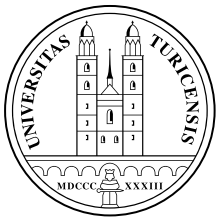

Universität Zürich
Institut für Informatik

Dynamic and Distributed
Information Systems

Diplomarbeit vom 26. Mai 2006

NLP-Reduce

Ein natürlichsprachliches Suchsystem für „Semantic Web" -Daten

Lorenz Fischer
von Basel BS, Schweiz

Matrikel-Nr.: 01-714-013
lorenz.fischer@access.unizh.ch

Betreuerin: **Esther Kaufmann**

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.unizh.ch/ddis

Zusammenfassung

Das *Semantic Web* ist die Vision der Verknüpfung verschiedenster Datenbasen und deren Anreicherung mit semantischer Information. Die dadurch erfassten Daten und Metadaten können über formale Abfragesprachen durchsucht werden. Hauptziel der Aktivitäten rund um das Semantic Web ist es, Daten nicht mehr wie im *World Wide Web (WWW)* üblich durch Volltextsuchmaschinen, sondern durch intelligente Algorithmen verarbeit- und durchsuchbar zu machen. Es soll also nicht mehr nur humanoiden Benutzern möglich sein, die Informationen dieses Netzes zu verstehen. Die maschinelle Verarbeitung der semantischen Daten steht im Vordergrund.

Nun sind sich jedoch nur Programmierer und Forscher gewohnt, Daten mittels formaler Abfragesprachen zu durchsuchen. Den normalen Anwender können komplizierten Queries leicht überfordern. Eine Mögliche Lösung um dieses Problem zu umgehen stellen natürlichsprachliche Anfragesysteme dar.

Die vorliegende Diplomarbeit befasst sich mit einem solchen Zugangssystem, dass natürlichsprachliche Anfragen in die formale Abfragesprache für OWL-Ontologien *SPARQL* übersetzen kann. Die in diesem Rahmen konzipierte Software *NLP-Reduce* wurde prototypisch in *Java* implementiert und evaluiert.

Abstract

The *Semantic Web* is the vision of the linking of various databases and the enrichment of those using semantic information. Users are able to query those databases using formal query languages. The major goal of the activities concerning the Semantic Web is to provide a collection of data that can be sought by intelligent systems rather than using simple full text search engines. The main focus in the development of the Semantic web therefore lies in making the semantic data accessible to machines.

Most people are not used to query databases through formal languages. A possible solutions to deal with this complaint is the use of *Natural Language Processing Systems*.

This thesis engages in a system that is capable of translating natural language sentences into a formal query language for *OWL* ontologies called *SPARQL*. The developed conceptual design has been implemented and evaluated in a *Java* prototype.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit natürlichsprachlichen Suchmöglichkeiten auf Daten des Semantic Web. In einer kurzen Einführung werden zunächst die grundsätzlichen Ideen und Konzepte für die Speicherung von Daten im Semantic Web aufgezeigt. Weiter werden die wichtigsten Werkzeuge zum Umgang mit Daten dieser Art vorgestellt.

1.1 Das Semantische Web

Das Semantische Web (oder *Semantic Web*) ist eine Erweiterung des uns bekannten *Wold Wide Web* bestehend aus Webseiten und Links. Inhalte sollen nicht mehr, wie bei herkömmlichen Webseiten üblich, primär für den menschlichen Benutzer aufbereitet werden. Vielmehr steht die Möglichkeit zur automatischen Verarbeitung von Informationen im Vordergrund. Zu diesem Zweck wird Wissen mit Metainformation[1] angereichert. Dies geschieht zum einen, indem die Datensätze in Ontologien geschrieben[2] und somit in einen semantischen Zusammenhang gebracht werden, und zum andern, indem für die Beschreibung der Ontologien ein möglichst einheitliches Vokabular verwendet wird.

1.1.1 Ontologien

Eine bekannte Definition des Begriffs *Ontologie* stammt von Thomas Gruber[6] und lautet:

> Spezifikation einer Konzeptualisierung

In Ontologien werden also Konzepte definiert und formal beschrieben. Im Gegensatz zu herkömmlichen Datenbanken werden bei Ontologien aber nicht nur die direkten Nutzdaten sondern auch deren Beziehungen und Bedeutungen abgelegt. Verknüpfungen zwischen den einzelnen Ontologien ermöglichen es

[1] Daten, die Informationen über andere Daten enthalten, werden *Metadaten* oder *Metainformationen* genannt (siehe dazu auch[21]).

[2] Obwohl die Idee des Semantic Web die Verwendung von Ontologien nicht zwingend vorschreibt, werden diese doch meist für die Speicherung der Daten verwendet (siehe dazu auch [14]).

wiederum, verschiedene Datenbestände miteinander in eine Beziehung zu setzen. Ein wichtiger Punkt ist dabei stehts die Wiederverwendbarkeit von Information. Gespeichert werden die Daten jeweils in Tripelform. Diese Tripel bestehen aus Subjekt, Prädikat (Eigenschaft) und Objekt (Ausprägung der Eigenschaft) (Siehe hierzu auch [3]). Ein Beispiel[3]:

```
geo:mississippi2 geo:runsTrough geo:illinois
```

Diese Aussage würde in ausgeschriebener Form „The river called Mississippi runs through the state called Illinois“ lauten. Die Information darüber, dass mit „geo:mississippi“ ein Fluss und mit „geo:illinois“ der amerikanische Staat Illinois gemeint ist, verbirgt sich hinter dem Wort „geo“. Es handelt sich dabei um die Kurzform von „http://www.mooney.net/geo#“. Diese URI[4] zeigt auf eine Ontologie mit Informationen über geographische Tatsachen. Im konkreten Fall ist „geo:mississippi2“ eine Instanzierung des Konzepts „geo:River“. Ein Fluss kann, wie in der realen Welt, durch ein Gebiet (in unserem Fall ein Staatsgebiet) fliessen. Die Beziehung zwischen Fluss und Staatsgebiet wird durch „geo:runsThrough“ beschrieben. In einer sauber definierten Ontologie werden alle Konzepte und die Relationen zwischen diesen formal und in Textform beschrieben. Das heisst, dass nebst der Information über Domäne und Bereich[5] („geo:River“ und „geo:State“) für die Objekteigenschaften auch eine Beschreibung in Textform in einer Ontologie gespeichert wird.

Neben der durch Metainformation beschriebenen Struktur sind vor allem die Inferenz- und Integritätsregeln von Ontologien ein Grund für deren Anwendung bei der Erschaffung des Semantic Web. Inferenzregeln erweitern den Informationsgehalt der Beziehungen zwischen den einzelnen Konzepten und Attributen. Ein Beispiel: Das Attribut „geo:hasCapital“ ist invers zu „geo:isCapitalOf“. Wenn also dem Attribut „geo:hasCapital“ des Objekts „geo:illinois“ der Wert „geo:springfieldIl“ zugeordnet wird, kann über die beschriebene Inferenzregel auch die Aussage „geo:springfieldIl geo:isCapitalOf geo:illinois“ aus der Ontologie gelesen werden. Weil das Konzept „geo:Capital“ eine Subklasse des Oberkonzepts „geo:City“ ist, ergibt sich daraus, dass Springfield auch eine Stadt ist.

1.1.2 RDF und OWL

Im Februar 2004 hat das *World Wide Web Consortium (W3C)* das *Resource Description Framework (RDF)* und die *Web Ontology Language (OWL)*[6] für die Im-

[3] Die meisten Beispiele in dieser Arbeit beziehen sich auf Datensätze von R. Mooney und seinem Team(siehe dazu Abschnitt 3.2). In verschiedenen Projekten wurden die Ontologien von Mooney verwendet, um die Performanz der entwickelten Systeme miteinander vergleichen zu können. Diese Datensätze sind in englischer Sprache verfasst, weshalb auch die meisten Beispiele in diesem Text in englischer Sprache sein werden.

[4] URI steht für *Uniform Resource Identifier* (einheitlicher Ressourcen-Bezeichner) und zeigt, analog zu den bekannten URLs, auf eine Ressource (Informationsquelle).

[5] Die Domäne (engl. *domain*) einer Ontologie-Eigenschaft definiert ein Konzept, für welches die Eigenschaft verwendet werden kann. Der Bereich (engl. *range*) spezifiziert das Zielkonzept, mit welchem die Instanzen der Domäne Beziehungen eingehen können.

[6] Wenn man *Web Ontology Language* richtig abkürzen würde, ergäbe sich das Kürzel WOL. Diese Kurzform passte jedoch den Gründern von OWL nicht und so wählten Sie OWL. Eine erst später erfundene Rechtfertigung für das Vertauschen der beiden Buchstaben *W* und *L* ist die, dass die Eule im Buch *Winnie the Pooh* ihren Namen auch mit vertauschten Buchstaben geschrieben hat (WOL) und man diesen Missstand nun so wieder korrigieren würde (siehe dazu auch [8]).

plementierung des Semantic Web vorgeschlagen. RDF definiert ein Datenmodell, durch welches Daten in *XML* oder normalem *Text* gespeichert und mit semantischen Informationen angereichert werden können. RDF wurde im Hinblick auf die Verwendung im Internet und die maschinelle Verarbeitung entwickelt. Die Väter von RDF verfolgten mit ihrem Ansatz verschiedene Ziele: Neben der Möglichkeit, semantische Information und Inferenzregeln definieren zu können, standen vor allem auch die Universalität und die Erweiterbarkeit des Standards im Vordergrund. Des weiteren sollte das Datenmodell nicht proprietär sein und somit von einer möglichst grossen Benutzergruppe verwendet werden können. Das verwendete Vokabular soll soweit als möglich standardisiert, jedoch über die Verwendung von URIs flexibel und erweiterbar bleiben.

Die Idee von OWL setzt auf einer etwas höheren Ebene an. OWL baut auf RDF auf und erweitert dieses um verschiedene, für die Erstellung von nützlichen Ontologien nötige Konzepte und Funktionen. Wie bei der Erstellung von Ontologien steht auch bei der Entwicklung von OWL die Wiederverwendbarkeit von Information im Zentrum. Ziel von OWL ist es, bestehende Ontologien zu verwenden, zu erweitern und miteinander zu verbinden. Auf den daraus resultierenden grossen Ontologien soll es möglich sein, Inkonsistenzen zu entdecken und neue Information mittels Inferenzregeln zu generieren.

Bei der Erstellung von Ontologien ist darauf zu achten, dass alle Mitentwickler ein einheitliches Vokabular verwenden. Dies vereinfacht oder ermöglicht erst die Wiederverwendung des gespeicherten Wissens. Und genau das ist das Ziel von Arbeitsgruppen wie *RDF-Schema* oder *Dublin Core*: Die Entwicklung eines einheitlichen Vokabulars, welches wann immer möglich, verwendet werden soll, um die Daten von OWL-Ontologien zu beschreiben (siehe dazu auch [2, 11, 7, 1, 20]).

1.2 Suchmaschinen und NLP-Suchdienste

Mit dem zunehmendem Informationsangebot im Internet steigen auch die Anforderungen an die Suchsysteme, um gewünschte Inhalte auffinden zu können. Ein grosses Problem bei der Entwicklung von leistungsstarken Suchdiensten stellt dabei die unstrukturierte Form der Daten im Internet dar. Die meisten Inhalte sind „bloss“ in Textform (HTML) abgelegt und daher nur über Volltextsuchmaschinen verarbeitbar. Natürlich gibt es auch in HTML verschiedene Möglichkeiten, Metainformation zu erfassen. Diese werden jedoch einerseits nicht von allen Webentwicklern genutzt und zum anderen dienen auch diese Informationen lediglich dem schnelleren Auffinden von Stichwörtern und ermöglichen keine semantische Suche über die Inhalte. Der Ansatz des *Semantic Web* versucht genau dieses Defizit zu umgehen und schreibt für die Datenerfassung die Verwendung verschiedener Standards vor, die es Programmierern erleichern sollen, *intelligente* und schnelle Systeme für die Verarbeitung der Daten zu schreiben.

