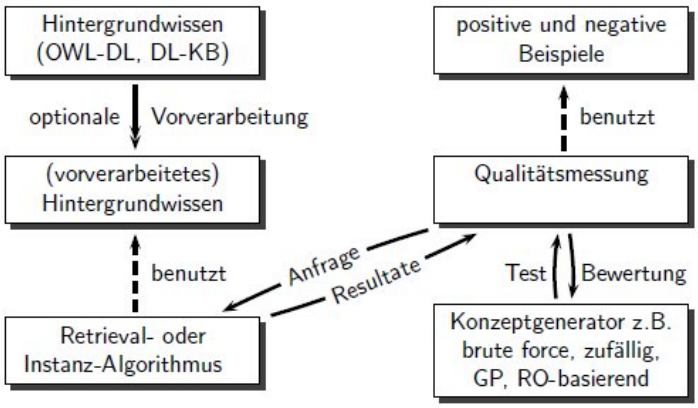

Abbildung 3: Diese Abbildung zeigt den Ablauf beim Lernen von Klassen im DL-Learner[25]

Die Architektur besteht im Kern aus den 4 Komponententypen Lernalgorithmus, Reasoner, Wissensbasis und Lernproblem, die über einen Komponenten-Manager integriert und gesteuert werden. Diese Komponenten-Architektur bietet ein hohes Maß an Flexibilität und Erweiterbarkeit, da somit zum Beispiel Lernalgorithmen ohne viel Aufwand ausgetauscht oder hinzugefügt werden können. Des weiteren werden verschiedene Nutzerschnittstellen wie Kommandozeile, GUI und Webservice bereitgestellt.

[25] Quelle: http://jens-lehmann.org/files/2006_alc_konzepte_lernen_praesentation.pdf

Ein einfaches Lernbeispiel ist das folgende, in DL Syntax gegebene, bei dem eine neue Definiton für die Klasse $father$ gelernt werden soll:

$\texttt{Male} \equiv \neg\texttt{Female}$	$\texttt{Male(MARC)}$
	$\texttt{Male(STEPHEN)}$
$\texttt{hasChild(STEPHEN,MARC)}$	$\texttt{Male(JASON)}$
$\texttt{hasChild(MARC,ANNA)}$	$\texttt{Male(JOHN)}$
$\texttt{hasChild(JOHN,MARIA)}$	$\texttt{Female(ANNA)}$
$\texttt{hasChild(ANNA,JASON)}$	$\texttt{Female(MARIA)}$
	$\texttt{Female(MICHELLE)}$

$$\text{positiv:}\{\texttt{STEPHEN}, \texttt{MARC}, \texttt{JOHN}\}$$
$$\text{negativ:}\{\texttt{JASON}, \texttt{ANNA}, \texttt{MARIA}, \texttt{MICHELLE}\}$$

gelerntes Konzept: $\texttt{Male} \sqcap \exists\texttt{hasChild}.\top$

Abbildung 4: Lernbeispiel in DL-Syntax[26]

[26] Quelle: http://jens-lehmann.org/files/2006_alc_konzepte_lernen_praesentation.pdf

3 Probleme in Ontologien

In diesem Kapitel wollen wir die Probleme analysieren, welche nach dem Hinzufügen von automatisch gelernten Informationen zur Wissensbasis auftreten können. Wir beschreiben dabei sowohl die Probleme, als auch die Lösungsmöglichkeiten dieser und geben im Anschluss einen Überblick darüber was unser Tool bereits lösen kann bzw. was noch nicht.
Wir haben bereits beschrieben, dass eine Klasse anhand positiver und negativer Beispiele in einer bestehenden Wissensbasis gelernt wird. Positive Beispiele sind dabei Instanzen die zu der zu lernenden Klasse gehören sollen und negative sind Instanzen die nicht dazu gehören sollen.
Es können Fehler auftreten, weil der Lernalgorithmus nicht nur 100%ige Lösungen für ein Lernproblem liefern muss. Das bedeutet es wird evtl. eine Klassendefintion vorgeschlagen, die nur einen Teil der positiven Beispiele abdeckt und zu welcher möglicherweise auch negative Beispiele gehören. Fügt man dieses Axiom zur Wissensbasis hinzu, ist ein Reparaturprozess notwendig um eine möglichst hohe Konsistenz zu wahren.
Da es 2 Arten von fehlerhaften Beispielen resp. Instanzzuordnungen gibt, ist auch die Behandlung dieser unterschiedlich.
Beim Lernen kann es dazu kommen, dass komplexe Klassen gelernt werden, die eine Vereinigung (engl. Union) oder Durchschnitt (engl. Intersection) von anderen, möglicherweise auch komplexen Klassen sind. Hier ist es jeweils notwendig rekursiv zu bestimmen, welchen Teil der Klasse die Instanz erfüllt (für neg. Beispiel) bzw. welchen nicht (für pos. Beispiel).

Seien im Folgenden:
$K^{'}$ die neue Wissensbasis
a die fehlerhafte Instanz
C_{alt} die Klasse die neu gelernt wird
C_{neu} die neu gelernte Klasse die zur Wissensbasis K hinzugefügt wird
C ein Teil von C_{neu} der Fehler verursacht, mit $C = A|\neg A|\exists R.D|\forall R.D|\geq nR|\leq nR|nR$
E eine beliebige Klasse in K

Fall 1: positives Beispiel a ist nicht Instanz von C_{neu}
Lösungsmöglichkeiten:

1. *Entfernen der ursprünglichen Instanzzuordung von a zu C_{alt}*
   Axiom, das Instanz a der Klasse C_{alt} zuordnet, wird aus der Wissensbasis K entfernt, wenn es explizit vorhanden ist
   $K^{'} = K \setminus \{C_{alt}\,(p)\}$, wenn gilt: $\exists C_{alt}\,(p) \in K$

2. *Vollständiges Löschen der Instanz a*
   Die Instanz a und alle Axiome, in denen sie vorkommt, werden aus der Wissensbasis K entfernt.
   $K^{'} = K \setminus \{k \in K \,|a$ kommt in k vor $\}$

