Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Gewinnung von probabilistischem Wissen aus ABoxen

04.02.2008

Studienarbeit im Studiengang Informatik-Ingenieurwesen

von

Phillip Neumann

Matrikelnr.: 22122

betreut durch

Prof. Dr. Ralf Möller

Dipl. Ing. Tobias H. Näth

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Ein Mensch filtert aus Daten die für ihn relevanten Information eigenständig und mit Hilfe persönlich festgelegter Schemata. Das Filtern der Information wird dabei durch das persönliche bzw. professionelle Interesse und die eigenen Fähigkeiten des Individuums bestimmt. Maschinen sind zu derartigen Filterungen (noch) nicht in der Lage.
Um die im Internet enthaltenen Daten auch für Maschinen lesbar zu machen, wird daran gearbeitet, das World Wide Web (WWW) um das Semantische Web zu erweitern. Diese Erweiterung hat zum Ziel, die Semantik von im Internet vorhandenen Daten, wie z.B. Bildern und Texten zusätzlich zu diesen Objekten zu hinterlegen und so für eine maschinelle Erfassung, Interpretation und Verarbeitung zur Verfügung zu stellen.
Anwendung könnte diese Technik z.B. in Suchmaschinen finden, die durch die zusätzlichen semantischen Information zu den Objekten, inhaltlich stärker bezogene "Treffer" ermitteln könnten, als Suchmaschinen, die Informationen ausschließlich über Schlüsselwörter und deren Häufigkeit ermitteln.
Eine zusätzliche Gewichtung von Informationen durch Zuordnung von Auftrittswahrscheinlichkeiten kann beispielsweise im Falle von Suchanfragen zu inhaltlich präziseren Antworten führen.

1.2 Aufgabenstellung

Ziel dieser Studienarbeit ist es, ein Programm zu entwerfen, welches frequenzabhängiges Auftreten von Individuen in ABoxen erfasst und daraus eine Abschätzung für die untere Grenzwahrscheinlichkeit eines Conditional Constraints bestimmt. Die Conditional Constraints sollen dazu aus einem selbstentworfenen Format einer PTBox ausgelesen, die Auftrittswahrscheinlichkeiten der Constraints bezüglich einer Menge von ABoxen ermittelt und anschließend in der PTBox gespeichert werden.

2 Theorie/Grundlagen

2.1 Beschreibungslogik

Wie schon in 1.2 erwähnt werden im semantischen Web zusätzlich zu den Objekten Informationen hinterlegt. Diese Informationen sollen für die maschinelle Verarbeitung aufbereitet werden und eine zusammenfassende Beschreibung der Objekte darstellen.
Beschreibungslogiken sind logische Systeme, die in der Lage sind in einem Wissensbereich Formalisierungen von Aussagen durchzuführen und ihre Gültigkeit zu überprüfen. Diese Art der Logiken, die auf Prädikatenlogiken abgebildet werden können, sind Logiken der ersten Stufe. Das heisst, daß die Aussagenlogiken um parametrisierte Aussagen über den Interpretationsbereich erweitert sind. Anstelle von Aussagenvariablen (a,b) kommen Aussagen (p(x), q(x,y) etc.) zum Einsatz, wobei nur über die Objektvariablen (x,y) unter Verwendung von Quantoren ($\forall$ "für-alle", $\exists$ "es-gibt" und $n\exists$ "für-einige") quantifiziert werden kann [Fischer Kompakt].
Mit diesen Eigenschaften ist es möglich über eine Beschreibungslogik Objekte zu schliessen, mit anderen Worten, aus vorhandenem Wissen impliziertes zusätzliches Wissen abzuleiten.
Man unterteilt die Wissensbasis einer Beschreibungslogik formal in eine Terminological Box und eine Assertional Box, die in dem folgenden Kapitel näher erläutert werden.

2.1.1 Terminological Box (TBox)

Am Anfang einer Wissensbasis steht immer die Aufgabe, die Gesamtheit aller Begriffe und Benennungen, die Terminologie, festzulegen. Dies geschieht, indem zuerst alle auftretenden Objekte, die so genannten Konzepte, in "Terminologischen Boxen" (TBoxen) deklariert werden. Die grundlegende Form der Deklaration in einer TBox ist die Definition eines Konzepts, basierend auf zuvor definierten Konzepten [DLHB-2003-TBOX].

$$Woman \equiv Person \sqcap Female$$

Durch die Definition der Konzepte ist es möglich, Beziehungen (Relationen) zwischen Konzepten herzustellen. Die Darstellung von Relationen geschieht hauptsächlich in der "Assertional Box" (siehe 2.1.2). Grundlegende Beziehungen zur Defintion von Konzepten können aber schon vorher in der Tbox

festgelegt werden.
Das Konzept $PoleVault$ wird im folgenden Beispiel durch die Relationen $hasPaticipant$ und $hasPart$ definiert. Um eine zusätzliche Aussage über die Menge der Relationen von $Person$ und $Pole$ auf $PoleVault$ zu treffen, werden die in 2.1 genannten Quantoren eingesetzt. In diesem Falle der Quantor $\exists$"es-gibt-eine".

$$PoleVault \equiv \exists hasParticipant.Person \sqcap \exists hasPart.Bar \sqcap \exists hasPart.Pole$$

Des Weiteren gibt es die Möglichkeit, ein Konzept über eine Subset-Beziehung darzustellen. Dabei ist ein Konzept eine Untermenge eines anderen Konzeptes, das als Obermenge aufritt.
$HighJump$ ist hierbei eine Untermenge von $Sportstrail$. Es ist nun möglich mit Hilfe der Subset-Beziehung Klassifizierungen durchzuführen. Hierbei werden implizierte Untermengenbeziehungen zwischen Konzepten erkannt.

$$HighJump \sqsubseteq Sportstrail$$

Mit Hilfe der oben beschriebenen Axiome werden alle Konzepte in TBoxen abgelegt. Dabei werden gewöhnlich folgende allgemeine Annahmen getätigt:

- Nur eine Definition pro Konzeptname ist erlaubt
- Definitionen sind azyklisch, d.h. sie sind weder durch sich selbst noch durch ein Konzept, welches von ihnen indirekt abhängt, beschrieben

2.1.2 Assertional Box (ABox)

Mit den in den TBoxen definierten Konzepten und Beziehungen können nun Aussagen über alle Individuen der Welt gemacht werden. Diese Dastellung erfolgt in "Assertional Box" (ABoxen), welche über den Wissensbereich sogenanntes erweitertes Wissen enthalten. Es handelt sich um erweiterte Aussagen über Individuen und deren Beziehungen untereinander, welche Zugehörigkeitsbehauptungen (membership assertions) genannt werden.
Die aus diesen Eigenschaften entspringende wesentliche Verwendung einer ABox ist das Überprüfen von Individuen im Zuge von Anfrageaufgaben. Dabei wird überprüft, ob ein Individuum von einem in der TBox definierten Konzept abstammt.
Weiter Anfrageaufgaben in der Individuenüberprüfung sind das Überprüfen der Wissensbasis (knowledge base consistency), das Umsetzen (realization)

und das Suchen (retrieval). Bei der Konsistenzprüfung der Wissensbasis wird abgefragt, ob jedes Konzept mindestens ein Individuum in der Wissensbasis zulässt. Für das Umsetzen hingegen findet die Überprüfung das spezifische Konzept, von dem ein Individuum abstammen kann. Beim Suchen werden alle Individuen einer Wissensbasis zu einem gegebenen Konzept gefunden [DLHB-2003-ABOX].