Abfragesprachen wie *RDQL* oder *SPARQL* ermöglichen dem Benutzer, formal exakte Anfragen an die Datenbestände des *Semantic Web* zu stellen. Ein Beispiel: Die folgende SPARQL-Abfrage sucht in der Ontologie *geo.owl* nach allen Städten (Subjekte, die dem Typ „geo:City“ entsprechen):

```
SELECT ?city
WHERE {
?city
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.mooney.net/geo#City>
  }
```

Das Verwenden von Anfragesprachen wie *SPARQL* hat für den ungeübten und gelegentlichen Benutzer zwei grosse Nachteile: Erstens muss er die Syntax der verwendeten Sprache erlernen und anwenden können. Das weitaus grössere Problem jedoch ist, dass auch ein geübter Programmierer auf einer Ontologie nur mit dem Wissen über die interne Struktur der Daten korrekte Antworten erhalten wird. Natürlich lässt sich diese Struktur mehr oder weniger komfortabel per *SPARQL* aus den Ontologien lesen. Bei grossen Ontologien allerdings, stösst auch eine trainierte Person schnell an ihre Grenzen. Aus diesen Gründen wird schon seit längerem an einer anderen Möglichkeit für das Beantworten von Suchanfragen geforscht: *Natürlichsprachliche Schnittstellen*[7] auf Ontologien oder andere Datenbanken stellen für die Anwendung bei Suchmaschinen eine gute Alternative zur Volltextsuche dar. Das Erlernen von Abfragesprachen entfällt und auch über die Struktur der Daten muss der Benutzer nichts wissen, weil diese sein Allgemein- oder Weltwissen widerspiegeln.

1.3 Zusammenfassung

Auf der Grundlage von RDF und OWL soll sich das *Semantic Web* weiterentwickeln. Dies bietet den Vorteil, dass die Daten nicht mehr nur von menschlichen Benutzern verwendet werden können, sondern durch die formale Beschreibung von Konzepten und Relationen auch exakte Suchanfragen möglich werden. Über Abfragesprachen wie *SPARQL* können die Daten des *Semantic Web* effizient durchsucht werden. Formale Abfragesprachen sind nicht für die breite Masse geeignet, da dem Benutzer nebst der Syntax der Sprache auch die innere Struktur der verwendeten Ontologien bekannt sein müssen.

[7] Natürlichsprachliche Systeme oder NLP-Systeme (engl. *natural language processing systems*) versuchen, natürliche Sprache maschinell zu verarbeiten.

Kapitel 2

NLP-Reduce - ein Versuch

NLP-Reduce ist der Name der Software, die im Rahmen dieser Arbeit konzipiert und prototypisch realisiert wurde. Im folgenden werden nach einer kurzen Einleitung die Systemarchitektur und die Funktionalität von NLP-Reduce beschrieben.

2.1 Motivation

Die meisten NLP-Systeme, die heutzutage zum Einsatz kommen, bauen auf Forschungsergebnissen von Linguisten oder Computerlinguisten auf. Die Computerlinguistik versucht, natürliche Sprache mittels Regeln und Theorien maschinell verarbeitbar zu machen. In [9] werden verschiedene solche Systeme vorgestellt. Obwohl einige davon beeindruckende Resultate erreichen, besitzen die meisten Schwachstellen, die im folgenden kurz erläutert werden. Dabei wird hauptsächlich auf jene Probleme eingegangen, die mit Hilfe von NLP-Reduce gelöst werden können.

2.1.1 Syntaxanalyse

Auf einer Syntaxanalyse basierende Systeme untersuchen die Eingabesätze der Benutzer anhand einer Grammatik. Diese beschreibt, wie eine Anfrage aussehen muss, um korrekt verarbeitet zu werden. Wenn ein NLP-System auf einer Grammatik aufbaut, ist es erforderlich, dass Benutzeranfragen grammatikalisch und orthographisch korrekt gestellt werden. Ist dies nicht der Fall, ist es beinahe unmöglich, eine Anfrage richtig zu beantworten. Falsch konjugierte Verben, inkorrekt deklinierte Nomen oder generell die Verwendung von falschen Kasus können einen Satzparser schnell zum Stolpern bringen. Eine Grammatik eines Parsers könnte zum Beispiel wie folgt aussehen:

S	->	NP VP
NP	->	Det N
Det	->	„Welches“, „Welche“
N	->	„Haus“, „Häuser“, „Fenster“
VP	->	V N
V	->	„hat“, „haben“

In ausgeschriebener Form ergeben sich daraus folgende Grammatikregeln:

1. Ein Satz besteht aus Nominalphrase (NP) und Verbalphrase (VP)
2. Eine Nominalphrase besteht aus einem Determinierer (Det) und einem Nomen (N)
3. Ein Determinierer ist entweder „Welches“ oder „Welche“
4. Ein Nomen ist entweder „Haus“, „Häuser“ oder „Fenster“
5. Eine Verbalphrase besteht aus einem Verb (V) und einem Nomen (N)
6. Ein Verb ist entweder „hat“ oder „haben“

Nach der obigen Grammatik wären die Sätze „Welches Haus hat Fenster“ oder „Welche Häuser haben Fenster“ korrekt. Leider wäre jedoch auch der Satz „Welches Fenster haben Häuser“ syntaktisch korrekt. Schlimmer jedoch ist, dass die kleinste Abweichung von dieser Grammatik ein Versagen des Satzparsers zur Folge haben würde. Natürlich sind diese Grammatiken erweiterbar. Die zugrundeliegenden Schwächen werden dadurch jedoch nicht behoben. Beim täglichen Gebrauch von Sprache verwenden die meisten Menschen nicht grammatikalisch korrekte Sätze, sondern machen unvollständige oder rein grammatikalisch gesehen fehlerhafte Aussagen. Natürliche Sprache ist im allgemeinen voll von Fehlern (siehe dazu auch [9]). Eine Möglichkeit, mit diesem Problem umzugehen, ist die automatische Korrektur von Fehlern. Dies birgt jedoch weitere Probleme. Wenn ein System auf eine fehlerhafte Eingabe keine Fehlermeldung, sondern ein unter Umständen nicht gewolltes Resultat liefert, wird es sich nicht durchsetzen können. Siehe hierzu auch Abschnitt 3.1.

Eine schöne Lösung für das Problem der grammatikalisch falsch gestellten Anfragen bietet das System *Ginseng* aus [10]. *Ginseng* basiert auf einer Grammatik mit dazugehörender Syntaxanalyse, leitet jedoch den Benutzer bei der Fragestellung. Somit kann verhindert werden, dass ein Benutzer Anfragen stellt, die das System nicht beantworten kann.

Aus obigen Gründen soll NLP-Reduce möglichst keine einschränkenden Regeln beinhalten. Die Eingabesätze werden keiner grammatikalischen oder lexikalischen Analyse unterzogen. Für die Suche nach Resultaten auf eine Anfrage sollen lediglich die in der Anfrage vorkommenden Wörter und deren Beziehung untereinander verwendet werden. Dabei werden die Wortbeziehungen nicht aus dem (potenziell falschen) Fragesatz gelesen, sondern aus den zugrundeliegenden Ontologien. Es wird davon ausgegangen, dass der Autor der verwendeten Ontologie exakter arbeitet, als ein Gelegenheitsbenutzer des Suchsystems.

2.1.2 Mustersuche

Eine weitere Gruppe von NLP-Systemen sind die sogenannten *Pattern-matching*-Systeme[1]. Diese Systeme bestehen aus einem Regelwerk von Mustern nach denen ein Eingabesatz untersucht wird. Das folgende Beispiel stammt aus [9]. In einer Datenbanktabelle stehen folgende Einträge:

[1] engl. pattern = Muster

COUNTRY	CAPITAL	LANGUAGE
France	Paris	French
Italy	Rome	Italian

Für diese Datensätze werden folgende Regeln definiert:

Muster	Aktion
... „capital“ ... [country]	Retourniere alle Einträge der Spalte CAPITAL für die gilt COUNTRY = [country]
... „capital“ ... „country“	Retourniere alle Einträge der Spalte CAPITAL mit den jeweiligen Einträgen der Spalte COUNTRY

Die Regeln werden folgendermassen interpretiert: Wenn ein Benutzer eine Anfrage stellt in der das Wort „capital“ gefolgt von einem Ländernamen vorkommt, wird der korrespondierende Eintrag der Spalte *CAPITAL* als Resultat zurückgegeben. Auf die Frage „What is the capital of Italy?“ würde das System über die erste Musterregel die Antwort „Rome“ retournieren. Analog würde auf die Frage „Capital and country please.“ eine Liste mit sämtlichen Ländern und Hauptstädten auf den Bildschirm geschrieben. Ein grosser Vorteil von *Pattern-matching*-Systemen ist deren Einfachheit. Eine syntaktische Satzanalyse entfällt, und auch sprachlich inkorrekt gestellte Anfragen können beantwortet werden. Um jedoch in einer Datenbank suchen zu können, muss das System diese Daten bereits „kennen“, das heisst, es müssen zu jeder Datenbank Regeln definiert werden. Der Ansatz bei der Entwicklung von NLP-Reduce ist demjenigen der Mustersuchsysteme zwar ähnlich, bietet aber den Vorteil, dass keine Regeln explizit definiert werden müssen. Vielmehr muss der Ersteller der Ontologie Weltwissen in die Daten selbst einbinden. Durch die hohe Wiederverwendbarkeit von OWL-Ontologien wird ihm diese Arbeit um einiges erleichtert. Wie in Abschnitt 1.1.2 erklärt wurde, gibt es verschiedene Datenquellen, die für die Neuerstellung einer Ontologie verwendet und ausgebaut werden können.

Ein weiterer Vertreter der natürlichsprachlichen Systeme sind die *Semantic Grammar Systems*. Diese basieren auf einer Satzparsergrammatik, verwenden jedoch zusätzlich semantische Informationen. Auch hier liegen die Nachteile auf der Hand: Einerseits wird auf einer Grammatik aufgebaut und andererseits müssen die semantischen Informationen zu jeder Datenbank separat erfasst werden.

2.1.3 NLP-Reduce

Da die Verwendung einer Grammatik und einer dazugehörenden Syntaxanalyse die Form der Anfragen zu stark einschränkt, soll NLP-Reduce ohne Syntaxanalyse arbeiten. Es wird eine Strategie ähnlich der einer Mustersuche angewendet, wobei nicht mit statischen Mustern, sondern mit Informationen aus den zugrundeliegenen Ontologien gearbeitet werden soll. Hierbei ist das Ziel, die Vorverarbeitung und Analyse der Eingabesätze auf ein Minimum zu redu-

zieren. Aus diesem Grund wurde der Name „NLP-Reduce“[2] gewählt. In den nächsten Abschnitten wird die gewählte Systemarchitektur genauer beschrieben.

2.2 Systemarchitektur

Abbildung 2.1 zeigt schematisch die Systemarchitektur von NLP-Reduce. Im folgenden werden die einzelnen Komponenten und Abläufe genauer erläutert und beschrieben. Die Reihenfolge, in der die einzelnen Teile von NLP-Reduce beschrieben werden, entspricht der Abfolge während der Ausführung der Applikation.