3. *Vervollständigen von Informationen zur Instanz a*

   (a) es liegt keine Negation der Klasse C vor

   - $C = A$, z.B. $C = Frau$
     i. a der Klasse C zuordnen, vorrausgesetzt C ist kein Komplement einer Klasse zu der a bereits gehört
        $K^{'} = K \cup \{C\,(a)\}$, wenn es kein $D \in K$ gibt für das gilt:
        $K \models D\,(a)$ und $D \sqcap C \equiv \bot$
     ii. verschiebe a von E zu C, wenn $E(a)$ explizit in K steht und C kein Komplement einer Klasse ist, zu der a bereits gehört, mit Ausnahme von E
        $K^{'} = (K \setminus \{E\,(a)\}) \cup \{C\,(a)\}$, wenn gilt:
        $K \models E\,(a)$ und $\nexists D \in K \setminus \{E\}$ für das gilt: $K \models D\,(a)$ und $D \sqcap C \equiv \bot$
   - $C = \forall R.D$, z.B. $C \equiv \forall hatKind.Arzt$
     i. alle Axiome mit der Property R und dem Subjekt a entfernen, in denen die Objekte nicht in D liegen
        $K^{'} = K \setminus \{R\,(a,p)\,|K \not\models D\,(p)\}$
     ii. alle Axiome mit der Property R und dem Subjekt a entfernen
        $K^{'} = K \setminus \{R\,(a,p)\}$
   - $C = \exists R.D$, z.B. $C = \exists hatKind.Arzt$
     i. mindestens ein Axiom mit der Property R, dem Subjekt a und einem Objekt aus D hinzufügen
        $Choices = \{R\,(a,p)\,|K \models D\,(p)\}$
        $K^{'} = K \cup$ mindest einem Element aus $Choices$
   - $C = \leq nR.D$, z.B. $C = \leq 4hatKind.Arzt$
     i. Axiome mit der Property R, dem Subjekt a und Objekten aus D entfernen, bis die Anzahl kleiner gleich der maximalen Anzahl n ist
        Elemente aus K entfernen, bis $|\{R\,(a,p) \in K | K \models D\,(p)\}| \leq n$
   - $C = \geq nR.D$, z.B. $C = \geq 4hatKind.Arzt$
     i. Axiome mit der Property R, dem Subjekt a und einem Objekt aus D hinzufügen, bis die Anzahl größer gleich der minimalen Anzahl n ist

- $C = n\,R.D$, z.B. $C = 4hatKind.Arzt$
  i. Anzahl der Axiome mit der Property R, dem Subjekt a und einem Objekt aus D so anpassen, dass die Anzahl gleich n ist

(b) es liegt eine Negation der Klasse C vor, d.h. $\neg C$

- $C = \neg A$, z.B. $C = \neg Mann$
  i. Klassenzuordung von a zur Klasse C entfernen, wenn Axiom explizit in K steht
  $K^{'} = K \setminus \{C\,(a)\}$, wenn gilt: $\exists C\,(a) \in K$
  ii. verschieben von a von Klasse C zu einer Klasse E, wobei E kein Komplement von einer der Klassen sein darf, zu denen a bereits gehört, mit Ausnahme von C
  $K^{'} = (K \setminus \{C\,(a)\}) \cup \{E\,(a)\}$, wenn gilt:
  $K \models C\,(a)$ und $\nexists D \in K \setminus \{C\}$ für das gilt: $K \models D\,(a)$ und $D \sqcap E \equiv \bot$

Fall 2: negatives Beispiel a ist Instanz von C_{neu}
Lösungsmöglichkeiten:

1. *Instanz a der Klasse C_{alt} zuordnen*
   Es wird ein Axiom dass a der Klasse C zuordnet zur Wissensbasis K hinzugefügt
   $K^{'} = K \cup \{C\,(a)\}$

2. *Vollständiges Löschen der Instanz*
   Die Instanz a und alle Axiome, in denen sie vorkommt, werden aus der Wissensbasis K entfernt.
   $K^{'} = K \setminus \{k \in K\,|a$ kommt in k vor $\}$

3. *Vervollständigen von Informationen zur Instanz*

   (a) es liegt keine Negation der Klasse C vor

   - $C = A$, z.B. $C = Frau$
     i. Axiom, das Instanz a der Klasse C_{alt} zuordnet, wird aus der Wissensbasis K entfernt wenn es explizit vorhanden ist
     $K^{'} = K \setminus \{C_{alt}\,(p)\}$, wenn gilt: $\exists C_{alt}\,(p) \in K$
     ii. verschieben von a von Klasse C zu einer Klasse E, wobei E kein Komplement der Klassen sein darf, zu denen a bereits gehört, mit Ausnahme von C
     $K^{'} = (K \setminus \{C\,(a)\}) \cup \{E\,(a)\}$, wenn gilt:
     $K \models C\,(a)$ und $\nexists D \in K \setminus \{C\}$ für das gilt: $K \models D\,(a)$ und $D \sqcap E \equiv \bot$
   - $C = \forall R.D$, z.B. $C \equiv \forall hatKind.Arzt$

i. mindestens ein Axiom mit der Property $hasChild$, dem Subjekt a und einem Objekt, das nicht in D liegt, hinzufügen
$Choices = \{R(a,p) | K \not\models D(p)\}$
$K' = K \cup$ mindest einem Element aus $Choices$

- $C = \exists R.D$, z.B. $C = \exists hatKind.Arzt$
  i. alle Axiome der Property R und dem Subjekt a entfernen, in denen die Objekte in D liegen
  $K' = K \setminus \{R(a,p) | K \models D(p)\}$
  ii. alle Axiome mit der Property R und dem Subjekt a entfernen
  $K' = K \setminus \{R(a,p)\}$
- $C = \leq nR.D$, z.B. $C = \leq 4hatKind.Arzt$
  i. Axiome mit der Property R, Subjekt a und einem Objekt aus D hinzufügen, bis die Anzahl größer als n ist
- $C = \geq nR.D$, z.B. $C = \geq 4hatKind.Arzt$
  i. Axiome mit der Property R, dem Subjekt a und einem Objekt aus D entfernen, bis Anzahl kleiner als n ist
  Elemente aus K entfernen, bis $|\{R(a,p) \in K | K \models D(p)\}| < n$
- $C = n\,R.D$, z.B. $C = 4hatKind.Arzt$
  i. Anzahl der Axiome mit der Property R, Subjekt a und einem Objekt D so anpassen, dass die Anzahl ungleich n ist

(b) liegt eine Negation vor, d.h. $\neg C$

- $C = \neg A$ ist vom Typ NamedClass z.B. $C = \neg Mann$
  i. a der Klasse C zuordnen, vorrausgesetzt C ist kein Komplement einer Klasse, zu der a bereits gehört
  $K' = K \cup \{C(a)\}$, wenn es kein $D \in K$ gibt für das gilt:
  $K \models D(a)$ und $D \sqcap C \equiv \bot$
  ii. verschiebe a von E zu C, wenn $E(a)$ explizit in K steht und C kein Komplement einer Klasse ist, zu der a bereits gehört, mit Ausnahme von E
  $K' = (K \setminus \{E(a)\}) \cup \{C(a)\}$, wenn gilt:
  $K \models E(a)$ und $\nexists D \in K \setminus \{E\}$ für das gilt: $K \models D(a)$ und $D \sqcap C \equiv \bot$

Es sei jedoch erwähnt, dass die aufgeführten Lösungsvorschläge keine sinnvollen Lösungen sein müssen. Es ist die Aufgabe des Nutzers, zu entscheiden welche Lösungen sinnvoll sind, und welche nicht.
Unser Tool deckt momentan alle oben gelisteten Fälle bis auf die Kardinalitätsrestriktionen der

Object Properties ab. Hier fehlt aktuell ein Konzept zur intuitiven Darstellung für die Lösungsvorschläge.
Weiterhin ist zu erwähnen, das unsere oben genannten Fälle lediglich einen kleinen Teil aller Fehlerarten abdecken. Eine Optimierung der bisherigen Lösungsvorschläge, sowie die Bearbeitung weiterer bekannter Inkonsistenzursachen in Ontologien ist aufgrund des Umfangs für weiterführende Arbeiten geplant.