2.2 Probabilistische Beschreibungslogik

In der Darstellung von unsicherem Wissen ist es hilfreich, den in der Wissensbasis auftretenden Konzepten Auftrittswahrscheinlichkeiten zuzuordnen. Unsicheres Wissen kann dabei bedeuten, dass keine eindeutige Zuordnung eines Individuums zu einem Konzept möglich ist. Bei einer Anfrage auf eine Wissensbasis, die durch eine probabilistische Beschreibungslogik definiert ist, ist es möglich den Grad der Wahrscheinlichkeit darzustellen, dem ein Individuum einem angenommenen Konzept entspricht. In [Giugno,Lukasiewicz;2002] ist eine Erweiterung einer Beschreibungslogik um unsicheres Wissen beschrieben. Die Erweiterung wurde in [naeth07] analysiert und implementiert.

2.2.1 Probabilistic Terminological Box (PTBox)

Die Representation unsicheren Wissens auf Konzeptebene erfolgt in dem zuvor erwähnten Ansatz, in der sogenannten probabilistisch terminologischen Box (PTBox). Sie beinhaltet im Gegensatz zur einfachen TBox bedingte Beschränkungen, sogenannte Conditional Constraints. Conditional Constraints bestehen dabei formal sowohl aus zwei Konzepten, über die eine Aussage der bedingten Wahrscheinlichkeit getroffen wird, als auch aus einer unteren (lowerboundprobability) und einer oberen Grenze der Wahrscheinlichkeit (upperboundprobability).
Ein Conditional Constraint hat dabei folgende Zusammensetztung:

$$(C|E)[l,u]$$

"E" entspricht hierbei dem Beweis (Evidence), "C" der Schlußfolgerung (Conclusion), "l" der unteren und "u" der oberen Grenzwahrscheinlichkeit.
Ein Conditional Constraint stellt mit den Elementen, die es beinhaltet, ein Verhältnis zwischen den Konzepten dar. Dieses Verhältnis wird zusätzlich durch Wahrscheinlichkeiten gewichtet. Man kann sich das gewichtete Verhältnis als Annahme über die Wahrscheinlichkeit vorstellen, mit der sich die Men-

gen zweier Konzepte überschneiden.
Für das Beispiel $(HighJump \mid Sportstrail)[0.1,0.2]$, in dem $HighJump$ und $Sportstrail$ die Konzepte eines Conditional Constraints darstellen, wird über die Grenzwahrscheinlichkeiten ermittelt, wie weit sich die Konzepte überschneiden. Mit anderen Worten gesagt, wird durch das Conditional Constraint dargestellt, wie hoch der Anteil der $HighJumps$ an der Menge der $Sportstrails$ ist.
Bei einer Anfrage für die Erfüllbarkeit eines Conditional Constraints unter der Benutztung der PTBoxen, wird überprüft, ob es eine probabilistische Interpretation gibt, die innerhalb der unteren und oberen Grenzwahrscheinlichkeit liegt. Liegt die Wahrscheinlickeit einer Interpretation innerhalb der Grenzen, so ist das Conditional Constraint erfüllbar.

3 Methodisches Vorgehen

Das in diesem Projekt zu entwickelnde Programm soll für jedes in PTBoxen hinterlegte Conditional Constraint eine untere Grenzwahrscheinlichkeit bestimmen und das Conditional Constraint mit der ermittelten Wahrscheinlichkeit versehen. Zur Bestimmung der Wahrscheinlichkeit muss eine statistische Erhebung für Autritte von Instanzen, die in den Conditional Constraints enthalten sind, über eine Wissensbasis durchgeführt werden. Aus den ermittelten Informationen soll die untere Grenzwahrscheinlichkeit berechnet werden.

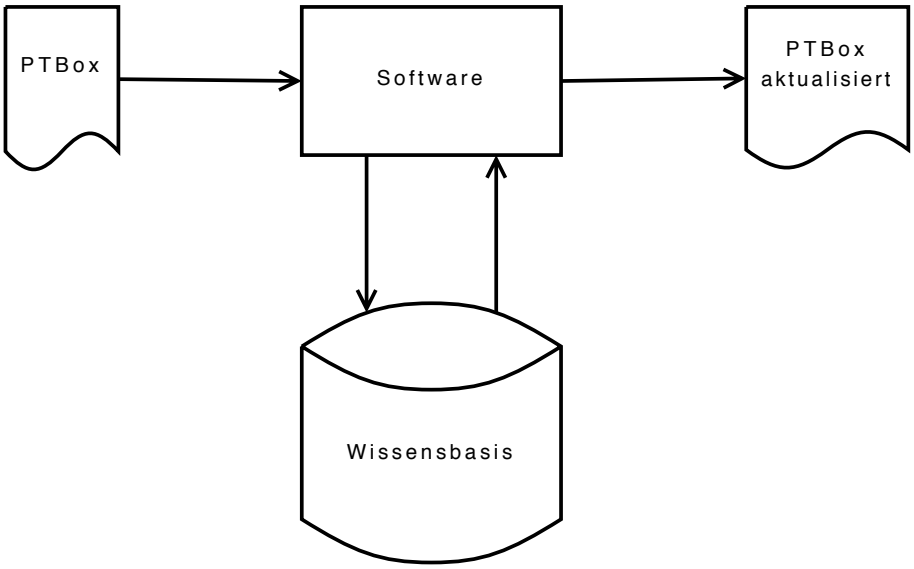

Abbildung 1: Modell des Szenarios

Für die Umsetzung der Aufgaben muss die Berechnung der Wahrscheinlichkeit, das Abfragen der Wissensbasis mit Hilfe eines semantischen Webservers und die Erzeugung von sinnvollen Abfragen festgelegt werden.