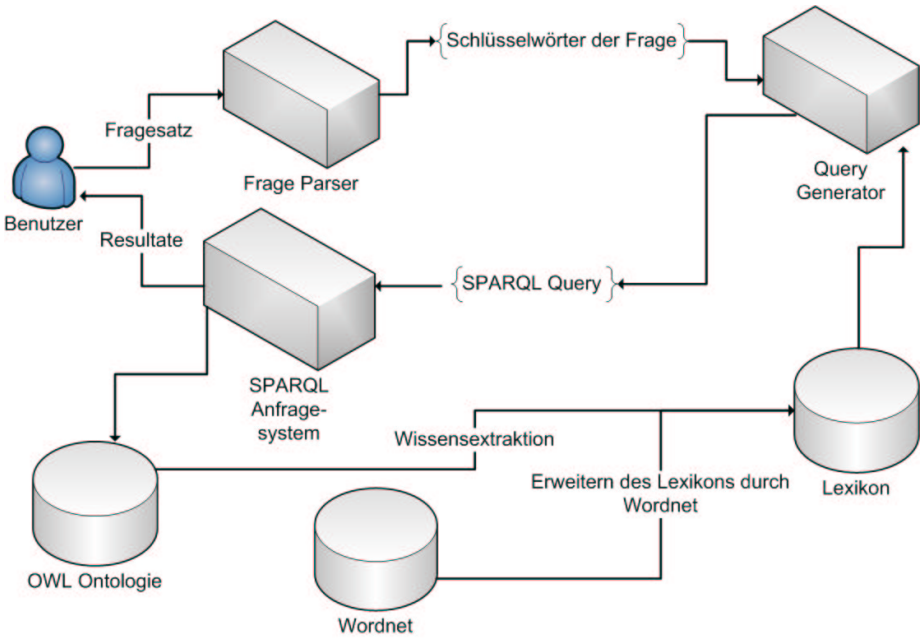

Abbildung 2.1: Systemarchitektur NLP-Reduce

2.2.1 Umgebung

Der Prototyps von NLP-Reduce wurde in Java [15] implementiert. Für das Auslesen und Abfragen der Ontologien wurden Jena [16] und Pellet [17] verwendet. Dies aus folgenden Gründen:

1. Java steht als freie Version für alle Betriebssysteme zur Verfügung
2. Java kann objektorientiert programmiert werden, was sich für NLP-Reduce gut eignet.

[2]NLP = Natural Language Processing (deutsch: Verarbeitung von natürlicher Sprache); Reduce (deutsch: Verminderung, Reduktion).

3. Mit Jena steht für Java eine gute Schnittstelle für die Arbeit mit Ontologien zur Verfügung. Jena unterstützt SPARQL - die Abfragesprache der Wahl für *Semantic Web*-Daten (siehe dazu auch den folgenden Abschnitt 2.2.2)

4. Pellet kann in Jena integriert werden und übernimmt das „Reasoning“ [3]

2.2.2 Das SPARQL Abfragesystem

SPARQL[4] ist jene Abfragesprache für RDF- (und damit auch OWL-)Daten, welche zur Zeit von der *RDF Data Access Working Group (DAWG)*[5] gefördert und vorangetrieben wird. In [4] wurde SPARQL vollständig spezifiziert und als *Candidate Recommendation*[6] freigegeben. Obwohl Jena auch andere Abfragesprachen unterstützt [7], wurde bei der Entwicklung von NLP-Reduce mit SPARQL gearbeitet. Dies vor allem im Glauben daran, das SPARQL zur Abfragesprache der Wahl bei RDF- und OWL-Datenbanken werden wird.

Während der Laufzeit greift NLP-Reduce zweimal per SPARQL auf die OWL-Ontologie zu. Einerseits wird beim Programmstart das Lexikon mit Information aus der Ontologie aufgefüllt, andererseits werden zum Schluss die generierten Anfragen über das SPARQL-Abfragesystem ausgewertet und die Resultate an den Benutzer zurückgegeben.

2.2.3 Die Ontologien

NLP-Reduce soll bei der Antwortsuche auf keine statisch definierten Muster oder Syntaxgrammatiken zurückgreifen. Deshalb werden alle für die Beantwortung einer Frage benötigten Informationen aus der Ontolgie gelesen[8]. Es ist deshalb von grosser Wichtigkeit, dass die zugrundeliegenden Ontologien möglichst komplett und vollständig sind. In den nächsten Abschnitten wird deshalb beschrieben, auf welche Details bei der Erstellung einer Ontologie geachtet werden muss.

Synonyminformation

Die wichtigsten Inhalte einer Ontologie, welche von NLP-Reduce verarbeitet werden soll, sind die Objekt- und Dateneigenschaften. Die folgenden Code-Ausschnitte aus der Geografie-Ontologie *geo.owl* zeigen beispielhaft die Definition der Objekteigenschaft „isLakeOf“ sowie der Datentypeigenschaft „lakeArea“.

[3] Um Inferenzregeln, wie in Abschnitt 1.1.1 beschrieben, auswerten zu können, wird ein sogenannter Reasoner benötigt. Diese Software verarbeitet Inferenzregeln einer Ontologie und generiert daraus quasi-neues Wissen. Siehe dazu auch Abschnitt 2.2.2

[4] SPARQL = Semantic Protocol and RDF Query Language

[5] Die DAWG ist eine Arbeitsgruppe des World Wide Web Consortium (W3C) [19]

[6] Eine *Candidate Recommendation* des W3C ist ein Dokument von dem das *World Wide Web Consortium* überzeugt ist, dass es von einer genügend grossen Anzahl von Leuten gegengelesen und überarbeitet wurde, sodass es nun für eine technische Evaluation freigegeben werden kann.

[7] Neben SPARQL steht auch eine Schnittstelle für RDQL zur Verfügung. Siehe auch [16].

[8] Später werden diese Informationen mit Daten aus dem einer Sprachdatenbank namens *Wordnet* erweitert.

```
<owl:ObjectProperty rdf:ID="isLakeOf">
<rdfs:subPropertyOf rdf:resource="#isIn"/>
  <rdfs:domain rdf:resource="#Lake"/>
  <rdfs:range rdf:resource="#State"/>
  <ginseng:ignore rdf:value="id text"/>
  <ginseng:phrase rdf:value="lies in"/>
  <ginseng:phrase rdf:value="lies in the state of"/>
  <ginseng:phrase rdf:value="lies in the OBJ state"/>
  <ginseng:phrase rdf:value="is in"/>
  <ginseng:phrase rdf:value="in"/>
  <ginseng:phrase rdf:value="is located in"/>
  <ginseng:phrase rdf:value="is located in the state_
   of"/>
  <ginseng:phrase rdf:value="is located in the OBJ_
   state"/>
  <ginseng:phrase rdf:value="are in"/>
  <ginseng:phrase rdf:value="are located in"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="lakeArea">
  <rdfs:domain rdf:resource="#Lake"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/_
   XMLSchema#float"/>
  <ginseng:ignore rdf:value="id text"/>
  <ginseng:phrase rdf:value="size"/>
  <ginseng:phrase rdf:value="area"/>
  <ginseng:interrogative rdf:value="how large"/>
  <ginseng:interrogative rdf:value="how big"/>
</owl:DatatypeProperty>
```

Beide Eigenschaften beziehen sich auf Instanzen der Klasse „Lake“. Während „lakeArea“ jedoch bloss einen numerischen Wert (nämlich die Fläche) eines Sees beschreibt, stellt „isLakeOf“ eine Verbindung zwischen einem See und einer Instanz der Klasse „State“ dar. Für NLP-Reduce sind vor allem die Zeilen beginnend mit „<ginseng:“[9] wichtig: Diese definieren Wörter oder Gruppen von Wörtern, welche von einem Benutzer verwendet werden können, um anzuzeigen, dass eine bestimme Objekt- oder Datentypeigenschaft bei der Antwortsuche verwendet werden muss[10]. Bei der Analyse des Fragesatzes „Which lakes are located in Illinois?“ wird NLP-Reduce aufgrund der Wörter „are“, „located“ und „in“ herausfinden, dass die Objekteigenschaft „isLakeOf“ für die Generierung des SPARQL-Query[11] benötigt wird. Analog funktioniert dies bei der Frage nach der Grösse des „Great Salt Lake“ (See im US Bundesstaat Utah) „What is the *size* of the *Great Salt Lake*“. NLP-Reduce wird aufgrund des Wortes „size“ die Datentypeigenschaft „lakeArea“ finden und diese für die Bildung des SPARQL-Queries verwenden.

[9] Da Ginseng([10]) ähnliche Informationen verwendet, wurde die gleiche Syntax verwendet. Dies in der Hoffnung, dass Ontologien somit für beide Systeme verarbeitbar sind.

[10] Ein Benutzer soll jedoch keinesfalls wissen müssen, welche Wörter er verwenden darf und welche nicht. Bei der Erstellung der Ontologie müssen möglichst alle wichtigen Vokabeln in die Ontologie geschrieben werden.

[11] engl. query = Anfrage

Natürlich werden bei einer Suche nach diesen Wörtern auch andere (ungewollte) Objekt- und Datentypeigenschaften gefunden. Wie diese zu einem späteren Zeitpunkt wieder aus der Menge der möglichen Antworten entfernt werden, wird in Abschnitt 2.2.6 beschrieben. Wichtig ist jedoch, dass NLP-Reduce lediglich die Wörter einer Anfrage verarbeitet und deren syntaktischen Zusammenhang komplett ignoriert. Die Verbindungen zwischen den einzelnen Teilen einer Anfrage soll sich ausschliesslich durch die in der Ontologie vorhandenen Informationen ergeben. Somit ist die Erfassung derartiger Synonyminformation für eine Ontologie, die durch NLP-Reduce verarbeitet werden soll, dringend notwendig.

Charakteristika von Eigenschaften

Wie bereits in Abschnitt 1.1.1 beschrieben, lassen sich in OWL zusätzlich zu den eigentlichen Datensätzen auch Metainformationen (auch semantische Informationen) speichern. Der folgende Auszug aus der Geografie-Ontologie *geo.owl* soll dies verdeutlichen:

```
<owl:Class rdf:ID="Country">
  <rdfs:subClassOf rdf:resource="owl#Thing"/>
  <ginseng:phrase rdf:value="countries"/>
</owl:Class>

<owl:Class rdf:ID="City">
  <rdfs:subClassOf rdf:resource="owl#Thing"/>
  <ginseng:phrase rdf:value="cities"/>
</owl:Class>

<owl:Class rdf:ID="State">
  <rdfs:subClassOf rdf:resource="owl#Thing"/>
  <ginseng:phrase rdf:value="states"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="isIn">
  <rdf:type rdf:resource="owl#TransitiveProperty" />
  <rdfs:domain rdf:resource="owl#Thing" />
  <rdfs:range rdf:resource="owl#Thing" />
  <ginseng tags...
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isStateOf">
  <rdfs:subPropertyOf rdf:resource="#isIn"/>
  <rdfs:domain rdf:resource="#State"/>
  <rdfs:range rdf:resource="#Country"/>
  <ginseng tags..
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isCityOf">
  <rdfs:subPropertyOf rdf:resource="#isIn"/>
  <rdfs:domain rdf:resource="#City"/>
```

```
  <rdfs:range rdf:resource="#State"/>
  <ginseng tags..
</owl:ObjectProperty>
```

Im angeführten Beispiel werden folgende Klassen und Eigenschaften definiert:

Element	Eigenschaften
geo:Country	Definiert die Klasse „Land“ als Unterklasse von „owl:-Thing“.
geo:State	Definiert die Klasse (Bundes-)“Staat“ als Unterklasse von „owl:Thing“.
geo:City	Definiert die Klasse „Stadt“ als Unterklasse von „owl:-Thing“.
geo:isIn	Definiert die Objekteigenschaft „geo:isIn“, wobei ein „owl:Thing“ in einem anderen „owl:Thing“ liegen kann. Zusätzlich wird diese Eigenschaft als „transitiv“ definiert.
geo:isStateOf	Definiert die Objekteigenschaft „geo:isStateOf“, wobei ein „geo:State“ in einem „geo:Country“ liegen kann. Zusätzlich wird dies Eigenschaft als Untereigenschaft von „geo:isIn“ definiert.
geo:isCityOf	Definiert die Objekteigenschaft „geo:isCityOf“, wobei eine „geo:City“ in einem „geo:State“ liegen kann. Zusätzlich wird dies Eigenschaft als Untereigenschaft von „geo:isIn“ definiert.