4 ORE Tool

In diesem Kapitel beschreiben wir den Aufbau von unserem Programm ORE und gehen dabei auf die funktionalen und nicht funktionalen Anforderungen, sowie die Archtitektur ein. Anschließend beschreiben wir noch den Prozessablauf des Programms.

4.1 Anforderungen

Zu den funktionalen Anforderungen gehören:

- Es sollen 2 verschiedene Möglichkeiten zum Einlesen einer Wissensbasis verfügbar sein. Die 2 Arten sollen OWL-Dateien und SPARQL Anfragen sein.
- Es sollen automatisch für eine ausgewählte Klasse sowohl positive als auch negative Beispiele ausgewählt werden.
- Es sollen die besten Ergebnisse vom Lernalgorithmus mit ihrer Genauigkeit angezeigt werden.
- Es sollen für eine ausgewählte neu gelernte Klasse fehlerhafte Instanzen angezeigt werden.
- Das Tool soll Lösungsvorschläge für die Reparatur fehlerhafter Beispiele anbieten. (vgl. Kap. 3)

Nicht funktionale Anforderungen sind:

- Es soll eine möglichst hohe Benutzerfreundlichkeit bieten, um einen leichten Einstieg in die Reparatur von Ontologien zu ermöglichen.
- Es soll robust gegenüber fehlerhaften Verhalten der Nutzer sein.

4.2 Architektur

Wir haben wir für unser Tool das MVC-Pattern benutzt, einem Entwurfsmuster der Softwareentwicklung bei dem Model (die Daten), View (die Darstellung) und Controller (die Steuerung zwischen View und Model) getrennt werden. Um eine möglichst geringe Einstiegsschwelle in das Programm zu gewährleisten haben wir uns bei der grafischen Darstellung für einen Wizard entschieden. Diese einfache Form der Darstellung führt einen Benutzer schrittweise durch mehrere aufeinanderfolgende Dialoge und bietet dabei zudem eine Hilfestellung bei der Interaktion in den jeweiligen Dialogen an. Zudem ermöglicht er das bestimmte Eingaben die für spätere Funktionen notwendig sind, zur Ausführungszeit dieser bereits vorhanden

bzw. abgeschlossen sind. Der Wizard besteht im Kern aus den Klassen Wizard, WizardModel und WizardController und verwendet für die einzelnen Dialoge je eine Klasse vom Typ WizardPanelDescriptor, welcher ein Panel steuert (Abb. 5). Die Panels dienen jeweils als Container für die grafischen Elemente mit denen der Nutzer interagieren kann. Als grafische Schnittstellen des Wizards wurden sowohl Java Swing als auch Java AWT, beide in der JRE enthalten, verwendet.

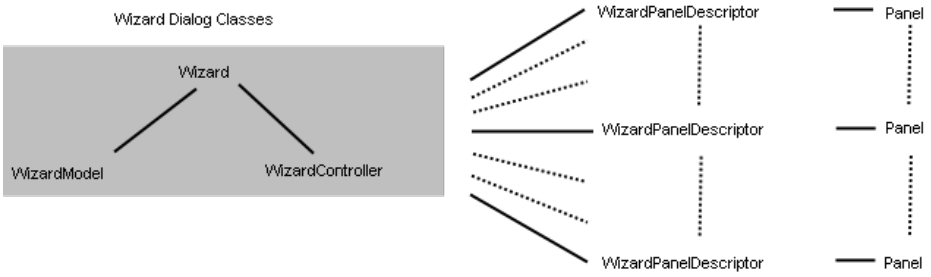

Abbildung 5: Diese Abbildung zeigt den allgemeinen Aufbau des Wizards

Neben den für den Wizard benötigten Klassen, existieren noch 2 weitere wichtige Klassen (Abb. 6). Eine zentrale Rolle in unserem Tool nimmt dabei die Klasse ORE ein. Sie ist eine Art Schnittstelle zwischen dem für die Darstellung und Interaktion verwendeten Wizard und der Wissensbasis. In ORE werden Lernalgoritmus initialisiert, Eingaben des Wizards bearbeitet und mögliche Ergebnisse an den Wizard zur Darstellung zurückgegeben.
Zum Arbeiten auf der Wissensbasis verwendet ORE die in der Hilfsklasse OntologyModifier verfügbaren Methoden, welche mit Hilfe der OWL-API implementiert wurden.

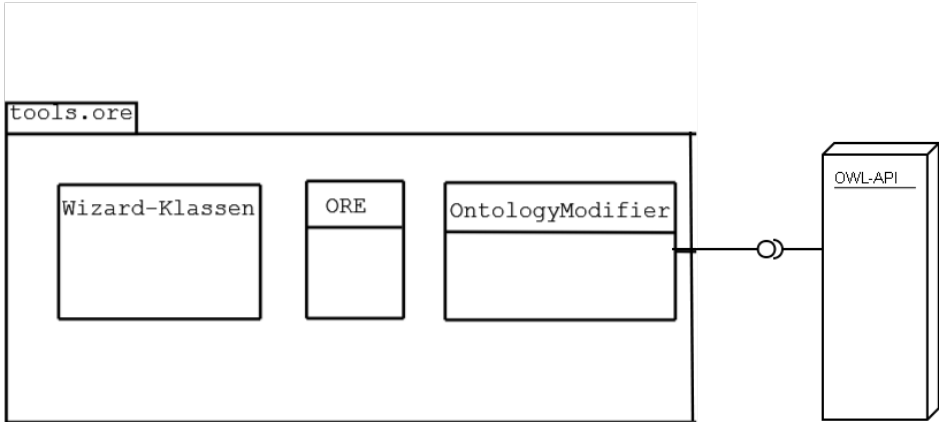

Abbildung 6: Diese Abbildung zeigt den allgemeinen Aufbau des Tools ORE

4.3 Prozessablauf

(a) Auswahl einer Klasse für die eine Definition (neu) gelernt werden soll

- Auswahl eines Fragments der Wissensbasis mit der DL-Learner SPARQL-Komponente bei großen Ontologien wie Open Cyc und DBpedia (entfällt bei kleinen Ontologien)