3.1 Abschätzung der Wahrscheinlichkeit

Um mit den Conditional Constraints Wissen modellieren zu können, müssen Werte für die untere und obere Grenzwahrscheinlichkeit ermittelt werden. Um dabei nicht von einer Bewertung durch eine Person, die Entscheidungen anhand von Intuition und Erfahrungen trifft, abhängig zu sein, soll ei-

ne statistische Abschätzung zur Bestimmung der Grenzwahrscheinlichkeiten durchgeführt werden. Diese Abschätzung wird über die gegebene Wissensbasis erstellt. Gezählt werden hierzu die Auftritte von Instanzen, die in die Schnittmenge von Evidence und Conclusion fallen und getrennt davon die Instanzen, die dem Konzept Evidence allein entsprechen, ermittelt.
Aus der Definition $l \leq \frac{Pr(E \sqcap C)}{Pr(E)} \leq u$ ergibt sich folgender Ansatz:

$$l = \frac{A}{B}$$

l = untere Grenzwahrscheinlichkeit,
A = $\sum Auftritte\ von\ Individuen$ die $Evidence \sqcap Conclusion$ erfüllen,
B = $\sum Auftritte\ der\ Individuen$ die das Konzept $Evidence$ erfüllen
Man errechnet somit eine Abschätzung für die untere Grenzwahrscheinlichkeit aus dem Verhältnis der Auftrittsmengen der Instanzen.
Die obere Grenzwahrscheinlichkeit "u" wird für dieses Projekt immer auf "1" gesetzt, da die Annahme getätigt wird, dass im maximalen Falle die Conclusion eine vollständige Untermenge der Evidence ist.

3.2 RacerPro©

Es ist also notwendig, Informationen über Individuen von Conclusions und Evidences aus den ABoxen zu extrahieren, ohne dabei selbst ein Programm zu entwickeln, das ABoxen verwaltet, angefragte Information aus eben diesen ABoxen selektiert und zur Verfügung stellt.
Eine Software zur Verwaltung von ABoxen ist RacerPro© von der Firma Racer Systems GmbH & Co. KG. Der RacerPro© ist ein semantischer Webserver, der darauf ausgerichtet wurde, Anforderungen zu erfüllen, die im Umgang mit OWL-Dokumenten und OWL-Wissensbasen (siehe 3.3) notwendig sind.
Dazu zählen unter anderem [RacerPro© Product Site]:

- Überprüfen einer OWL Ontologie und einer Reihe von Datenbeschreibungen auf Konsistenz
- Finden von indirekten Beziehungen, die durch die Deklaration in der Ontologie induziert werden.
- Finden von Synonymen für Quellen
- Verarbeiten und Beantworten von Queries

3.3 Web Ontology Language (OWL)

Wie in 1.1 bereits erwähnt, wird das Semantische Web als Erweiterung des WWW weiterentwickelt. Ziel der Erweiterung ist es, Informationen im Internet explizite Bedeutungen zuzuordnen, um die maschinelle Verarbeitung der Informationen zu ermöglichen. Mit Hilfe der Extensible Markup Language (XML) und dem Resource Description Framework (RDF) werden Daten dargestellt. Zuvor wird jedoch eine Ontologiesprache benötigt, die die Bedeutung einer Terminologie, welche in Webdokumenten benutzt wird, formal beschreibt. Wenn von Maschinen erwartet wird, sinnvolle Operationen auf diese Dokumente anzuwenden, muss die Sprache weitreichender als die Grundbedeutungen eines RDF-Schemas sein.
Die Web Ontology Language (OWL) wurde vom W3C Konsortium entwickelt, um diese Anforderungen zu erfüllen. OWL erweitert das Beschreibungsvokabular, um Eigenschaften und Klassen darzustellen. Unter anderem mit:

- Beziehungen zwischen Klassen (z.B Disjunktheit)
- Kardinalität (z.B "exakt eins")
- Gleichheit
- Umfangreichere Typisierung
- Besonderheiten von Eigenschaften (z.B. Symmetrie)
- Aufgezählte Klassen

OWL ist somit in der Lage alle Begriffe und Benennungen formal zu beschreiben. In OWL-Dokumenten werden Instanzen und ihre Beziehungen in ABoxen und Referenzen auf die entsprechenden Konzepte dargestellt [OWL].

3.4 Queries

Um Informationen über das in ABoxen gespeicherte Wissen zu erhalten, müssen Anfragen an den RacerPro© gestellt werden. Da es sich um eine Abfrage einer Datenbank oder eines datenbank-ähnlichen Systems handelt, wird diese Anfrage auch Query genannt.
Die Queries werden in der New RacerPro Query Language (nRQL) verfasst.

Mit nRQL kann der RacerPro© über ABoxen, TBoxen, RDF und OWL-Dokumente abgefragt werden. nRQL bietet außerdem die Möglichkeit Verbundanfragen, sogenannte conjunctive queries zu verfassen. Man kann vereinfacht sagen, dass nRQL eine mächtige Abfragesprache für ABoxen, RDF und OWL-Dokumente ist [adl04.pdf].
Eine in nRQL geschriebene Query besteht im Wesentlichen aus einem Query-head und einem Query-body. Der Query-head beinhaltet dabei die Variable, der das Anfrageergebnis zugeordnet wird. Im Query-body befinden sich nochmals die Variable und die Bedingung(en), die es zu erfüllen gilt. Dem Query-head wird zudem noch die Operation vorangestellt, die auf die Wissensbasis angewendet werden soll.
Im folgenden Bespiel ist es eine Retieve-Operation bei der eine mögliche Anfrage im Beispielcode gegeben ist:

(RETRIEVE (?X) (?X SPORTSTRAIL))

Suche alle Individuen, die eine Instanz des Konzeptes SPORTSTRAIL sind. Eine mögliche Antwort könnte folgende sein:

(((?X HIGHJUMP01)) ((?X POLEVAULT07)) ((?X SPRINT01)) ((?X HAMMERTHROW01)))

Der RacerPro© gibt in diesem Beispiel eine Liste von Paaren zurück, die jeweils aus der Variablen X und einem Individuum bestehen. Diese zurückgegebenen Individuen erfüllen vollständig die in der Query gestellten Bedingung. Um Anfragen mit mehreren Bedingungen zu stellen, gibt es Verbundanfragen, sogenannte Conjunctive Queries. Bei einer Conjunctive Query werden die Bedingungen der Anfrage mit dem “AND”-Operator verbunden und beziehen sich weiterhin auf eine Variable:

(RETRIEVE (?X) ((AND (?X HIGHJUMP)(?X SOME HASPARTICIPANT.PERSON))

Die Conjunctive Query fordert alle Instanzen, die beide Bedingungen erfüllen. Mit Hilfe der Queries ist es möglich, die für dieses Projekt notwendigen Informationen zur Auszählung von Individuen in ABoxen, die im späteren Verlauf genauer dargestellt werden, durchzuführen. [Racer User Guide Queries]