Eine Stadt wird also als „in einem Staat liegend“ und ein Staat als „in einem Land liegend“ definiert. Über die transitive Eigenschaft „owl:TransitiveProperty“ und die Tatsache, dass sowohl „geo:City“ und „geo:State“ wie auch „geo:Country“ als Unterklassen von „owl:Thing“ festgelegt wurden, kann folgende Information abgeleitet werden: Wenn sich eine Stadt *S* in einem Bundesstaat *B* und sich dieser wiederum in einem Land *L* befindet, muss sich die Stadt *S* zwangsläufig auch im Land *L* befinden. Durch die Verwendung der Objekteigenschaft „owl:TransitiveProperty“ wird somit auf einfache Weise eine Tatsache definiert, die für eine Maschine nicht ohne Weiteres „begreifbar“ wäre. Die Buchstabenreihenfolge „is In“ ist nur für den menschlichen Leser interpretierbar und muss für die automatisierte Verarbeitung explizit in einen semantischen Zusammenhang zu „is state of“ und „is city of“ gestellt werden.

Analog zum vorgestellten Beispiel mit „owl:TransitiveProperty“ wirkt sich auch die Verwendung anderer Eigenschaftscharakteristika positiv auf die Performanz von NLP-Reduce aus. Leider existieren zur Zeit nur eine beschränkte Anzahl solcher Charakteristika für OWL. Eine komplette Liste findet sich in [11]. Während das Einbinden von Synonyminformation für NLP-Reduce dringend notwenig ist, stellt die Implementierung von Eigenschaftscharakteristika eine optionale Möglichkeit dar, die Leistungsfähigkeit von NLP-Reduce zu erhöhen.

Subjekt	Prädikat	Objekt
geo:michigan2	rdf:type	geo:Lake
geo:illinois	rdf:type	geo:State
geo:michigan2	geo:isLakeOf	geo:illinois
geo:michigan2	geo:label	„michigan“
geo:illinois	geo:label	„illinois“
geo:michigan2	geo:lakeArea	58016

Tabelle 2.1: Beispiel: Tripel einer Ontologie

2.2.4 Extraktion des Lexikons

Wie in Abbildung 2.1 dargestellt, wird das Lexikon direkt aus der Ontologie gebildet. Optional kann dieser Grundstock von Information durch das Hinzufügen von Synonymen aus der Wörterdatenbank *WordNet*[12] erweitert werden. Wie in Abschnitt 1.1 beschrieben, werden die Informationen aus OWL-Ontologien in Tripelform gespeichert. Diese bestehen aus *Subjekt*, *Prädikat* und *Objekt*. In Tabelle 2.1 ist die Beziehung zwischen dem See „Michigan“ und dem amerikanischen Bundesstaat „Illinois“ beispielhaft dargestellt. NLP-Reduce unterscheidet zwischen zwei verschiedenen Arten von Tripeln: Erstens jene, bei welchen eine URI sowohl als Subjekt wie auch Objekt eingesetzt wird. Zweitens jene Tripel, bei welchen an der Position des Objekts ein Literal[13] verwendet wird. Da bei der Umsetzung der gefundenen Daten in SPARQL unterschiedlich vorgegangen werden muss, ist die Unterscheidung dieser beiden Tripelformen wichtig (näheres dazu folgt in Abschnitt 2.2.6). Die Extraktion der Lexikondaten aus der Ontologie geschieht deshalb auch in zwei Schritten. Im folgenden werden diese beiden Vorgänge einzeln erklärt.

Objekt und Datentypeigenschaften

Als erstes werden über SPARQL alle Objekt- und Datentypeigenschaften[14] aus der Ontolgie gelesen. Eigenschaften, welche lediglich der semantischen Beschreibung[15] dienen, werden dabei ignoriert[16]. Die gefundenen Einträge werden zusammen mit den jeweiligen Klassentypen (Domäne und Bereich) und den zugehörigen Synonyminformationen im Lexikon abgelegt. Zur Objekteigenschaft „geo:isCityOf“ würde beispielsweise folgende Information gespeichert:

[12] WordNet ist eine Wortdatenbank die Synonyminformationen in englischer Sprache enthält. Näheres dazu findet sich in [18]

[13] Ein Literal wird immer dann verwendet, wenn keine URI, sondern ein alphanumerischer Datenwert in der Ontologie gespeichert wurde. An jeder Stelle, an der sich ein Literal befindet, könnte auch eine URI als Platzhalter eingesetzt werden. Oft ist die Verwendung von Literalen jedoch praktischer und verständlicher. Tripel dieser Art wurden im Javasource als „LexikonLiteral“ bezeichnet.

[14] Da es sich bei diesen Daten ausschliesslich um Prädikate im Sinne von Tabelle 2.1 handelt, wurden sie im Javasource als *LexiconPredicates* bezeichnet.

[15] Beispielsweise Eigenschaften der Namensräume *RDF* oder *OWL* (z.B. rdfs:type)

[16] Nicht weil diese den späteren Suchprozess stören würden, sondern um die Menge der Einträge im Lexikon zu vermindern. Dies wiederum erhöht die Geschwindigkeit bei der Suche.

„geo:isLakeOf“ ist eine Objekteigenschaft, welche sich auf eine Instanz der Klasse „geo:Lake“ (Domäne) bezieht. Sie stellt eine Beziehung zwischen dieser Instanz und einer Instanz der Klasse „geo:State“ (Bereich) dar. Diese Eigenschaft soll immer dann als Teil der gesuchten SPARQL-Anfrage in Erwägung gezogen werden, falls einer der folgenden Texte[17] in genügendem Masse[18] im Fragesatz vorkommt: „lies in“, „lies in the state of“, „lies in the OBJ state“, „is in“, „in“, „is located in“, „is located in the state of“, „is located in the OBJ state“, „are in“ oder „are located in“.

Optional können die Textbausteine, welche zu den einzelnen Objekt und Dateneigenschaften gespeichert werden, mit Synonymen aus Wordnet erweitert werden. Natürlich gibt es meist für jedes einzelne Wort einer Eigenschaft mehrere Synonyme. In diesem Fall werden alle möglichen Wortkombinationen zu der Eigenschaft ins Lexikon gespeichert.

Um die Suche unabhängig von den verwendeten Wortendungen (Kasus, Numerus, Genus) durchführen zu können, werden die Wörter, welche für die Bildung der Wortindizes verwendet werden, vorgängig auf ihren Wortstamm zurückgeführt. Dafür wird der Stemming-Algorithmus von Martin Porter [12] verwendet.

Literale

Neben den Objekt und Datentypeigenschaften werden auch die in der Ontologie vorkommenden Literale im Lexikon abgelegt. Hierbei wird jedoch lediglich das Literal, dessen Wert und der dazugehörige Klassenname gespeichert. Für Eigennamen macht es keinen Sinn automatisiert in Wordnet nach Synonyminformation zu suchen. Jedoch kommt bei Literalen eine andere Form von Zusatzinformation zum Einsatz. Dies wird an folgendem Beispiel deutlich:

```
<Country rdf:ID="usa">
  <label>United States of America</label>
  <abbreviation>usa</abbreviation>
  <abbreviation>us</abbreviation>
  <abbreviation>america</abbreviation>
  <abbreviation>united states</abbreviation>
</Country>
```

Für die Instanz „geo:usa“ der Klasse „geo:Country“ werden nebst dem Wert für „geo:label“ mehrere Literale des Typs „geo:abbreviation“[19] in der Ontolgie erfasst. Dies bewirkt, dass nicht nur bei einer Benutzerfrage nach „United States of America“ sondern auch bei der Suche nach „usa“, „us“, „america“ oder „united states“ das Land „geo:usa“ als Teil der SPARQL-Anfrage identifiziert werden kann.

[17] Siehe dazu das Beispiel in Abschnitt 2.2.3

[18] Während der Generierung der SPARQL-Anfrage werden die einzelnen Bestandteile bewertet. Näheres dazu folgt in Abschnitt 2.2.6

[19] abbreviation = engl. Abkürzung

2.2.5 Vorverarbeiten der Benutzerfragen

Obwohl NLP-Reduce ohne computerlinguistische Analysen des Fragesatzes auskommt, kann durch eine minimale Vorverarbeitung der Benutzerfragen die Leistungsfähigkeit der Suche erhöht werden. Nebst dem einfachen Ersetzen von Zeichen[20] wird eine Suche nach Eigennamen durchgeführt. Dazu wird ein separater Index zur Zeit der Lexikonerstellung angelegt. Danach werden alle Wortgruppen markiert, die aus mehr als einem Wort bestehen und als Eigennamen identifiziert werden können. Dabei wird auf die Reihenfolge und die exakte Schreibweise der Wörter geachtet[21]. Wenn in einem Fragesatz zum Beispiel die Wortfolge „United States of America“ vorkommt, wird diese als Eigenname erkannt und nur als ganzes in der darauffolgenden SPARQL-Anfrage vorkommen. Falls sich ein Schreibfehler in der Wortfolge befindet oder die Reihenfolge der Wörter nicht zu 100% übereinstimmt, kann diese nicht als Name erkannt werden. Das ist jedoch nicht weiter schlimm, da immer noch die Möglichkeit besteht, dass der Name später wie ein normales Literal gefunden und verwendet werden kann.

In einem weiteren Schritt wird versucht, aufgrund der ersten Wörter einer Frage, den Fragetyp zu erkennen. So wird bei einem Fragesatz beginnend mit „how many“ wahrscheinlich eher die Anzahl der gefundenen Resultate und nicht eine Auflistung derselben gewünscht sein. Diese Information ist jedoch nur für eine automatisierte Auswertung der Leistungsfähigkeit von NLP-Reduce wichtig. Bei der grafischen Ausgabe aller Resultate wird immer die Anzahl der gefundenen Einträge dargestellt, was dem menschlichen Benutzer erlaubt, sich die Antwort auf seine Frage selbst zu geben.

Zu guter Letzt wird, in einem eigens dafür angelegten Index, nach Synonyminformation für Klassennamen gesucht (siehe dazu auch Abschnitt 2.2.3). Dadurch gefundene Klassennamen werden direkt in die Benutzerfrage integriert. So wird beispielsweise bei der Anfrage „Give me some good places for french food.“ das Wort „places“ durch den Klassennamen „Restaurant“ ersetzt[22]. Da beim Generierungsprozess der SPARQL-Anfrage nur noch nach Objekt- und Datentypeigenschaften gesucht wird, und die Klassennamen als Verbindung zwischen diesen verwendet werden, muss dies in der Vorverarbeitung geschehen.