(b) Instanzen dieser Klasse werden als positive Beispiele gewählt, Nicht-Instanzen als negative Beispiele, weitere negative Beispiele werden zufällig gewählt

(c) Start des Lernalgorithmus: Resultate sind Vorschläge für Klassendefinitionen

(d) Nutzer akzeptiert einen der Vorschläge

(e) Axiom wird zur Wissensbasis hinzugefügt (Ontologie Erweiterung)

(f) feststellen welche Instanzen problematisch sind und anzeigen

(g) entsprechende Schritte einleiten (s. Kap. 3) , die der Nutzer steuern kann (Ontologie Reparatur)

5 ORE-Benutzerschnittstelle

In diesem Kapitel beschreiben wir die im vorhergehenden Kapitel erwähnten Schritte die beim Arbeiten mit dem Tool durchlaufen werden. Wir erläutern zuerst die allgemeine Darstellungsform des Wizards, bevor wir im Anschluss daran auf die einzelnen Schritte eingehen.
Allgemein ist der Wizard in 3 Teile geteilt (Abb. 7).
Im linken Teil (Abb. 7, 1) stehen alle Schritte die im Wizard durchlaufen werden mit ihrem Namen, wobei der aktuelle Abschnitt immer textuell hervorgehoben ist. Der untere Teil (Abb. 7, 3) umfasst die Navigation durch den Wizard. Es existieren hier 3 Buttons: ein Back- und ein Next-Button mit denen man jeweils vor- und zurück navigieren kann und ein Cancel-Button, mit dem man das Programm beenden kann.
Der Hauptteil des Wizards (Abb. 7, 2) umfasst die für den jeweiligen Schritt notwendigen Panels, die als Container für grafische Eingabe- und Ausgabeelemente dienen.
Durch die zwei unterschiedlich zur Verfügung gestellten Arten eine Wissensbasis einzulesen, unterscheiden sich auch die Dialoge in ihrer Darstellung. Außerdem sind die Schritte 5 und 6 zur Zeit nur für OWL-Dateien verfügbar.

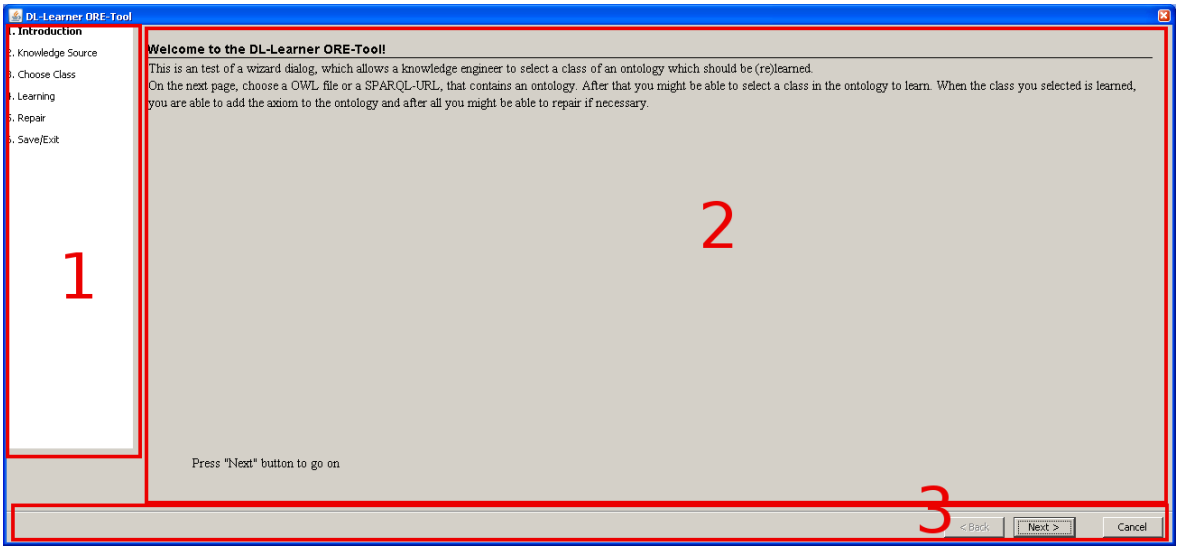

Abbildung 7: Diese Abbildung zeigt den grafischen Aufbau des Wizards

Schritt 1: Einführungs Dialog

Der Einführungsdialog gibt lediglich eine kurze Information über das Tool und die darauffolgenden Schritte, weshalb es hier nur der Vollständigkeit halber erwähnt wird.

Schritt 2: Auswählen der Wissensquelle

Der zweite Dialog ermöglicht die Auswahl einer Wissensquelle (Abb. 8). Da für die folgenden Dialoge eine Wissensbasis notwendig ist, ist der Next-Button zu Beginn gesperrt, d.h. ein Maus-Klick auf diesen bleibt ohne Wirkung.
Man kann in diesem Dialog zwischen 2 Arten von Wissensquellen wählen: einer OWL-Datei und einem SPARQL-Endpoint. Je nachdem was man ausgewählt hat, wird die jeweils andere Eingabe dann blockiert, um nur genau einen Typ von Wissensquelle zu ermöglichen. Wählt man OWL-File als Typ, so aktiviert sich ein Eingabefeld, in welches man den Pfad angeben kann, an dem sich die Datei befindet. Desweiteren besteht die Möglichkeit durch betätigen des Browse-Button, mit einer grafischen Darstellung durch die Verzeichnisstruktur zu navigieren und die gewünschte Datei auszuwählen. Sobald eine existierende Datei mit der Endung OWL ausgewählt wurde, aktiviert sich der Next-Button und man kann zum nächsten Dialog gehen. Wählt man hingegen den Typ SPARQL, so kann man im darunter liegenden Eingabefeld die URL eingeben, an der man eine SPARQL-Anfrage ausführen möchte. Auch hier aktiviert sich der Next-Button erst wenn eine erfolgreiche Verbindung zu der URL aufgebaut werden konnte.