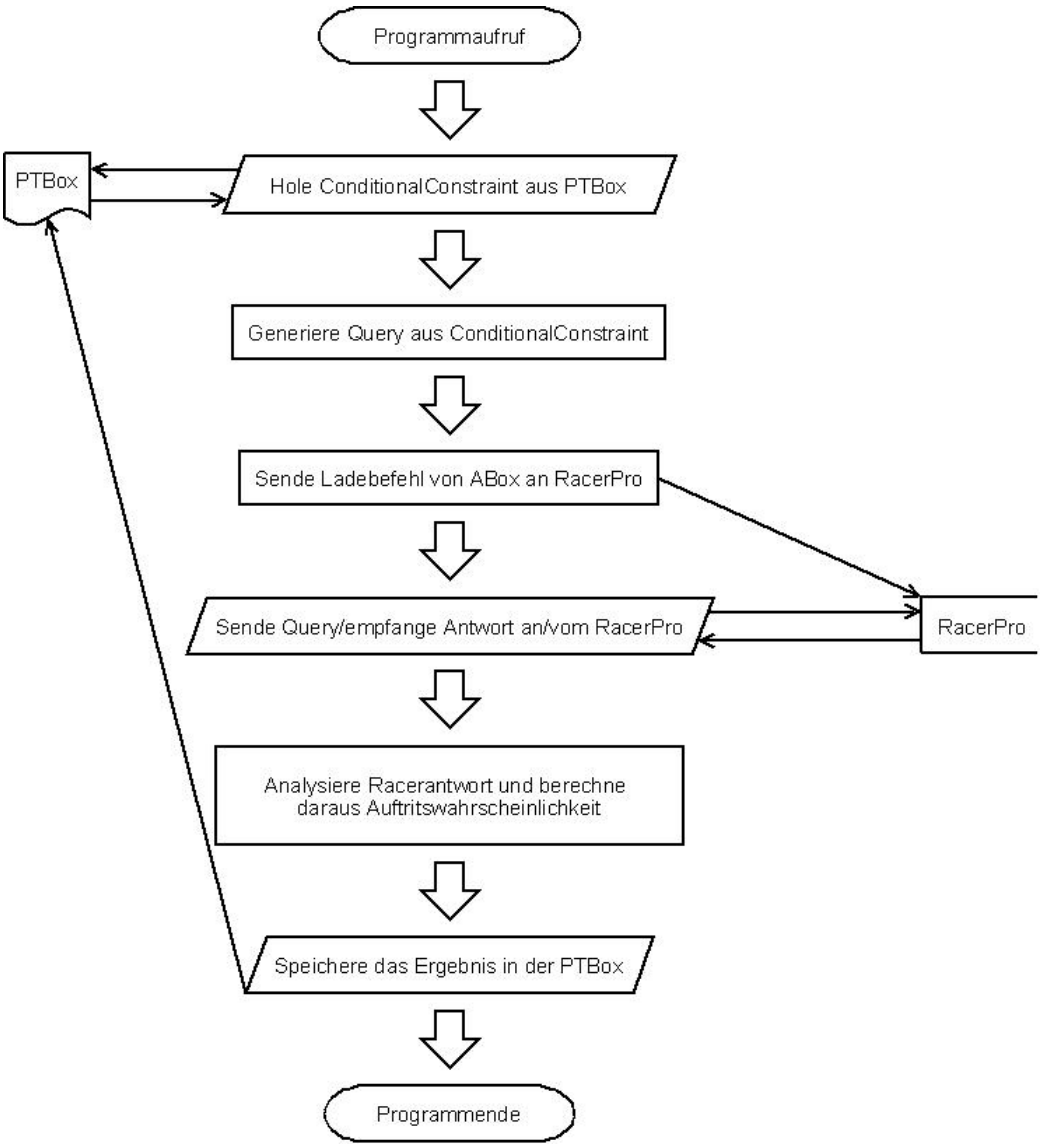

Abbildung 2: Programmablauf

4 Durchführung

Der Programmablauf, wie in Abbildung 2 zu sehen, beginnt mit der Beschaffung der PTBox aus dem PTBoxdokument. Sie enthält alle Conditional Constraints aus denen im nächsten Schritt Queries generiert werden müssen. Bevor aber Queries an den RacerPro© gesendet werden können, muss erst der Befehl zum Laden einer ABox gesendet werden, da die Suchanfrageoperation in diesem Projekt stets auf ABoxen angewendet wird. Die vom RacerPro© empfangenen Antworten werden danach ausgewertet und zur Berechnung der unteren Grenzwahrscheinlichkeit verwendet (siehe 3.1). Sobald alle Ergebnisse ermittelt und den Conditional Constraints zugeordnet sind, wird die aktualisierte PTBox im PTB-Dokument abgelegt.
Aus diesem Programmablauf ergeben sich folgende Aufgaben:

- Definition und Erstellung eines Dateiformats; Realisierung der dazugehörigen Schnittstelle für den Zugriff auf das Dokument
- Erzeugung von Queries
- Kommunikation mit dem RacerPro©
- Verarbeitung der RacerPro©-Antwort
- Berechnung der Auftrittswahrscheinlichkeiten
- Zuordung der Wahrscheinlichkeiten zu den entsprechenden Konzepten
- Speicherung in der PTBox

4.1 Dateiformat und Schnittstellen

Eine wesentliche Aufgabe in dieser Arbeit ist, die Funktionalität zu realisiern, um Daten zu extrahieren, auszuwerten und in einem Dokument, in diesem Falle der PTBox, ablegen zu können. Dazu ist es notwendig ein Dateiformat für das zu erstellende Dokument und eine Schnittstelle für den Dateizugriff zu entwickeln.

4.1.1 XML-Format

Bei der Wahl des Dokumentenformats besteht die Möglichkeit, Daten im XML-Format abzulegen. Da dieses Format in allen anderen OWL-Dokumenten verwendet wird, weil es unter anderem die Eigentschaft der Maschinenlesbarkeit besitzt, und es zudem mit den XMLBeans eine Technologie für einen komfortablen Zugriff auf XML-Dokumente gibt, fiel die Wahl auf die Kombination von XML-Format und XMLBeans.
Das XML-Format besitzt des Weiteren vorteilhafte Eigenschaften, wie:

- Strukturiertes und eindeutiges Darstellen von Daten
- HTML-ähnlicher Aufbau mit Tags (wobei die Tags im Gegensatz zu HTML nur dem Abgrenzen des Inhaltes dienen und die Verarbeitung und Interpretation des Inhaltes der Anwendung obliegt)
- Breite Unterstützung durch die "XML Familie" (stellen Module zur Arbeit mit XML zur Verfügung)
- Lizenzfreiheit
- Plattformunabhängigkeit

[XML - 10 Points] Mittels eines im XML-Format angelegten Schemas ist es möglich die Eigenschaften der auftretenden Elemente festzulegen. Dabei kann

1. die Zusammensetztung
2. die maximale Anzahl
3. der Typ
4. die Bezeichnung

eines jeden Elements per Definition festgelegt werden.
Das für diese Arbeit speziell erstellte XML-Schema hat folgende Gestalt:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
       <xs:element name="PTBox">
             <xs:complexType>
                    <xs:sequence>
```