Alle obigen Prozesse beeinflussen die Funktionsweise von NLP-Reduce nur minim. Auch ohne diese Optimierungen verrichtet NLP-Reduce seinen Dienst. Die Verminderung der vorkommenden Wörter im Fragesatz beschleunigt jedoch den Suchvorgang erheblich.

2.2.6 Generieren von SPARQL-Anfragen

Der komplexeste Programmteil von NLP-Reduce ist der Anfragegenerator. Deshalb wird dieser im Folgenden etwas ausführlicher beschrieben, als dies bei den anderen Programmkomponenten nötig ist. In Abbildung 2.2 wird der ganze Prozess schematisch aufgezeigt und in Tabelle 2.2 stichwortartig erklärt. Im folgenden werden die einzelnen Schritte im Prozess der Anfragegenerierung

[20] Vor der Verarbeitung des Fragesatzes werden beispielsweise sämtliche Satzzeichen entfernt.
[21] Gross- und Kleinschreibung ist unwichtig.
[22] Natürlich nur wenn diese Synonyminformation in der Ontologie erfasst wurde.

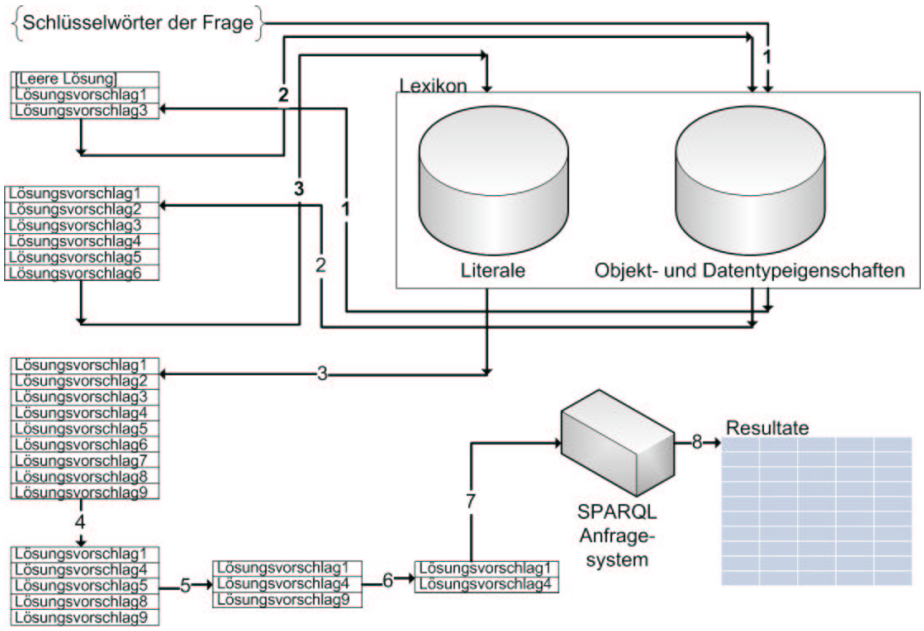

Abbildung 2.2: SPARQL-Anfrage Generator

genauer erläutert. Für diesen Zweck wird der Beispiel Fragesatz „How big are the cities of Illinois?“ verwendet. Die Schlüsselwörter dieser Frage lauten nach der Vorverarbeitung „How big are cities of Illinois“ (der Artikel „the“ wurde aus Effizienzgründen entfernt).

1.+2. Lexikonsuche im Index der Objekt- und Datentypeneigenschaften

In einem ersten Schritt wird eine Suche im Index der Objekt- und Datentypeigenschaften durchgeführt. Da diese Suche insbesondere bei grossen Ontologien eine erhebliche Menge an Resultaten ergeben kann, werden die gefundenen Eigenschaften bezüglich ihrer Ähnlichkeit mit dem Fragesatz bewertet.

Folgende Kriterien werden für diese Bewertung verwendet:

1. Die Anzahl der Wörter im Eintrag, die den Wörtern des Fragesatzes entsprechen, im Verhältnis zur gesamten Anzahl Wörter, die der Eintrag besitzt.

2. Die Genauigkeit der Übereinstimmung

Am Beispielsatz von oben würde dies folgendermassen aussehen: Gesucht ist die Objekteigenschaft „geo:isCityOf“ und die Datentypeigenschaft „geo:cityPopulation“. Für die Wörter „are“, „cities“ und „of“ werden unter anderem die Objekteigenschaften „geo:isCityOf“, „geo:isCapitalOf“, „isIn“ und „isHighestPoint“[23] gefunden. Tabelle 2.3 zeigt die Bewertungen aller vier Einträge[24].

[23] Diese Liste ist je nach Grösse der Ontolgie ziemlich lang. Deshalb werden hier nicht alle Resultate aufgezählt.

[24] Die Wörter werden sowohl beim Schreiben des Lexikons wie auch beim Suchvorgang auf ihren

Nr.	Beschreibung
1	Lexikonsuche nach allen Wörtern die im Fragesatz vorkommen. Um auch Fragen ohne Objekt- oder Datentypeigenschaften beantworten zu können, wird ein leerer Lösungsvorschlag mitgeneriert.
2	Lexikonsuche unter Verwendung der verbleibenden Wörter des Fragesatzes. So können Verbindungen unter den Objekt- und Datentypeigenschaften gefunden werden.
3	Lexikonsuche im Index der Literale unter Verwendung der verbleibenden Wörter.
4	Entfernen aller gefundenen SPARQL-Anfragen, die nicht alle Wörter des Fragesatzes verwenden.
5	Da es beim angewendeten Algorithmus zu Duplikaten kommen kann, werden in diesem Schritt alle mehrfach vorkommenden SPARQL-Anfragen aus der Menge der Lösungsvorschläge entfernt.
6+7	Die verbleibenden SPARQL-Anfragen werden bewertet, und nur jene Lösungen mit der besten Bewertung werden weiter an das SPARQL-Anfragesystem geleitet.
8	Die gefundenen Antworten werden dem Benutzer angezeigt.

Tabelle 2.2: Legende zu Abbildung 2.2

Um die Übersichtlichkeit zu wahren, ist pro Eintrag jeweils nur eine Gruppe von Wörtern ersichtlich. Natürlich werden beim Programmlauf alle vorhandenen Synonyme bewertet. Siehe dazu auch Abschnitt 2.2.3. Nebst der blossen Anzahl Wörter wird auch die Genauigkeit der Überinstimmung beachtet, das heisst, die Länge der gefundenen Wörter wird mit der Länge der Wörter im Fragesatz verglichen. Für das obige Beispiel würde also die Objekteigenschaft „geo:isCityOf“ die höchste Bewertung erreichen[25]. Somit würde diese Eigenschaft in die Menge der möglichen Lösungen aufgenommen und zusammen mit den verbleibenden Wörtern des Fragesatzes abgespeichert. Neben der Information darüber, welche Wörter des Fragesatzes bereits verwendet wurden, werden auch jene Klassennamen abgelegt mit welchen die Objekteigeschaft eine Beziehung eingehen kann. In diesem Fall wären dies die Domäne „geo:City“ und der Bereich „geo:State“.

Abbildung 2.3 zeigt schematisch, wie die einzelnen Bestandteile einer möglichen Lösung abgelegt werden[26].

Nicht alle Fragen müssen sich zwingend auf eine Objekt- oder Datentypeigenschaft beziehen. Ein Benutzer der eine Antwort auf „Give me all cities.“ möchte, erwartet lediglich eine Liste aller in der Ontolgie stehenden Städte.

Wortstamm gekürzt.

[25]In diesem Beispiel wurde eine einzige Objekteigenschaft als beste identifiziert und in die Lösungsmenge aufgenommen. Es gibt jedoch auch Fälle, bei denen mehrere Eigenschaften eine gleichwertige Bewertung erreichen. In diesen Fällen werden alle Eigenschaften mit der höchsten Bewertung zur Lösungsmenge hinzugefügt.

[26]Im Javasource werden die einzelnen Lösungen mit „QueryBucket“ und die Bestandteile einer Lösung mit „QueryToken“ bezeichnet.

Eintrag	Wörter des Eintrages	Wörter in Fragesatz
geo:isCityOf	„ar“, „citi“, „of“	3 von 3
geo:isCapitalOf	„ar“, „capital“, „citi“, „of“	3 von 4
geo:isIn	„ar“, „in“	1 von 2
geo:isHighestPoint	„ar“, „highest“, „point“, „of“	2 von 4

Tabelle 2.3: Bewertungsbeispiele für „How big are cities of Illinois?“

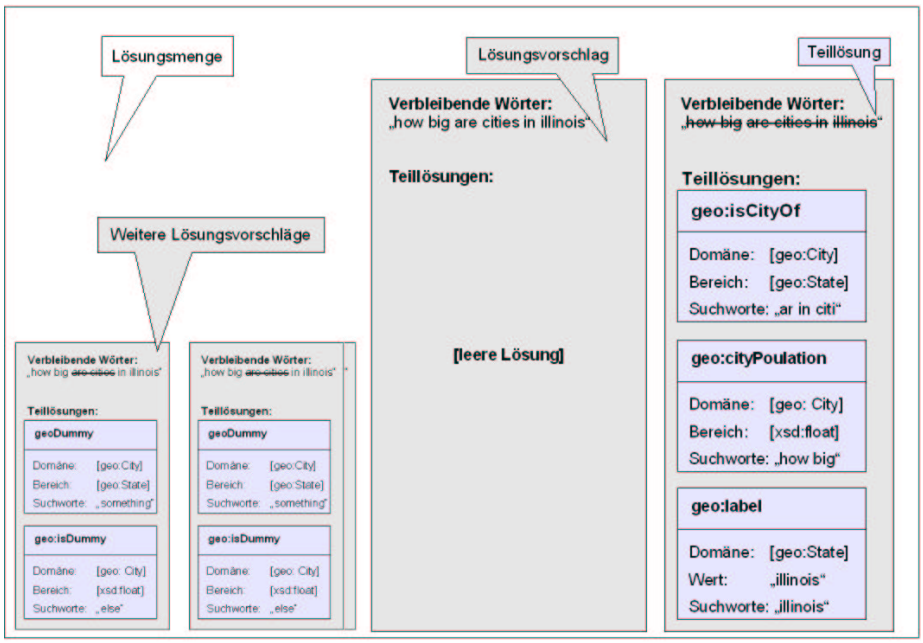

Abbildung 2.3: Teillösungen, Lösungsvorschläge und Lösungsmenge

Dabei ist es irrelevant in welchem Staat sich diese befinden oder wie gross die jeweiligen Städte bezüglich ihrer Einwohneranzahl sind. Um auch diese Art von Anfragen bearbeiten zu können, wird bei der Suche nach Objekt- und Datentypeigenschaften auch ein leerer Lösungsvorschlag in die Lösungsmenge aufgenommen. Da sich die Lösungsmenge ansonsten ausschliesslich aufgrund von Lexikoneinträgen zusammensetzt, würden Anfragen der oben beschriebenen Art ansonsten zu keiner Lösung führen.