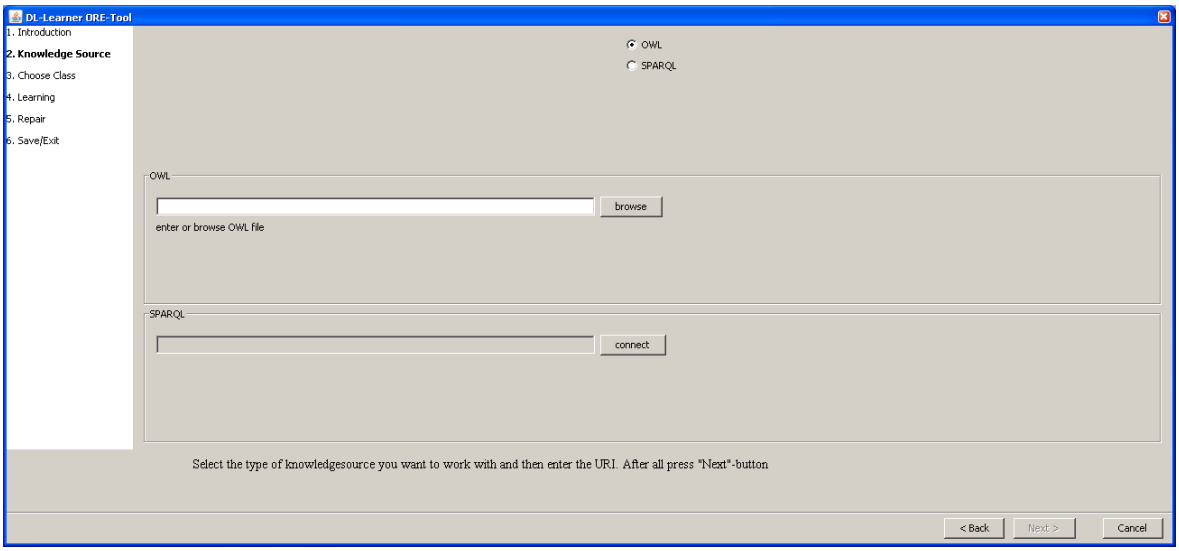

Abbildung 8: Diese Abbildung zeigt den Dialog zur Auswahl der Wissensquelle

Schritt 3: Auswählen der zu lernenden Klasse

In diesem Dialog werden, nach Auswahl einer Wissensbasis vom Typ OWL im vorangegangenen Dialog, alle atomaren Klasse aufgelistet, die sich in der Wissensbasis befinden. Man hat die Möglichkeit eine Klasse auszuwählen, die neu gelernt werden soll. Auch in diesem Dialog ist der Next-Button solange deaktiviert, bis eine Klasse gewählt ist. Sobald eine Klasse ausgewählt wurde, aktiviert sich der Next-Button.
Wurde SPARQL als Typ der Wissensbasis ausgewählt, so wird keine Liste angezeigt, sondern es muss in einem Eingabefeld die Klasse, die neu gelernt werden soll, manuell eingegeben werden.

Abbildung 9: Diese Abbildung zeigt den Dialog zur Auswahl einer Klasse

Schritt 4: Lernen einer neuen Klassenbeschreibung

Der Lerndialog besteht aus einer Liste, 2 Buttons und einem Schieberegler. Der Start-Button startet den Lernalgorithmus. Mit dem Stop-Button kann man den Algorithmus manuell beenden, solange dieser läuft. Mit dem Noise-Schieberegler kann man das Rauschen einstellen, d.h. man gibt an welche Toleranz die Genauigkeit der Ergebnisse haben kann. Setzt man den Schieberegler z.B. auf 20% , so terminiert der Algoritmus sobald ein Ergebnis verfügbar ist, das eine Genauigkeit von mindestens 80% vorweist. In der Liste werden die Ergebnisse angezeigt, die nach dem Starten des Lernalgorithmus aktuell verfügbar sind. Solange der Algorithmus läuft, wird die Liste periodisch mit den aktuell besten Ergebnissen aktualisiert.
Sobald der Algorithmus beendet ist, kann man aus der Liste eine Klassendefintion auswählen, die der Wissensbasis hinzugefügt werden soll.

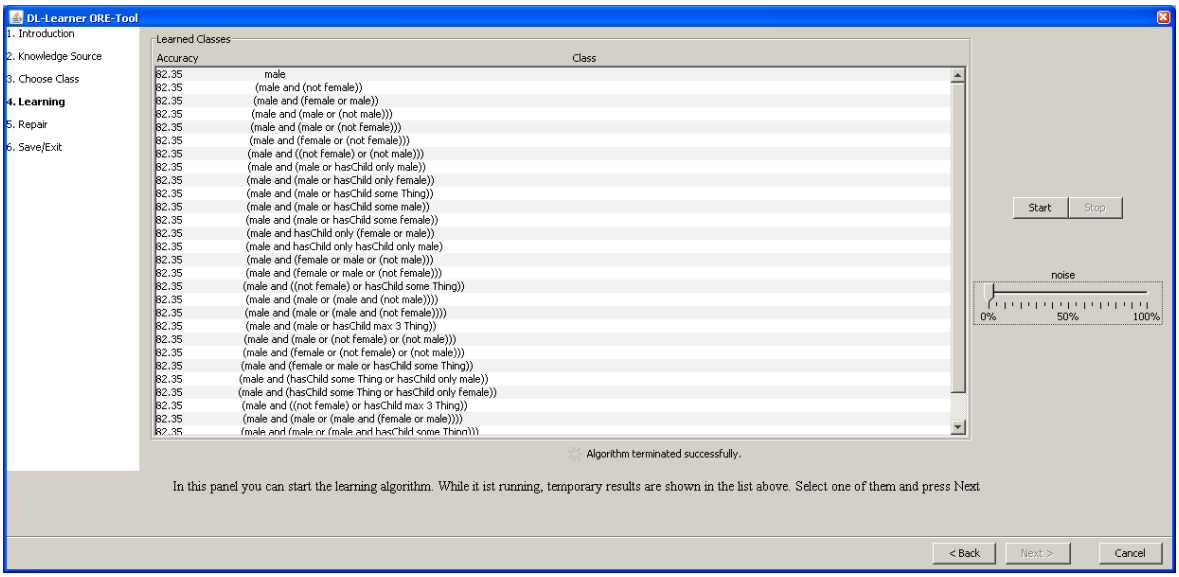

Abbildung 10: Diese Abbildung zeigt den Dialog zum Starten des Lernalgorithmus

Schritt 5: Reparieren fehlerhafter Instanzen

Im Reparaturdialog (Abb. 11) werden die fehlerhaften postiven und negativen Beispiele aufgelistet, die durch das Hinzufügen der ausgewählten Klassenbeschreibung im vorangegangenen Lerndialog zu Inkonsistenz führen. Es wird dabei je eine Liste für positive und eine für negative Beispiele dargestellt. Weil sich die Reparaturmöglichkeiten unterscheiden sind für beide Arten von Fehlern unterschiedliche Aktionen vorhanden. Wählt man ein positives Beispiel aus und betätigt den links neben der Liste stehenden Remove-Button, wird die Klassenzuordnung zu der zu lernenden Klasse entfernt (falls eine Zuordnung explizit in der Ontologie vorkommt). Wählt man in der rechten Liste ein negatives Beispiel aus, kann man es mit dem rechts neben der Liste stehenden Add-Button zu der zu lernenden Klasse hinzufügen. Beide genannten Aktionen führen außerdem dazu, dass das markierte Beispiel aus der jeweiligen Liste entfernt wird. Des weiteren verfügen beide Listen über je einen Delete- und Repair-Button. Ein Betätigen des Delete-Buttons entfernt alle Axiome der Ontologie, in denen das ausgewählte Beispiel vorkommt. Der Repair-Button öffnet ein neues Fenster (Abb. 12), in dem zu den Instanzen neue Information hinzugefügt werden können.