```
                    <xs:element ref="PTBEntry" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="PTBEntry">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Evidence"/>
                    <xs:element ref="Conclusion"/>
                    <xs:element ref="LowerBound"/>
            <xs:element ref="UpperBound"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="LowerBound" type="xs:double"/>
        <xs:element name="UpperBound" type="xs:double"/>
        <xs:element name="Evidence" type="xs:string"/>
        <xs:element name="Conclusion" type="xs:string"/>
</xs:schema>
```

Das XML-Schema gibt dabei vor die PTBox besäße mehrere PTBoxEntries (Einträge). Die Einträge bestehen wiederum aus vier Elemeten.

- Evidence - Indiz
- Conclusion - Schlußfolgerung
- LowerBound - berechnete untere Auftrittswahrscheinlichkeit
- Upperbound - maximale Auftrittswahrscheinlichkeit

Evidence und Conclusion sind vom Typ “String”, LowerBound und Upperbound vom Typ “double” und dürfen, wie im Schema vorgegeben, nur einmal pro PTBoxeintrag erstellt werden. Die mit diesem Schema erzeugbare PTBox besitzt das notwendige Format, die Daten von Conditional Constraints strukturiert darzustellen und berücksichtigt dabei ebenfalls den in 2.2.1 beschriebenen Aufbau der Conditional Constraints.

4.1.2 XMLBeans

Um auf die PTBoxen zugreifen zu können, kommt die Technologie der XMLBeans zum Einsatz.

Der Vorteil in der Wahl der XMLBeans ist die Möglichkeit, für ein gegebenes XML-Schema ein Application Programming Interface (API) zu generieren. Diese API wird von den XMLBeans in der Form eines JAR-Files zu Verfügung gestellt und kann somit in jedes JAVA-Projekt eingebunden werden.
Die Haupteigenschaften der XMLBeans sind:

- Volle XML-Schema Unterstützung (das Schema kann als "Schablone" für alle zu erstellenden Dokumente benutzt werden)
- Volle XML-Infoset Genauigkeit (XML-Elemente können den Vorgaben des XML-Namensraumes unterworfen werden)

Bei der Nutzung dieser Eigenschaften kann die Integrität und Konsistenz für Dokumente und Elementenbezeichnungen gewahrt werden.
Eine durch die XMLBeans generierte API ist in der Lage Methoden und Klassen bereitzustellen, um Dokumente, welche dem gegebenen Schema folgen, zu erzeugen und zu speichern. Sie bietet weiterhin Methoden und Klassen an, um auf einzelne Elemente des Dokuments zuzugreifen. Das bedeutet, dass Inhalte aus Elementen extrahiert und auch manipuliert werden können. Die Methoden und Klassen werden auf die im Schema deklarierten Elemente Bezug nehmend benannt. Dadurch bietet die API einen namentlich logischen, an das Schema angelehnten Aufbau der Werkzeuge für den Zugriff auf Dokumentenelemente und deren Inhalt.
Die für dieses Projekt erzeugte API stellt folgende Klassen und Methoden zur Anwendung bereit:

PTBoxDocument	PTBoxDocument.PTBox
getPTBox() save(String)	getPTBEntryArray(int) setPTBEntryArray(int,string) addNewPTBEntry()

PTBEntryDocument.PTBEntry
getPTBox() save(String) SetPTBEntry()

Die zum Auslesen von Evidence oder Conclusion aus einem PTBoxeintrag generierten Methoden sind "getEvidence()" bzw. "getConclusion()". Analog dazu werden die Methoden "setEvidence()" und "setConclusion()" zur

Speicherung von Inhalt in Evidence und Conclusion zu Verfügung gestellt. Diese Gestaltung der Namensgebung von Methoden wird auf alle im Schema vorkommenden Elemente angewendet.

4.1.3 PTBoxHandler Klasse

Die durch die XMLBeans für das gewählte XML-Schema erzeugte API bietet, wie in 4.1.2 beschrieben, die Funktionalität auf XML-Dokumente und ihren Inhalt zuzugreifen und zu manipulieren. Für die Anforderungen, die sich im Zuge der Entwicklung des Projektes ergaben, erschien es von Vorteil, den Funktionsumfang der API in einer Klasse noch weiter zusammenzufassen.
Der vereinfachte Umgang mit der API wird durch die Entwicklung der PTBoxHandler Klasse realisiert. Der PTBoxHandler übernimmt das Initialsieren der API und bekommt über die Methode SetFilePath(String File) den Pfad zum PTBoxdokument als Parameter übergeben. Mit der Methode GetPTBox() übergibt der PTBoxhandler ein Objekt des Typs PTBoxDocument.PTBox an das Hauptprogramm. Das PTBox Objekt wird beim Initialisieren des PTBoxHandlers aus den Informationen, die in dem PTBoxDokument vorhanden sind, generiert. Somit beinhaltet das Objekt alle im Dokument hinterlegten PTBoxeinträge. Um die Einträge übergeben zu bekommen, kann man die Methode getPTBEntryArray() auf das PTBoxobjekt anwenden. getPTBEntryArray() ist allerdings eine Methode der für die PTBox generierten API. Die Einträge werden hierbei nicht einzeln sondern in einem Array übergeben. Es ist aber auch möglich mit getPTBEntryArray("Indexzahl") gezielt einen Eintrag zu selektieren.

PTBoxHandler
SetFilePath(String File) GetPTBox() SetPTBEntry(PTBEntryDocument.PTBEntry SetEntry)

Mit der Methode SetPTBEntry(PTBEntryDocument.PTBEntry SetEntry) wird ein Eintrag in der PTBox gespeichert. Dabei wird überprüft, ob der Eintrag schon existiert und aktualisiert werden kann, oder ob ein neuer Eintrag angelegt werden muss. Um die Konsistenz der Daten für die Benutztung der PTBox im Hauptprogramm zu wahren, wird die aktualisierte PTBox von SetPTBEntry(PTBEntryDocument.PTBEntry SetEntry) nach dem Speichern als Rückgabewert zur Verfügung gestellt.

4.2 Kommunikation mit dem RacerPro©

Um eine Kommunikation zwischen einer JAVA-Applikation und dem RacerPro© zu realisieren wird vom Hersteller eine JavaAPI die sogenannte JRacer-API zur Verfügung gestellt. Diese API ermöglicht es, über ein RacerPro©-Objekt mit der RacerPro©-Instanz zu kommunizieren. Bei der Erzeugung des Objektes wird dabei die IP-Adresse und der Port der Serverinstanz übergeben. Nun können Methoden auf das Objekt angewendet werden, die eine Verbindung zum Server aufbauen/schließen, Nachrichten an den Server senden und Antworten vom Server empfangen können. Die empfangenen Servernachrichten werden aufbereitet und dem Hauptprogramm als Stringantwort übergeben. [RacerPro© API Site]

4.2.1 Erzeugung von Queries

Nachrichten mit Suchanfragen, die an die RacerPro©-Instanz gesendet werden, müssen, wie in 3.4 beschrieben, Queries sein und in nRQL verfasst sein. Dazu wird in der Funktion createNRQLQuery(String object) des Hauptprogramms eine Query erzeugt, wobei es sich bei dem Übergabewert "object" um das Indiviuum handelt, das durch die Query vom RacerPro© abgefragt werden soll. Aus 2.2.1 geht hervor, dass in diesem Projekt Elemente der Conditional Constraints als Objekte in die Queries eingesetzt werden sollen. Die Evidence und die Schnittmenge von Evidence und Conclusion eines Conditional Constraints sollen zur Erstellung der Queries benutzt werden, da es die Aufgabe ist über eben diese Elemente Informationen zu sammeln. Die Inhalte von Evidence und Conclusion können mit getEvidence() bzw. getConclusion() aus dem PTBoxeintrag geholt werden.
Eine erzeugte Query wird als "String" zurückgegeben und ist von folgender Gestalt:

```
(retrieve (?x) (?x EVIDENCE)) bzw.