Nach einem „ersten“ Suchlauf wird über alle gefundenen Teillösungen iteriert. In diesem zweiten Schritt werden zusammenhängende Objekt- und Datentypeigenschaften im Lexikon gesucht. Die Suche findet analog zu Schritt 1 statt - es wird jedoch nur nach den jeweiligen verbliebenen Wörtern der Anfrage gesucht. In unserem Beispiel würde also der Index nach den Wörtern „How big Illinois“ abgesucht. In diesem Fall werden gleich drei Datentypeigenschaften aufgrund ihrer Synonyminformation gefunden, und zwar sind dies „geo:cityPopulation“, „geo:lakeArea“ und „geo:stateArea“. Diese werden nun bezüglich ihrer Kompatibilität mit der bereits gefundenen Teillösung „geo:-

isCityOf“ verglichen. Als einzige Kombinationsmöglichkeit ergibt sich daraus die Verbindung von „geo:cityPopulation“ und „geo:isCityOf“ über die Klasse „geo:City“. Somit wird die Datentypeigenschaft „geo:cityPopulation“ als Teillösung zur bereits gespeicherten Objekteigenschaft „geo:isCityOf“ kopiert und die für die Suche verwendeten Wörter aus den verbleibenden Wörtern der Anfrage gestrichen. Dies ergibt als letztes noch nicht aufgelöstes Wort in der Anfrage „Illinois“.

3. Lexikonsuche im Index der Literale

Im dritten Schritt der Lexikonsuche wird der Index der Literale durchsucht. Auch hierbei werden die Suchergebnisse bewertet und nur jene Resultate mit der grössten Übereinstimmung weiterverwendet. In unserem Fall ergibt die Suche nach „Illinois“ genau einen Treffer, und zwar ein Literal vom Typ „geo:label“ mit dem Wert „illinois“. Dieses Literal setzt als Verbindungsobjekt eine Instanz der Klasse „geo:State“ voraus. Somit kann das Literal an die bereits im Lösungsvorschlag vorhandenen Objekteigenschaft „geo:isInCity“ anknüpfen, welche eine Instanz der Klasse „geo:City“ mit einer Instanz der Klasse „geo:State“ verbindet.

4. Entfernen unvollständiger Lösungsvorschläge

Für den Fall, dass sich mehrere Lösungsmöglichkeiten in der Lösungsmenge befinden, werden in diesem Schritt alle Lösungsvorschläge entfernt, welche eines der folgenden Kriterien nicht erfüllen.

1. Alle Wörter der ursprünglichen Suchanfrage müssen für die Lösungsfindung verwendet werden[27].
2. Es dürfen keine Verbindungen der Teillösungen (URIs) ungenutzt bleiben.
3. Ausnahme: Wenn Verbindungen nicht aufgelöst werden können, müssen die Namen dieser Verbindungen in der ursprünglichen Frage vorkommen. Siehe dazu das folgende Beispiel.
4. Ausnahme: Falls Wörter der Suchanfrage nicht verwendet wurden, müssen diese in der Liste der offenen Verbindungen vorkommen.

   Ein Beispiel: Für die Frage „Give me all cities of Illinois.“ wurden folgende Teillösungen gefunden:

   „geo:isCityOf“ mit Domäne „geo:City“ und Bereich „geo:State“
   „geo:label“ mit Domäne „geo:State“ und Wert „illinois“

   Die verwendeten Wörter sind „citi of illinois“[28]. Die Wortgruppe „Give me all“ wird durch die Vorverarbeitung der Anfrage weggeschnitten. Kriterium Nummer 1 wäre also erfüllt, da alle Wörter der

[27] Da in der natürlichen Sprache Wörter verwendet werden, die für die Beantwortung einer Frage unwichtig sind, werden vor dieser Prüfung all jene Wörter mittels einer Stop-Wort-Liste aus der Menge der „verbleibenden“ Wörter gelöscht.

[28] Die gestemmte Form von „city“ ist „citi“.

> Suchanfrage verwendet wurden. Jedoch würde der Lösungsvorschlag aus der Lösungsmenge entfernt, da er das Kriterium Nummmer 2 nicht erfüllt. Die URI „geo:City“ kommt in den Teillösungen nur einmal vor und konnte somit nicht vollständig aufgelöst resp. mit einer anderen Teillösung verknüpft werden. Nun tritt jedoch Ausnahme Nummer 1 in Kraft, nach der ein Lösungsvorschlag korrekt ist, falls der Name der Verbindung im ursprünglichen Anfragesatz vorkommt. Dies ist hier der Fall, da der Name „City“ in der Frage „Give me all **cities** of Illinois“ vorkommt.

In unserem Beispielfall befinden sich in diesem Schritt unter anderem[29] folgende zwei Lösungsvorschläge in der Lösungmenge: Zum einen die Konstruktion aus „geo:isCityOf“ und „geo:label“ und zum anderen die Verbindung zwischen der leeren Lösung und „geo:label“. Da bei Letzterem nicht alle Wörter der Suchanfrage aufgelöst wurden, wird er aus der Menge der möglichen Lösungen entfernt.

5. Entfernen von Duplikaten

Weil beim oben beschriebenen Suchalgorithmus Duplikate auftreten können, werden diese in diesem Schritt entfernt. Der Vergleich der Lösungsvorschläge kann je nach Komplexität der Suchanfrage viel Rechenleistung erforden. Aus diesem Grund wird dieser Schritt so spät wie möglich durchgeführt (nachdem ungültige Lösungsvorschläge bereits aus der Lösungsmenge entfernt wurden).

6. Bewertung der Lösungsvorschläge und Generierung der SPARQL-Anfragen

Ähnlich wie bei der Bewertung der gefundenen Lexikoneinträgen werden auch die gefundenen Lösungsvorschläge bewertet. Da sich ein Teil der Bewertungskriterien auf die Form des generierten SPARQL-Codes bezieht, wird die Bewertung der Lösungsvorschläge erst zum Zeitpunkt der Generierung des SPARQL-Codes durchgeführt. Aufgrund dessen, dass die meisten Kriterien die Bewertung eines Lösungsvorschlages verringern, wird bei allen Lösungsvorschlägen eine Grundbewertung von 10’000 Punkten[30] als Standard gesetzt. Wie bereits in Abschnitt 2.1 erklärt, versucht NLP-Reduce in keiner Weise den Inhalt des Fragesatzes zu „verstehen“. Aus diesem Grund haben alle generierten SPARQL-Anfragen folgende Form:

```
SELECT DISTINCT *
WHERE {
[SPARQL-Code für Teillösung 1]
..
[SPARQL-Code für Teillösung n]
}
```

[29] Da alle möglichen Lösungen gefunden werden sollen, werden unvollständige oder bereits erweiterte Lösungsvorschläge nicht aus der Lösungsmenge entfernt, sondern für eine allfällig spätere Erweiterung in dieser belassen.

[30] Für die Sortierung der Lösungsvorschläge wird ein Array verwendet, welches keine negativen Indices erlaubt.

Die Gesamtnote setzt sich aus den einzelnen Bewertungen für die Teillösungen zusammen. Zum Schluss wird die gesamte Note über einen Multiplikator gesteuert. Dieser bewertet eine lange Anfrage mit einer niedrigeren Note als eine kürzere[31]. Die Bewertung für die zwei Typen von Teillösungen funktioniert unterschiedlich. Im folgenden werden beide Vorgehensweisen kurz vorgestellt.

Vorgehen bei Teillösungen des Typs „Objekt- und Datentypeigenschaft“: Die Übersetzung von Teillösungen dieser Art in einen SPARQL-Teil erfolgt relativ einfach. Einerseits wird die Objekt- resp. Datentypeigenschaft in den korrekten Zusammenhang mit Domäne und Bereich gestellt, andererseits wird für alle dadurch generierten Variablen sichergestellt, dass sie der korrekten Typendefinition entsprechen. Somit ergibt sich folgende Form:

```
?[Domäne] [Objekt- resp. Datentypeigenschaft] ?[Bereich] .
?[Domäne] rdf#type [Domäne] .
?[Bereich] rdf#type [Bereich]
```

Dies ergibt für unser obiges Beispiel:

```
?City geo#cityPopulation ?cityPopulation .
?City rdf#type geo#City .
?City geo#isCityOf ?State .
?State rdf#type geo#State
```

Die Bewertung der SPARQL-Teile errechnet sich aus der Anzahl Wörter, die im Namen der Eigenschaft stehen im Verhältnis zur Anzahl derjenigen, die zu diesem Eintrag ins Lexikon eingetragen wurden.

Vorgehen bei Teillösungen des Typs „Literal“: Teillösungen für Literale werden sehr ähnlich in SPARQL übersetzt. Jedoch spielt hier die Bewertung eine grössere Rolle. Auch hier wird die Bewertung aufgrund des Verhältnisses der Anzahl Wörter im Literal zur Anzahl Wörter des Lexikoneintrages gemessen. Falls dieses Verhältnis jedoch unter 80%[32] fällt, wird nicht der ursprüngliche Wert des Literals bei der Suche verwendet. In diesen Fällen sucht NLP-Reduce mit Hilfe von *Regular Expressions* nach jenen Wörtern, die in der Original-Suchanfrage gefunden wurden. Im Normalfall wird der SPARQL-Code also nach folgendem Muster gebildet:

```
?[Domäne] [Name des Literals] [Wert]
```

Angewendet auf obiges Beispiel ergibt dies:

```
?State geo#label 'illinois'
```

[31] Dies aus dem Grund, dass eine richtige Antwort auch über Umwege gefunden werden kann, dies aber meist ungewollte Zusatzinformation mit sich bringt. So kann beispielsweise die Zugehörigkeit einer Stadt zu einem Land über den Bundesstaat eruiert werden. Kürzer (und wahrscheinlich eher beabsichtigt) ist jedoch die direkte Beziehung zwischen der Stadt und dem Land.

[32] Diser Wert hat sich während der Entwicklung von NLP-Reduce ergeben. Es handelt sich dabei also lediglich um eine heuristische Grösse.

Ein Beispiel für die Suche nach „virginia“ würde für das Literal mit Wert „west virginia“ wie folgt aussehen:

```
?State geo#label ?State_label .
FILTER( REGEX(?State_label,'virginia','i') )
```

Der obige Ausdruck führt eine Suche nach dem Text „virginia“ innerhalb der Werte des Literals „geo:label“ aus, wobei die Gross- und Kleinschreibung ignoriert wird.

2.2.7 Zurückliefern von Antworten

Bei der Programmierung von NLP-Reduce wurde nach dem Architekturmuster *Modell-Präsentation-Steuerung (MPS)*[33] vorgegangen. Die Datenhaltung wird in Form des Lexikons und die Steuerung im Anfragegenerator implementiert. NLP-Reduce unterstützt zur Zeit drei verschiedene Möglichkeiten, um dem Benutzer die Antworten zu präsentieren. Diese werden im folgenden kurz vorgestellt.

Manuelle Ausführung

Die einfachste und für den Endbenutzer mit Sicherheit unbrauchbarste Möglichkeit, NLP-Reduce zu starten, ist die manuelle Ausführung. Der Start der Applikation erfolgt über die Kommandozeile und akzeptiert als Parameter eine natürlichsprachliche Frage sowie den Pfad zur OWL-Ontologie, die zur Beantwortung der Frage verwendet soll. Diese Art der Ausführung ist nur für Entwickler gedacht, die nicht für jeden Testlauf die ganze grafische Benutzeroberfläche laden wollen.

Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche ist für normale[34] Benutzer gedacht. Im Gegensatz zur *manuellen Ausführung* ist beim Start der Applikation lediglich ein Parameter erforderlich. Dieser besteht im vollständigen Pfad zur Ontologie, die für die Anfragen verwendet werden muss. Die Oberfläche wurde sehr einfach gehalten. Abbildung 2.4 zeigt die einzelnen Komponenten: Im Textfeld am oberen Rand des Fensters kann der Benutzer eine Frage an das System stellen. Durch drücken des Knopfes *GO* oder durch Beenden der Eingabe mit der Eingabetaste wird die Suche gestartet. Die gefundenen Lösungsvorschläge werden darauf als Registerkarten unterhalb des Eingabefeldes dargestellt. Den SPARQL-Code und die dadurch gefundenen Resultate werden innerhalb des grossen Textfeldes beziehungsweise in Tabellenform dargestellt.

[33] Dies wird in englischer Sprache mit *Model-View-Controller* bezeichnet. Ziel ist eine Trennung von Daten, Algorithmen und Benutzeroberfläche.

[34] Im Gegensatz zu Entwicklern oder Forschern wird der Gelegenheitsbenutzer als „normaler“ Benutzer bezeichnet.

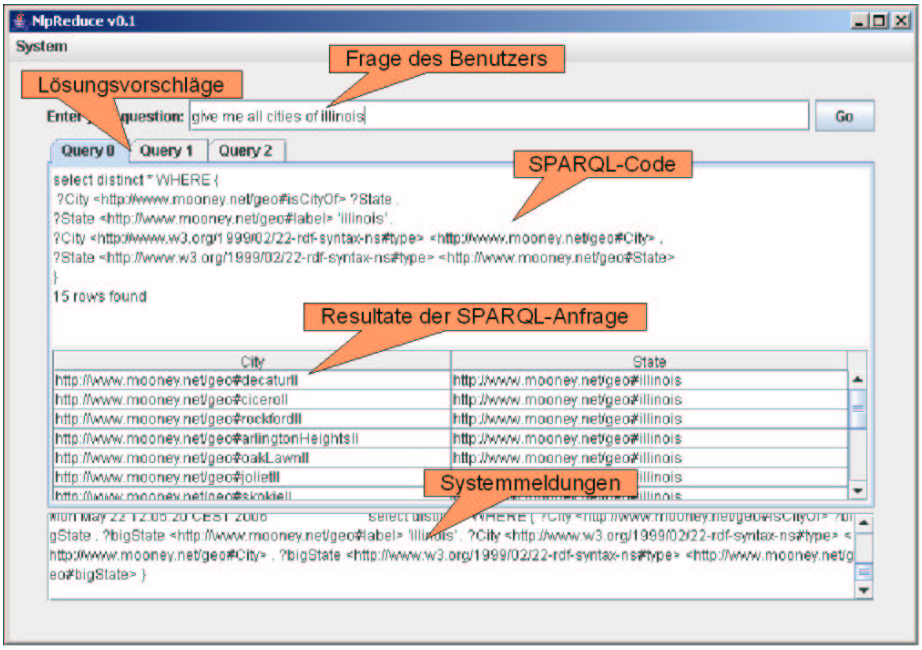

Abbildung 2.4: Graphische Benutzeroberfläche von NLP-Reduce

Automatisierte Evaluation

Als letzte Möglichkeit, Fragen über NLP-Reduce zu beantworten, steht eine automatisierte Evaluation zur Verfügung. Um eine aussagekräftige Bewertung der Performanz von NLP-Reduce machen zu können, wurden eine Reihe von Frage-Antwort-Paaren automatisiert von NLP-Reduce verarbeitet und die erzeugten Resultate bewertet. Die Ergebnisse dieser Tests können in Kapitel 3 nachgelesen werden. Instruktion zum Starten der Evaluation findet sich in Anhang A.3.

2.3 Zusammenfassung

Bei der Konzeption von NLP-Reduce wurde versucht, soweit wie möglich ohne linguistische Analysen auszukommen. Es wurden lediglich zwei Vertreter der computerlinguistischen Werkzeuge implementiert. Dies ist einerseits die Verwendung der Stopwortlisten und andererseits die Anwendung des Stemming-Algorithmus von Porter.

Kapitel 3

Evaluation

Um aussagekräftige Werte über die Leistungsfähigkeit von NLP-Reduce zu erhalten, wurden verschiedene Testläufe durchgeführt. Nach einer kurzen Einführung in die verwendeten Techniken der Evaluation, wird der Ablauf der Tests genauer beschrieben und erklärt.

3.1 Bewertungskriterien

Die beiden am meisten verwendeten Messgrössen für die Performanz von Informationswiedergewinnungs-Systemen[1] sind „Precision“ und „Recall“. Dabei bezeichnet die *Precision*[2] die qualitative Güte eines Suchlaufs während *Recall*[3] eine eher quantitative Bewertung der Performanz eines Suchsystems zulässt (siehe dazu auch [13]).

Precision: Die Präzision der durch eine Suche gefundenen Resultate gibt Auskunft darüber, wie gross der Prozentsatz der relevanten Resultate unter der gesamten Anzahl gefundener Ergebnisse einer Suche ist. In der Schreibweise der Mengenlehre lässt sich der Wert von *Precision* wie folgt errechnen:

$$precision = \frac{|\{Relevant\}| \bigcap |\{Gefunden\}|}{|\{Gefunden\}|}$$

Mathematisch ergibt sich dafür folgende Formel:

$$precision = \frac{Anzahl - relevanter - gefundener - Resultate}{Gesamtanzahl - gefundener - Resultate}$$

Recall: Dieser Wert beschreibt die Vollständigkeit der gefundenen Resultate. Er ergibt sich aus dem Anteil relevanter Resultate in der Lösungsmenge im Verhältnis zur Anzahl relevanter Einträge in den durchsuchten Daten. Die Formeln in der Schreibweise der Mengenlehre beziehungsweise der Algebra definieren sich wie folgt:

[1] Systeme zur Informationswiedergewinnung werden meist mit dem englischen Begriff *Information Retrieval (IR) Systems* bezeichnet.

[2] engl. precision = Präzision

[3] engl. recall = Ausbeute

$$recall = \frac{|\{Relevant\}| \bigcap |\{Gefunden\}|}{|\{RelevantInDB\}|}$$

$$recall = \frac{Anzahl - relevanter - gefundener - Resultate}{Gesamtanzahl - relevanter - Eintraege - in - der - Datenbank}$$

Sowohl *precision* wie auch *recall* haben im Idealfall den Wert 1. Das bedeutet, dass sowohl alle relevanten Daten gefunden wurden (*recall*) und gleichzeitig alle gefundenen Resultate zur Menge der relevanten Daten gehören (*precision*).

3.2 Testdaten

Die Ontologien, welche für die Evaluation verwendet wurden, stammen aus den Testdaten der *Machine Learning Research Group* an der Universität von Texas in Austin unter der Leitung von Raymond J. Mooney (siehe dazu auch [5]). Da Mooney und seine Leute mit Prolog-Programmen experimentieren, sind auch deren Datenbestände in Prologdateien abgelegt. Es existieren jedoch auch OWL-Ontologien mit denselben Daten. Bei der Extraktion dieser Datensätze wurden nicht alle von *RDF* und *OWL* unterstützen Funktionen verwendet. Da NLP-Reduce die semantischen Informationen der durchsuchten Ontologien verwenden kann, wurden fehlende Metainformationen manuell hinzugefügt. So wurden beispielsweise die transitive Eigenschaft „geo:isIn“ oder das inverse Attribut „geo:hasCity“ nachträglich in die Ontologie der Geografiedaten eingefügt.

3.3 Manuelle Korrekturen

Für die Auswertung wurden verschiedene Anpassungen an den Frage-Antwortdateien manuell durchgeführt. So wurden beispielsweise vorgegebene Antworten, die eindeutig als falsch erkannt und auf einfache Art und Weise korrigiert werden können, manuell angepasst. Ein Beispiel dafür bietet die Frage „How many cities are in montana?“. In den vorgegebenen Antworten ist für diese Antwort die Nummer „2“ vermerkt. In der Ontologie befinden sich jedoch nebst den zwei Instanzen „geo:billingsMt“ und „geo:greatFallsMt“ (die der Klasse „geo:City“ angehören), auch ein Eintrag mit dem Bezeichner „geo:-helenaMt“ der der Klasse „geo:Capital“ angehört. Über die Inferenzregeln kann NLP-Reduce - im Gegensatz zu Mooneys System - die Hauptstadt von Montana „Helena“ jedoch auch als „Stadt“ erkennen. Somit wurde in der Antwortdatei die Zahl „2“ durch die korrekte Anzahl Städte „3“ ersetzt.

3.4 Ignorierte Fragen

Leider sind die von Mooney vorgegebenen Antworten nicht in jedem Fall korrekt. So steht als Antwort auf die Frage „What is the length of the longest river in the usa?“ zum Beispiel folgender Text in der Antwortdatei: „451,459,462,-483,492,523,541,579,...“. Eine Frage nach „einem“ längsten Fluss kann natürlich

nicht mehrere unterschiedliche Resultate ergeben. Fragen, deren Antworten aufgrund von zu hohem Aufwand nicht manuell korrigiert werden konnten, wurden in eine Liste mit zu ignorierenden Fragen aufgenommen. Auch Fragen welche in die Gruppe der „Nicht unterstützen Fragetypen“[4] gehören, wurden in die erwähnte Liste geschrieben, um die Resultate nicht zu verfälschen.

3.5 Ablauf der Tests

Die automatisierte Evaluation läuft in zwei Phasen ab. In einem ersten Schritt werden alle Fragen und Antworten zur jeweiligen Ontologie eingelesen. In der zweiten Phase werden darauf die Fragen abgearbeitet und von NLP-Reduce beantwortet. Die generierten Antworten werden dann mit denjenigen der Ausgangsdatei verglichen. Aufgrund dieses Vergleichs werden die Berechnungen für *precision* und *recall* durchgeführt. Im folgenden werden diese beiden Phasen etwas genauer erläutert. Aus Günden der höheren Geschwindigkeit wurden die folgenden Tests ohne die Verwendung der Wörterdatenbank WordNet durchgeführt.

3.5.1 Einlesen der Fragen und Antworten

Die Frage-Antwort-Dateien enthalten pro Frage jeweils zwei Zeilen: Die erste beinhaltet eine natürlichsprachliche Frage welche an das System gestellt werden soll. Die zweite Zeile beinhaltet eine durch Kommas getrennte Liste, deren Inhalte der korrekten Antwort entsprechen.

Zur Zeit unterstützt NLP-Reduce eine Reihe von Fragetypen noch nicht. Um diese Fragen auszuschliessen werden Listen[5] mit jenen Fragen geführt, die in die Kategorie der nicht unterstützen Fragetypen gehören. Eine Auflistung dieser Fragetypen findet sich in Abschnitt 4.5.

3.5.2 Generieren der Antworten und Bewertung

Wie in Abschnitt 2.2.6 beschrieben, generiert NLP-Reduce auf eine Anfrage eine SPARQL-Anfrage, welche keinerlei Hinweise darauf gibt, welches diejenigen Daten sind, für die sich der Benutzer interessiert. Es werden lediglich alle unbekannten Variablen aufgelöst, die in der generierten SPARQL-Anfrage vorkommen. In den vorgegebenen Antworten steht jedoch pro Eintrag in der Antwortliste nur eine mögliche Lösung zur Verfügung. Aus diesem Grund kann nicht ein eins-zu-eins Vergleich von Soll- und Istzustand durchgeführt werden. Ein Beispiel:

> Auf die Frage „how big is texas?“ steht in der Frage-Antwort-Datei genau eine mögliche Antwort: „266807.0“. NLP-Reduce generiert auf diese Anfrage eine SPARQL-Anfrage und nach der Ausführung derselben erhält der Benutzer eine Tabelle mit folgenden Einträgen:

[4] Siehe dazu auch Abschnitt 4.5.