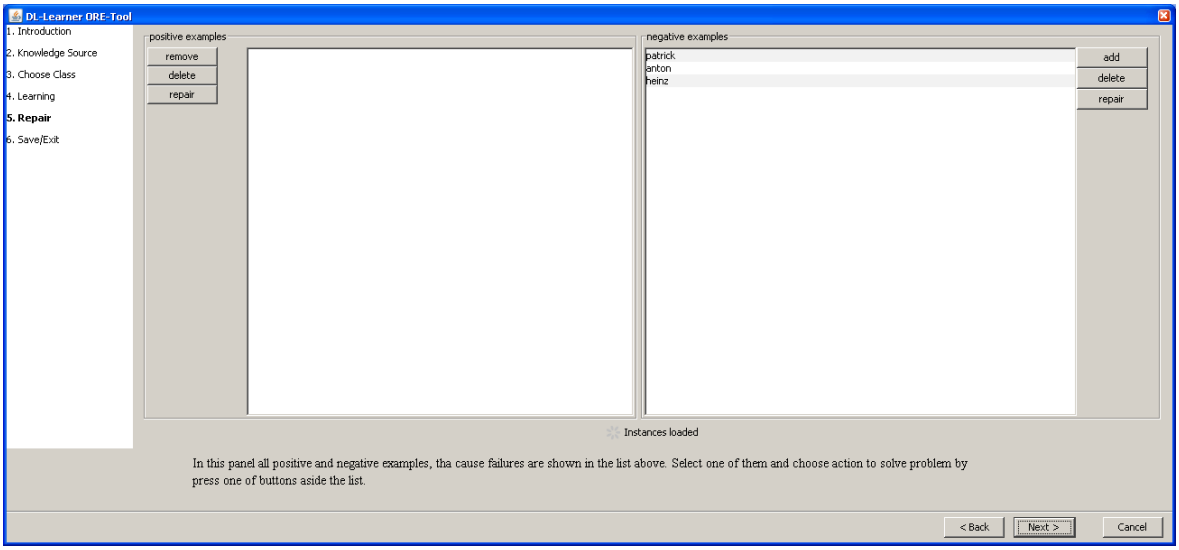

Abbildung 11: Diese Abbildung zeigt den Dialog zur Reparatur fehlerhafter Beispiele

Dieses Fenster besteht aus 3 Teilbereichen. Der obere Bereich (Abb. 12, 1) enthält die gelernte Klassenbeschreibung, wobei fehlerhaften Teilaxiome jeweils mit roter Schrift dargestellt werden. Klickt man mit der Maus auf einen rot markierten Bereich, öffnet sich ein Menü mit Lösungsvorschlägen. Der mittlere Bereich (Abb. 12, 2) beinhaltet Informationen über das ausgewählte Beispiel. Es werden Namen, Klassenzuordnungen und Object Properties aufgelistet. Im unteren Bereich (Abb. 12, 3) werden die Aktionen dargestellt, welche ausgeführt wurden um die Instanz zu reparieren. Zu jeder ausgeführten Aktion steht außerdem eine Undo-Funktion zur Verfügung, die die Aktion rückgängig macht.