(retrieve (?x) (?x (and EVIDENCE CONCLUSION)))
```

4.2.2 Suchanfrage an den RacerPro©

Bevor eine Suchanfrage durchgeführt werden kann, sollte die RacerPro©-Instanz zuerst einen "Full Reset" vollziehen, um eventuell unvollständige Prozesse auf der RacerPro©-Instanz zu beenden und den RacerPro© für eine neue Aufgabe vorzubereiten. Der Befehl, der dafür an den RacerPro©

gesendet werden muss, lautet: “(full-reset)”. Der Server bestätigt den erfolgreichen Reset mit der Antwort: “:okay-full-reset”. Nun kann der Racer mit dem Befehl “(owl-read-file owlFile)”, wobei “owlFile” dem Pfad einer ABox entspricht, instruiert werden eine ABox zu laden. Der Befehl kann beispielweise folgender sein:

```
(owl-read-file "D:/HighJumpTest/Highjump4InstanceswithBar.owl")
```

Ein erfolgreiches Laden zeigt der RacerPro© mit “read OWL-File done” an. Jetzt ist der RacerPro© in der Lage, Suchanfragen für ein Individuum auf die geladene ABox anzuwenden. Mit der Methode “.send(Query)” wird die Query an den RacerPro© gesendet.
Nachdem die Anfrageoperationen abgearbeitet worden sind, muß die Verbindung zur RacerPro©-Instanz geschlossen werden, um ein Überschreiten der Anzahl maximal möglicher Verbindungen durch ungenutzte Verbindungen zum RacerPro© zu verhindern.
Dieser gesamte Vorgang wird in der Funktion “sendNRQLQuery()” realisiert, die die RacerPro©-Antwort als Rückgabewert übergibt.
Der für den Ladebefehl “(owl-read-file “owlFile”)” wichtige Dateipfad der ABox wird vorher durch eine Suche mit Filterfunktion ermittelt. Die Funktion “getOWLFileArray()” übergibt ein Array mit allen ABoxen, nachdem in einem als Parameter übergebenen Ordner alle Datein ausgewählt wurden, die die Dateiendung “.owl” besitzten. Die Filterung wird hierbei durch das Objekt der Klasse “OWLFileNameFilter” durchgeführt. Das Objekt unterscheidet bei der Filterung die Dateitypen einzig anhand der Dateiendung. Eine inhaltliche Überprüfung ist mit dieser Klasse nicht möglich und auch nicht notwendig, da die Integritätsprüfung der Dokumente, wie in 3.2 beschrieben, eine Funktion des RacerPros© ist.

4.3 Verarbeitung der RacerPro©-Antwort

Nach der Verarbeitung einer Suchanfrage sendet der RacerPro© einen Antwortstring. Durch die Anwendung von Suchanfragen auf eine Menge von ABoxen, werden somit eine Vielzahl von Antworten generiert. Bei der so erzeugten Menge von Antworten muss allerdings noch eine Filterung auf gültige und ungültige Antworten durchgeführt werden.
Bei positiver Beantwortung einer Suchanfrage kann eine Antwort, wie in 3.4 angeführt, aus einem oder mehreren Antworttupeln bestehen. Für den Fall, dass einer Suchanfrage durch den RacerPro© keine Antwort aus dem in den

ABoxen enthaltenen Wissen zugeordnet werden kann, antwortet die Serverinstanz mit dem String “NIL”. In diesem Fall wurde für das gesuchte Indiviuum keine Instanz in einer ABox gefunden.
Vor der weiteren Auswertung der Serverantworten muss zwischen Antworten mit Tupeln und Anworten mit “NIL” unterschieden werden, um negative Antworten vor dem nächsten Verarbeitungschritt zurückzuhalten. Das Aussortieren der negativen Antworten wird durch eine if()-Abfrage realisiert, indem die Bedingung gesetzt wird, dass nur Antwortstrings zugelassen werden, deren Inhalt nicht “NIL” ist.

4.4 Auszählung

An die Funktion countElements(String) werden die Antworten übergeben, die Antworttupel als Inhalt besitzten. Nun werden die Antworten gezählt, indem ein Zähler stetig um “1” erhöht wird, wenn die Stringanalyse ein “?X” findet. Da die Antworttupel bei hier gestellten Anfragen immer von der Form “(?X ’Instanz’)” sind, kann ein einfacher Algorithmus gewählt werden, um die “?X” zu zählen.
Die Anfragen, die aus einem Conditional Constraint erzeugt werden, bestehen zum einen aus der Schnittmenge von Evidence und Conclusion und haben zum anderen nur die Evidence zum Parameter. Die Ergebnisse der Auszählung getrennt von einander aufaddiert. Sobald alle ABoxen jeweils um die Schnittmenge, als auch um das einzelne Konzept abgefragt sind und die Auszählung abgeschlossen ist, findet die Berechnung der unteren Grenzwahrscheinlichkeit statt. Dazu werden die Ergebnisse der Auszählungen wie in 3.1 beschrieben dividiert.
Die Funktion, die diese Aufgabe übernimmt, heißt calculateBoundaries(). Sie führt die Division durch und gibt das Ergebnis als Rückgabewert an das Hauptprogramm zurück.

4.5 Zuordnung des Ergebnisses und Speicherung

Das Hauptprogramm führt für jedes Conditional Constraint getrennt die Suchanfrage auf die gegebene Wissensbasis durch. Dabei werden die Conditional Consraints der Reihe nach abgearbeitet. Sobald ein Ergebnis ermittelt ist, wird es dem PTBoxEntry, welches auch das dazugehörige Conditional Constraint beinhaltet, hinzugefügt und mit der in 4.1.3 beschriebenen Funktion SetPTBEntry() in der PTBox abgelegt.

Sobald alle PTBoxeinträge bzw. alle Conditional Constraints verarbeitet und die berechneten unteren Grenzwahrscheinlichkeiten zugeordnet sind, wird die nun aktualisierte PTBox in das PTBoxdokument geschrieben und das Programm beendet.

5 Experiment

Für einen Test mit dem Prototypen und zur Verifizierung der Queries wurden zwei ABoxen und eine PTBox erstellt. In den ABoxen wurden in gleicher Anzahl Individuen des Konzeptes $HighJump$ mit und ohne der Relation $hasPart$ auf das Konzept Bar dargestellt. Die Relation $hasParticipant$ auf das Konzept $Person$ war für jedes Konzept gesetzt. Diese Informationen sollten bei der Bestimmung der unteren Grenzwahrscheinlichkeit der beiden Conditional Constraints

$$(HighJump \mid \exists hasParticipant.Person \sqcap \exists hasPart.Bar)[0,1]$$

und

$$(HighJump \mid \exists hasParticipant.Person)[0,1]$$

verwendet werden und zu folgendem Ergebnis führen:

$$(HighJump \mid \exists hasParticipant.Person \sqcap \exists hasPart.Bar)[0.5,1]$$

und

$$(HighJump \mid \exists hasParticipant.Person)[1,1]$$

5.1 Test

Zur Verifizierung der zu erzeugenden Queries und Befehle für den RacerPro© wurde das RacerPorter-Interface benutzt, um alle Befehle manuell eingeben zu können. Für den Verifizierungsvorgang wurde der Ablauf der Befehle des Projekts simuliert. Zunächst wurde ein "Full Reset" an den RacerPro© gesendet. Darauf wurde eine ABox mit dem "read-owl-file"-Befehl geladen, die aus dem Wissensbereich "Athletic Events" stammt. Abgefragt wurden das Konzept $HighJump$ und die Schnittmenge der Konzepte $HighJump$ und $hasParticipant.Person$. Die Antworten des RacerPro© wirkten zunächst sinnvoll.
Der darauf folgende Testlauf des Prototyps erzeugte jedoch nicht die erwarteten Ergebnisse. Die zu ermitteltenden Grenzwahrscheinlichkeiten hatten den Wert "1", waren also immer wahr. Außerdem fiel auf, dass der Programmdurchlauf für die geringe Anzahl von gebenen Testdokumenten viel Zeit in Anspruch nahm.

5.2 Analyse

Im folgenden Teil werden die Ursachen für die Ermittlung der falschen Werte der Grenzwahrscheinlichkeiten und das Performanceproblem analysiert und Lösungverschläge dargestellt.

5.2.1 Analyse der fehlerhaften Grenzwahrscheinlichkeitswerte

Bei genauer Analyse der Ergebnisse stellte sich heraus, dass nicht alle der gefundenen Instanzen direkt der gesuchten Schnittmenge von Konzepten entsprachen. Der Grund für das Finden von indirekten Instanzen liegt in der Reasoningfunktion des RacerPro©.
Mit Reasoning sucht der RacerPro© Instanzen, die dem gesuchten Konzept indirekt entsprechen. Das heisst, dass die Eigenschaften, die das Konzept der gesuchten Instanz besitzt, ebenfalls genutzt werden, um Instanzen zu finden. Zur Findung von entsprechenden Instanzen bedient sich der RacerPro© nicht nur der Suche in ABoxen sondern auch der Suche nach Instanzen gestützt durch die in TBoxen gespeicherten Definitionen von Konzepten.
In diesem Beispiel wurden die Instanzen gesucht, die der Schnittmenge von $HighJump$ und $\exists hasParticipant.Person \sqcap \exists hasPart.HorizontalBar$ entsprechen, also alle $HighJump$ - Veranstaltungen, die ein Element vom Typ $HorizontalBar$ besitzen. Der RacerPro© liefert auch Instanzen von Veranstalltungen, die keine explizit modellierte $HorizontalBar$ besitzen.
Der Grund für diese Ergebnisse liegt in der Definition der $Highjump$ - Veranstaltung in den TBoxen.

$$HighJump \equiv \exists hasParticipant.Person \sqcap \exists hasPart.HorizontalBar$$

Die Definition des Konzepts $HighJump$ besitzt die Existenz eines Elements $HorizontalBar$ als Eigenschaft. Es kann aber vorkommen, dass die Eigenschaft bei der Instanziierung des Konzeptes zur Darstellung einer $HighJump$ - Veranstalltung in einer ABox nicht gesetzt wird. Beispielsweise wenn für die Veranstaltung kein Element $HorizontalBar$ (mehr) existiert. Trotzdem wählt der RacerPro© alle Veranstaltungen aus, da über die Definiton des Konzeptes $HighJump$ alle Instanzen die Anfrage indirekt erfüllen.
Um Suchanfragen ohne Reasoning durchzuführen, ist es möglich den Suchmodus des RacerPro© zu ändern. Im "nRQL-Modus 0" führt der RacerPro© Suchanfragen durch, ohne die Definitionen von Konzepten in TBoxen für die Entscheidungsfindung zu benutzten. Dieser Modus besitzt jedoch starke Einschränkungen bezüglich der Vollständigkeit. Mit dem Befehl "(set-nrql-mode

0)" fordert man den Racer auf den Suchmodus zu ändern. Der RacerPro© antwortet bei erfolgreichem Wechsel mit ":okay-mode-0". Zusätzlich muss jetzt auch die Form der Queries geändert werden. Man teilt beispielsweise eine Conjuctive Query