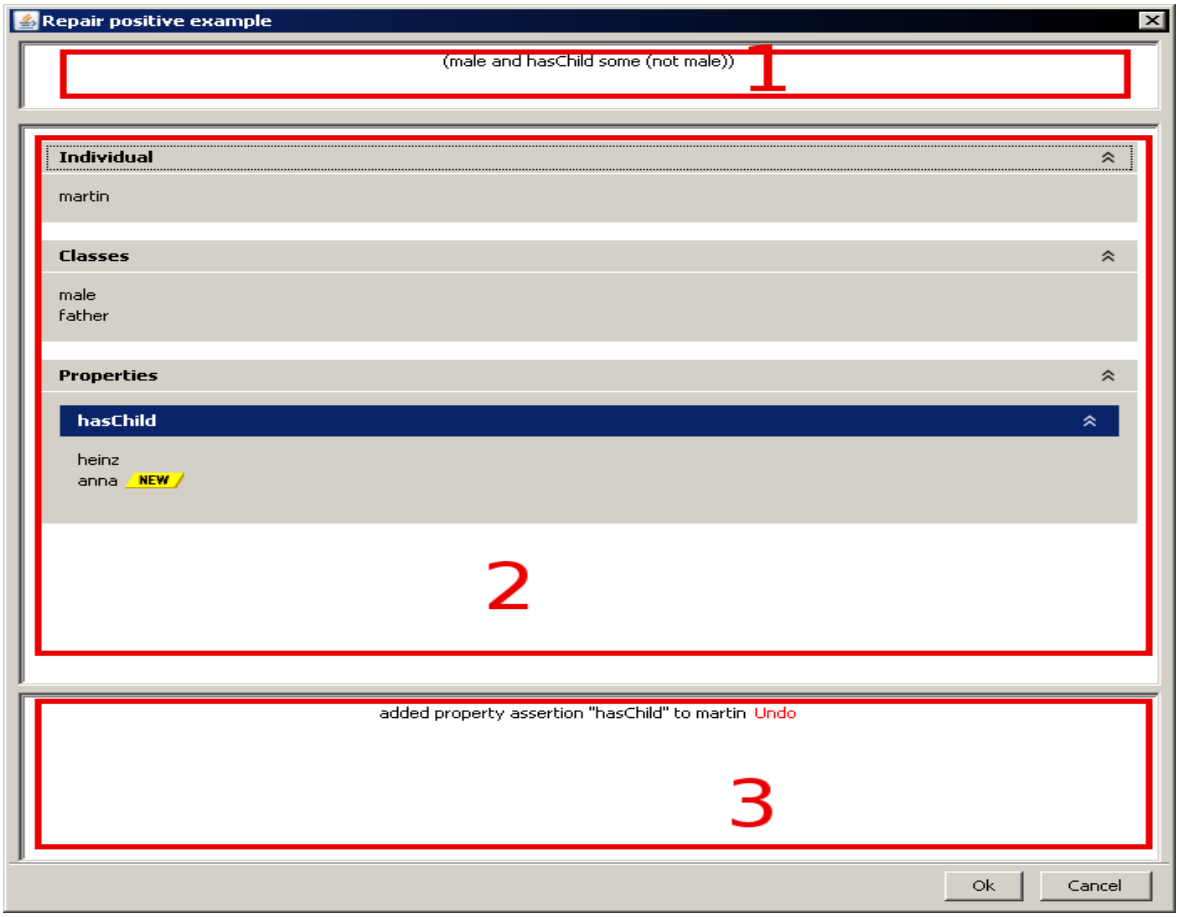

Abbildung 12: Diese Abbildung zeigt das Reparatur Menü

Möchte man dieses Fenster schließen und zur vorherigen Ansicht zurückkehren, besteht zum einen die Möglichkeit mit dem Ok-Button alle Änderungen zu übernehmen, oder mit dem Cancel-Button diese rückgängig zu machen. Wurde die Instanz bezüglich aller in der Klassenbeschreibung angezeigten Fehler repariert, wird auch hier das Beispiel aus der jeweiligen Liste entfernt.

Schritt 6: Speichern und beenden/neue Klasse lernen

In diesem Dialog (Abb. 13) besteht die Möglichkeit alle vorgenommenen Änderungen an der Ontologie zu speichern, und entweder das Programm zu beenden oder eine weitere Klasse zum Lernen auszuwählen.

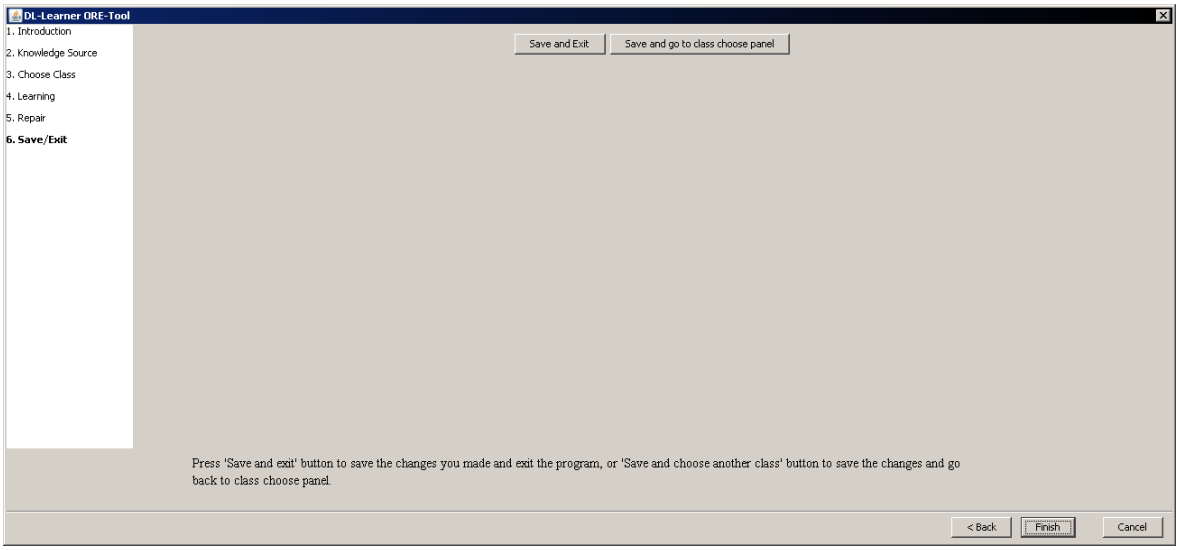

Abbildung 13: Diese Abbildung zeigt den Dialog zum Speichern und Beenden

6 Beispiel einer Reparatur

In diesem Kapitel wollen wir die Fähigkeiten von ORE an einem Beispiel demonstrieren. Wir werden anhand einer Beispiel-Ontologie (Abb. 14) zwei Klassen neu lernen lassen und aus den vorgeschlagenen Resultaten des Lernalgorithmus je eines auswählen. Im Anschluss daran werden wir die fehlerhaften Beispiele nennen, und für einige dann die Lösungsvorschläge unseres Programm auflisten.

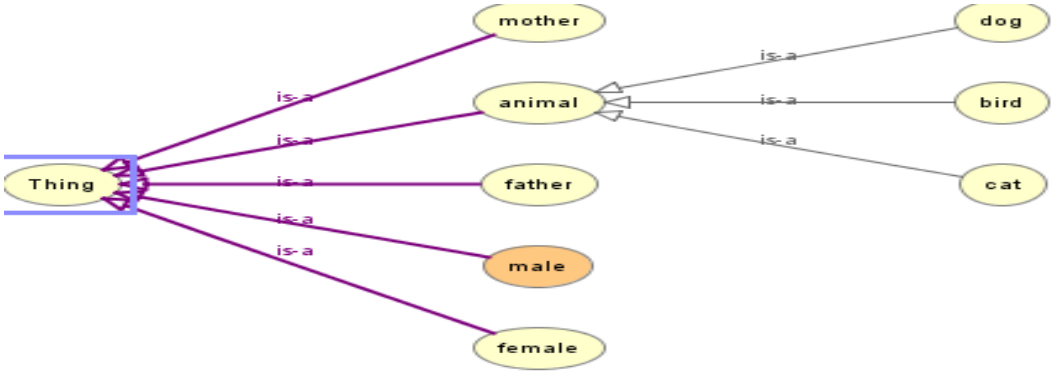

Abbildung 14: Ontologie Beispiel, erstellt mit Protégé

Die Ontologie besitzt folgende Instanzzuordnungen:

- $animal$: $carlo$, $minka$, $aiko$, $bigfoot$
- cat: $minka$
- dog: $aiko$, $bigfoot$
- $bird$: $carlo$
- $male$: $heinz$, $markus$, $maik$, $martin$, $nico$, $stefan$, $thomas$
- $female$: $nadine$, $nina$, $anna$, $bobby$, $katrin$, $michelle$
- $mother$: $anna$, $katrin$, $nadine$
- $father$: $anton$, $bobby$, $maik$, $markus$, $martin$, $stefan$

Weitere Axiome in Ontologie:

- $male$ äquivalent zu $\neg female$
- $father$ und $animal$ disjunkt
- $mother$ und $animal$ disjunkt

Ausgewählte zu lernende Klasse: $father$

Automatische gewählte positive Beispiele:

- $anton$, $bobby$, $maik$, $markus$, $martin$, $stefan$

Automatische gewählte negative Beispiele:

- $anna$, $katrin$, $nadine$, $aiko$, $bigfoot$, $minka$, $carlo$, $nina$, $michelle$, $heinz$, $nico$, $thomas$

Wir wählen zum Lernen einen Toleranzwert (noise) von 20% aus, das bedeutet der Algorithmus terminiert bei Ergebnissen von min. 80 % Genauigkeit.

gewählte Klassenbeschreibung mit einer Genauigkeit von 88,89%:

- $male\, and\, (not\, animal)\, and\, (not\, dog)\, and\, hasChild\, some\, Thing$

Als fehlerhafte positive Beispiele werden angezeigt:

- $anton$, $bobby$

Als fehlerhafte negative Beispiele werden angezeigt:

-

Angezeigte Ursachen für Fehler bei Instanz $anton$:

(a) keine Zuordnung zur Klasse $male$

Reparaturmöglichkeiten für $anton$:

- Löschen der ursprünglichen Instanzzuordnung zu $father$
- vollständiges Löschen der Instanz in der Ontologie
- die fehlerhaften Informationen bearbeiten
  (a) Instanz der Klasse $male$ zuordnen

Angezeigte Ursachen für Fehler bei Instanz $bobby$:

(a) keine Zuordnung zur Klasse $male$

Reparaturmöglichkeiten für $bobby$:

- Löschen der ursprünglichen Instanzzuordnung zu $father$
- vollständiges Löschen der Instanz in der Ontologie
- die fehlerhaften Informationen bearbeiten
  (a) verschieben der Instanz von $female$ zu $male$

Ausgewählte zu lernende Klasse: $mother$

Automatische gewählte positive Beispiele:

- $anna$, $katrin$, $nadine$

Automatische gewählte negative Beispiele:

- $aiko$, $bigfoot$, $minka$, $carlo$, $nina$, $michelle$, $heinz$, $nico$, $thomas$, $anton$, $bobby$, $maik$, $markus$, $martin$, $stefan$

Wir wählen zum Lernen einen Toleranzwert (noise) von 20% aus, das bedeutet der Algorithmus terminiert bei Ergebnissen von min. 80 % Genauigkeit.

gewählte Klassenbeschreibung mit einer Genauigkeit von 88,89%:

- $female\,and\,hasChild\,some\,Thing$

Als fehlerhafte positive Beispiele werden angezeigt:

- $katrin$

Als fehlerhafte negative Beispiele werden angezeigt:

- $bobby$

Angezeigte Ursachen für Fehler bei Instanz $katrin$:

(a) keine Property Assertion $hasChild\,some\,Thing$ vorhanden

Reparaturmöglichkeiten für $katrin$:

- Löschen der ursprünglichen Instanzzuordnung zu $mother$
- vollständiges Löschen der Instanz in der Ontologie
- die fehlerhaften Informationen bearbeiten
  (a) Property Assertion $hasChild$ hinzufügen mit möglichen Objekten:
    - $aiko$, $bigfoot$, $minka$, $carlo$, $nina$, $michelle$, $heinz$, $nico$, $thomas$, $anton$, $bobby$, $maik$, $markus$, $martin$, $stefan$, $anna$, $nadine$

Angezeigte Ursachen für Fehler bei Instanz $bobby$:

(a) Property Assertion $hasChild\,some\,Thing$ vorhanden

(b) der Klasse $female$ zugeordnet

Reparaturmöglichkeiten für $bobby$:

- Löschen der ursprünglichen Instanzzuordnung zu $father$
- vollständiges Löschen der Instanz in der Ontologie
- die fehlerhaften Informationen bearbeiten
  (a) alle Property Assertions $hasChild$ entfernen
  (b) i. Zuordnung zur Klasse $female$ entfernen
  ii. verschieben der Instanz von $female$ zu folgenden möglichen Klassen:
  – $male$

7 Zusammenfassung

Das Semantic Web kombiniert Methoden der Wissensrepräsentation mit Techniken und Sprachen des WWW. Diese vom World Wide Web Consortium geförderte Initiative derzeit ein sehr aktives Forschungsgebiet. Als Grundlage der formalen Wissensrepräsentation gelten dabei Ontologie, ein Konzept zur formalen Spezifikation von Wissen. Weil Ontologien mit wachsender Größe auch an Komplexität zunehmen, sind sie nur noch in einem bestimmten Maße manuell wartbar.

Wir haben in unserer Arbeit ein Tool entwickelt, dass den Nutzer von Ontologien dabei unterstützt, in dem es mit Hilfe von Machine Learning Verfahren des DL-Learners vollständigere Klassenbeschreibungen lernt und zur Wissensbasis hinzufügt. Um ein möglichst hohes Maß an Konsistenz in der Ontologie zu wahren, haben wir außerdem einen einfachen Reparaturmodus in das Tool integriert, der dem Nutzer Lösungsvorschläge anbietet, um mögliche Fehler zu entfernen.

Unser Tool deckt bisher nur einen kleinen Teil von Fehlern in Ontologien ab, weshalb zukünftige Arbeiten geplant sind. In diesen sollen alle möglichen Fehler behandelt werden können, wobei eine neues Gesamtkonzept erstellt werden muss.

Literatur

[1] F. Esposito, N. Fanizzi, L. Iannone, I. Palmisano, and G. Semeraro. A counterfactual-based learning algorithm for description logic. In *AI*IA*, pages 406–417, 2005.

[2] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[3] P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)*, volume 3729 of *LNCS*, pages 353–367. Springer, NOV 2005.

[4] P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure. *Semantic Web — Grundlagen*. eXamen.press. Springer, Berlin–Heidelberg, Germany, 2008. In German.

[5] J. Lehmann. Hybrid learning of ontology classes. In *Proceedings of the 5th International Conference on Machine Learning and Data Mining (MLDM)*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.

[6] J. Lehmann and P. Hitzler. Foundations of refinement operators for description logics. In H. Blockeel, J. W. Shavlik, and P. Tadepalli, editors, *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2008.

[7] J. Lehmann and P. Hitzler. A refinement operator based learning algorithm for the alc description logic. In H. Blockeel, J. W. Shavlik, and P. Tadepalli, editors, *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2008.

[8] B. Parsia, E. Sirin, and A. Kalyanpur. Debugging owl ontologies. In *Proc. 14th Int?l Conf. World Wide Web*, pages 633–640. ACM Press, 2005.

[9] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, Berlin–Heidelberg, Germany, 2004.

Ich möchte mich hiermit noch bei 2 Menschen bedanken, die mich während der Arbeit unterstützt haben. Das ist einmal Sebastian, der öfters hilfreiche Hinweise gegeben hat. Und ganz besonders Jens, der bis zum Ende immer Zeit für Fragen und Probleme hatte, egal wie kompliziert sie waren oder ich sie gemacht habe.

„Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann“.

Ort Datum Unterschrift