```
(retrieve (?x) (?x and http://www.boemie.org/aeo.owl#HighJump
(some http://www.boemie.org/aeo.owl#hasPart
http://www.boemie.org/aeo.owl#HorizontalBar)))
```

zu

```
(retrieve (?x) (and (?x http://www.boemie.org/aeo.owl#HighJump)
(?x (some http://www.boemie.org/aeo.owl#hasPart
http://www.boemie.org/aeo.owl#HorizontalBar))))
```

auf. Beide Bedingungen müssen weiterhin von einer Instanz erfüllt werden, damit sie als Ergebnis der Query gilt. Der RacerPro© ist jedoch durch die Unvollständigkeit des Modus nicht in der Lage, Argument wie

```
(?x (some http://www.boemie.org/aeo.owl#hasPart
http://www.boemie.org/aeo.owl#HorizontalBar))))
```

mit Instanzen zu beantworten, so dass ein "NIL" als Antwort vom RacerPro© zurückgegeben wird. [Racer User Guide Mode]

5.2.2 Analyse des Performanceproblems

Bei der Entscheidung zum Design des Programms, wurde der Ablauf so gewählt, dass zuerst der Inhalt der PTBox aus einem Dokument gelesen und die Pfade der ABoxen ermittelt wurden. Eine Schleife arbeitet nach und nach jeden PTBoxeintrag in einem Durchlauf ab. Pro Schleifendurchlauf wurden Queries zu dem jeweils aktuellen Eintrag erzeugt und sämtliche ABoxen wiederum in einer Unterschleife bearbeitet, indem sie nacheinander in den RacerPro© geladen und die Anfragen an den RacerPro© gestellt wurden. Parallel wurden die Ergebnisse ausgewertet und aufaddiert. Am Ende eines jeden Hauptschleifendurchlaufs wurde die untere Grenzwahrscheinlichkeit errechnet und der gerade aktuelle PTBoxeintrag mit dem Ergebnis aktualisiert. Es stellte sich jedoch bei Versuchen heraus, dass die Wahl dieses Designs von großem Nachteil für die Performance des gesamten Programms ist. Dadurch,

dass für jeden PTBoxeintrag erst einmal alle ABoxen in einer Schleife abgefragt wurden, macht sich der Ladevorgang von ABoxen durch den RacerPro© als "Performancebremse" bemerkbar. So wurden NxM Ladevorgänge erzeugt. (N = Anzahl Conditional Constraint, M = Anzahl ABoxen).
Für eine deutliche Performancesteigerung ist es sinvoll, die Schleifenreihenfolge dahingegend zu ändern, dass in der Hauptschleife ABoxen nacheinander geladen und in einer Unterschleife sämtliche aus Conditional Constraints erzeugten Queries auf die jeweils geladene ABox angewendet werden. Diese Optimierung des Programmdurchlaufs reduziert den Zeitverlust durch Ladevorgänge erheblich. Die Anzahl der Ladevorgänge sinkt auf M Vorgänge.

6 Zusammenfassung und Ausblick

Im Zuge der Durchführung dieses Projekts wurde deutlich, dass es mit dem gewählten Ansatz nicht möglich ist, das Bestimmen der unteren Grenzwahrscheinlichkeit zu automatisieren.
Mit der Erstellung eines XML-Schemas konnte die Dokumentenstruktur komfortabel festgelegt werden, um PTBoxen und deren Inhalt, die Conditional Constraints, darzustellen. Dabei waren alle Elemente zur Darstellung der PTBox vollständig und mit allen Eigenschaften durch das XML-Schema definierbar. Das Modul XMLBeans, das vom Apache XML Project entwickelt wurde ermöglicht die die Dokumenten- und Inhaltsverwaltung durch eine auf das XML-Schema bezogene API. Diese API lässt sich in jede beliebige JAVA-Applikation einbinden, um PTBox-Dokumente nutzbar zu machen. Eine Änderung am XML-Schema der PTBox-Dokumente erfordert allerdings eine erneute Generierung der API.
Der RacerPro© war als sematischer Webserver in der Lage die Queries zur Instanzenabfrage, die durch das entwickelte Programm gezielt erzeugt werden, zu verarbeiten und zu beantworten. Für diese Aufgaben lud er die ABoxen und bot die Funktion in den ABoxen nach Instanzen zu suchen, die die in den Queries gestellten Bedingungen erfüllten.
Im Zuge von Experimenten für das Projekt stellte sich allerdings heraus, daß die Reasoningfunktion die Ergebnisse verfälschte. Es wurden vom RacerPro© durch Reasoning nicht nur direkte sondern auch indirekte Instanzen der gesuchten Konzepte gefunden. Für die statistischen Erhebungen der Instanziierungen von Konzepten, mit deren Hilfe die untere Grenzwahrscheinlichkeit eines Conditional Constraints berechnet wurde, waren aber nur die direkten Instanzen von Bedeutung. Dieses Problem liess sich aber auch nicht durch ein Expriment lösen, in dessen Zuge der RacerPro© in den "Mode 0" versetzt wurde. In diesem Modus führt der RacerPro© die Suchanfragen nur über die gegebenen ABoxen aus und ignoriert die in den TBoxen hinterlegten Definitionen von Konzepten, die sonst zur indirekten Instanziierung herangezogen werden. Die in 5.2.1 aufgezeigten Veränderungen der Querystruktur bringen ebenfalls nicht den gewünschten Erfolg.
Ein weiterer Lösungsansatz kann die Verwendung von Conjunktive Queries sein. Hierbei werden die Argumente wie im folgenden Beispiel dargestellt:

```
(|retrieve| (|?x|) (and
(|?x| |http://www.boemie.org/BOEMIE_ontologies/aeo.owl#HighJump|)
(|?x||?y| |http://www.boemie.org/BOEMIE_ontologies/aeo.owl#hasParticipant|)
(|?y| |http://www.boemie.org/BOEMIE_ontologies/aeo.owl#Person|)
(|?x||?z|  |http://www.boemie.org/BOEMIE_ontologies/aeo.owl#hasPart|)
(|?z| |http://www.boemie.org/BOEMIE_ontologies/aeo.owl#HorizontalBar|)))
```

Die Conjunctive Query erzwingt die explizite Bindung von Argumenten an Variablen. Somit werden zum Beispiel nur die Instanzen "x" ausgewählt, die

eine explizite Relation mit "z" besitzten (hier die Instanzen mit einer expliziten Relation zu einer Instanz $HorizontalBar$). Diese Art der Queryerzeugung setzt voraus, dass die Conditional Constraints der folgenden Form entsprechen:

$$(Conclusion \mid (and(some\ hasPart\ Concept)\ldots))[l,u]$$

Um dies umzusetzen, ist es notwendig, dass Konzepterme aus den Conditional Constraints durch das Programm untersucht und in eine solche Query transformiert werden. Im Zeitrahmen der Studienarbeit war diese Aufgabe jedoch nicht mehr zu realisiern.
Mit den in 5.2.2 vorgeschlagenen Änderungen des Programmdesigns ist es möglich eine Optimierung der Programmperformance zu erzielen. Die in 5.2.1 aufgezeigte Problematik verlangt jedoch nach einer grundlegenden Veränderung des Designs der Queryerzeugung. Somit ist es mit dem hier gewählten Ansatz nicht möglich die Gewinnung von probabilistischem Wissen aus ABoxen zu automatisieren.

Literatur

[Racer User Guide Queries] elektronisches Dokument *RacerPro-User-Guide-1-9-2-beta.pdf*. Racer Systems GmbH & Co. KG, http://www.racer-systems.com. *RacerPro User's Guide Version 1.9.2*. 2007, Kapitel 6.1.1.1 , Seite 94

[Racer User Guide Mode] elektronisches Dokument *RacerPro-User-Guide-1-9-2-beta.pdf*. Racer Systems GmbH & Co. KG, http://www.racer-systems.com. *RacerPro User's Guide Version 1.9.2*. 2007, Kapitel 6.2.5.1 , Seite 212-213

[DLHB-2003-TBOX] Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider. *THE DESCRIPTION LOGIC HANDBOOK*. 2003, Kapitel 1.3.1 The TBox

[DLHB-2003-ABOX] Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider. *THE DESCRIPTION LOGIC HANDBOOK*. 2003, Kapitel 1.3.2 The ABox

[naeth07] Tobias H. Näth, ZD (B.Sc.). *Analysis of the average-case behavior of an inference algorithm for probabilistic description logics*. Diplomarbeit, 2007

[Giugno,Lukasiewicz;2002] Rosalba Giugno and Thomas Lukasiewicz. *P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web*. In JELIA, Seite 8697, 2002.

[adl04.pdf] elektronisches Dokument *adl04.pdf* Volker Haarslev, Ralf Möller, and Michael Wessel. *Querying the Semantic Web with Racer + nRQL* 2004

[RacerPro© API Site] Onlinedokument

`http://www.racer-systems.com/products/download/nativelibraries.phtml`

[XML Beans Overview] Onlinedokument

`http://xmlbeans.apache.org/overview.html`

[Fischer Kompakt] Onlinedokument

http://www.fischer-kompakt.de/sixcms/
detail.php?template=glossar_detail&id=186834#

[RacerPro© Product Site] Onlinedokument

http://www.racer-systems.com/products/
racerpro/index.phtml

[XML - 10 Points] Onlinedokument

http://www.w3.org/XML/1999/XML-in-10-points.html.en

[OWL] Onlinedokument

http://www.w3.org/TR/owl-features/