[5] Diese Listen wurde für jede Ontologie, mit der NLP-Reduce evaluiert wurde, manuell erstellt.

State	City
geo:illinois	geo:skokieIl
geo:illinois	geo:peoriaIl
geo:illinois	geo:elginIl
geo:illinois	geo:oakLawnIl
geo:illinois	geo:jolietIl
geo:illinois	geo:waukeganIl
geo:illinois	geo:champaignIl
geo:illinois	geo:auroraIl
geo:illinois	geo:ciceroIl
geo:illinois	geo:evanstonIl
geo:illinois	geo:rockfordIl
geo:illinois	geo:arlingtonHeightsIl
geo:illinois	geo:springfieldIl
geo:illinois	geo:chicagoIl
geo:illinois	geo:decaturIl

Tabelle 3.1: Antwort von NLP-Reduce auf „give me all cities in illinois."

State	StateArea
http://www.mooney.net/geo#texas	266807

Aufgrund dessen wird bei der Antwortgenerierung für jede Spalte der Antwort eine eigene Liste mit möglichen Antworttexten generiert. Jede dieser Listen wird mit der Soll-Liste verglichen, und jene mit den besten Werten für *precision* und *recall* wird für die Gesamtbewertung weiterverwendet. Ein weiteres Problem beim Vergleich der Antworten betrifft unvollständig gestellte Fragen. Die Benutzeranfrage „Give me all cities in Illinois." gibt keinerlei Hinweise darauf, für welche Detailinformation der Städte sich der Benutzer interessiert. NLP-Reduce gibt in diesem Fall lediglich Resultate in Form der Tabelle 3.1 als Antwort[6].

Obwohl ein menschlicher Benutzer in diesem Fall die Antwort auf seine Frage aus der Tabelle lesen könnte, beinhaltet diese Antwort nicht direkt die vom Anwender gesuchte Antwort. Die Bezeichungen der Form „geo:[Städtenamen][Bundesstaat]" müssen nämlich nicht in jedem Fall so klar gewählt sein. Da es sich bei diesen *URIs* lediglich um Bezeichner handelt, könnten diese auch durch Zahlen oder Buchstabencodes ersetzt werden. Die eigentliche Information, welche vom Benutzer gesucht wird, ist der Wert des Literals „geo:label", welches für alle diese Städte in der Ontologie abgelegt ist. Aus diesem Grund werden bei der Antwortgenerierung nicht bloss die gefundenen *URIs*, sondern über einen zuvor erstellen Index, der jeweilige Bezeichner und in Klammern dahinter sämtliche Detailinformation des jeweiligen Eintrags in die Antwortlisten geschrieben. Am Beispiel der Stadt „Chicago" ergäbe dies:

```
http://www.mooney.net/geo#chicagoIl(3005172|chicago)
```

[6] Aus Gründen der Übersicht wurde der Text „http://www.mooney.net/geo#" durch „geo:" abgekürzt.

Evaluation	geo.owl	restaurant.owl
Total Fragen	879	251
Ignorierte Fragen	412	48
Verarbeitete Fragen	467	203
Beantwortete Fragen	308	192
Prozentsatz	66.0%	94.6%
Precision	70.7%	67.7%
Recall	76.4%	69.6%

Tabelle 3.2: Übersicht Evaluation

Für den Fall, dass auf eine Anfrage mehr als eine SPARQL-Anfrage generiert wird, werden die Antwortlisten der einzelnen Anfragen separat ausgewertet. Für die Bewertung der Antworten wird danach die durchschnittliche Bewertung aller Teilbewertungen weiterverwendet.

3.6 Resultate für geo.owl

Diese Ontologie enthält Informationen über geografische Tatsachen in den *Vereinigten Staaten von Amerika*. Die Fragedatei für diese Ontologie beinhaltet 879 Fragen. Von diesen wurden 412 als nicht-beantwortbar aussortiert (Siehe dazu Abschnitt 4.5). Von den verbleibenden 467 Fragen konnte für 308 eine SPARQL-Anfrage generiert werden. Dies entspricht 35% der ursprünglichen und 66% der beantwortbaren Fragen. Die Werte für *Precision* und *Recall* entsprechen dabei 70.7% beziehungsweise 76.4%[7]. Eine Übersicht findet sich in Tabelle 3.2.

3.7 Resultate für restaurant.owl

Noch bessere Resultate konnten unter Verwendung der Ontologie „restaurant-.owl“ erreicht werden: NLP-Reduce konnte für 192 von 203 Fragen eine Antwort finden. Dies entspricht 94.6% der verarbeiteten Fragen. Jedoch mussten auch aus dem Antwortset für „restaurant.owl“ 48 Fragen als „nicht verarbeitbar“ deklariert, und somit ignoriert werden. Die Grafik in Abbildung 3.1 zeigt die Vehältnisse von Precision und Recall. Auch hier können durch NLP-Reduce gute Ergebnisse erzielt werden. Sowohl Precision wie auch Recall befinden sich für beide Ontologien über der 65% Marke.

3.8 Laufzeitverhalten

Bei der Entwicklung von NLP-Reduce wurde darauf geachtet, dass die Beantwortung der Fragen schnell durchgeführt werden kann. Aus diesem Grund

[7] *Precision* und *Recall* wurden nur für die tatsächlich beantworteten Fragen errechnet. Bei Fragen auf die keine Antwort gefunden werden konnte, entsprechen diese Werte immer 0%.

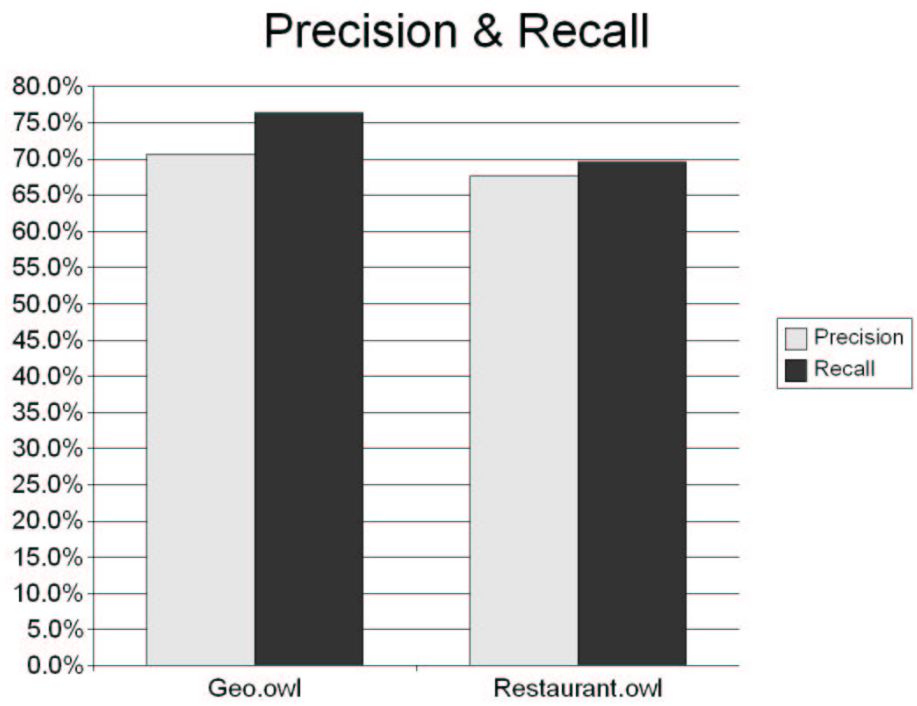

Abbildung 3.1: Precision & Recall

werden beim Start der Applikation viele Daten in Indizes abgelegt, und dadurch im Arbeitsspeicher der Maschine für den schnellen Zugriff bereitgestellt (siehe dazu auch 2.2.4). Die Evaluation wurde auf einem Notebook mit einem 1.6GHz Prozessor und 1024MB Ram durchgeführt. Obwohl dieser Rechner über viel Leistung verfügt, dauert das Laden einer Ontologie je nach Grösse 2-4 Minuten. Die Beantwortung der Fragen ist danach jedoch nur noch mit kurzen Wartezeiten verbunden (je nach Anzahl Wörter der Frage zwischen 2-20 Sekunden). Der Speicherverbrauch zur Laufzeit ist jedoch sehr hoch. So wurden die 512 Megabyte zugewiesener Speicher (siehe dazu auch Anhang A.3) vor allem bei der Evaluation voll ausgenutzt. Diese Zahlen und Werte könnten insbesondere durch die Neuimplementation in einer Prorammiersprache mit effizienterer Speicherverwaltung (z.B. C++) verbessert werden.

3.9 Zusammenfassung

Für die zwei Ontologien „geo.owl“ und „restaurant.owl“ erreicht NLP-Reduce gute Ergebnisse. Angesichts der Tatsache, dass NLP-Reduce ohne jegliche linguistische Werkzeuge auskommt, und sich bei der Antwortsuche lediglich der in den Fragesätzen vorkommenden Stichworte bedient, sind die Resultate sogar sehr erfreulich.

Erstaunlich ist auf den ersten Blick lediglich, dass sowohl Precision wie auch Recall eher niedrig sind. Da NLP-Reduce intern formale Queries verwen-

det, würde man bessere Resultate erwarten. Dies hat jedoch logische Gründe. Beispielsweise sind die Werte für Precision und Recall automatisch 0%, wenn bei der Frage nach einer Anzahl nicht die exakte Menge an Resultaten gefunden werden konnte. Wenn also auf die Frage „How many chinese restaurants are there in the Bay Area?“ anstelle von 1016, 1047 Restaurants gefunden werden, wird dies mit einem sehr schlechten Wert (0%) bestraft. Ein weiterer Grund für die schlechten Werte von Precision und Recall sind Anfragen, die aus irgendwelchen Gründen inkorrekte SPARQL-Anfragen generieren. Auch hierzu ein Beispiel: In der Frage „Where is a good place in the Bay Area for chinese food?“ befindet sich der Name eines Chinesischen Restaurants. In der Stadt „San Leandro“ gibt es ein Lokal mit Namen „Chinese Food“. NLP-Reduce wird also über SPARQL versuchen ein Restaurant mit Namen „Chinese Food“ zu finden, das sich in der Region „Bay Area“ befindet. Diese Suche liefert jedoch keine Resultate. Somit sind auch die Precision und Recall Werte für diese Frage 0%. In der Endbewertung ziehen diese Zahlen den Durchschnitt stark nach unten.

Natürlich muss die Performanz eines Suchsystems für den täglichen Gebrauch noch um einiges höher sein, für weitere Entwicklungen stellt NLP-Reduce jedoch bereits in seiner heutigen Form eine gute Basis dar (Siehe dazu auch Katpitel 5).

Kapitel 4

Beschränkungen

Während der Planung und der Implementierung von NLP-Reduce mussten eine Vielzahl von technischen Problemen gelöst oder umgangen werden. In diesem Kapitel werden die wichtigsten erläutert sowie mögliche Lösungsansätze diskutiert.

4.1 Qualität der Daten

Die Performanz von Suchmaschinen, die versuchen, die Struktur von Ontologien für die Antwortsuche auszunützen, hängt in einem sehr hohen Masse von den zugrundeliegenden Daten ab. Deshalb ist auch die Leistung von NLP-Reduce stark von der Qualität der Ontologien abhängig.

Weltwissen: RDF und OWL bieten eine Vielzahl an Möglichkeiten, um die gespeicherten Informationen mit Metadaten oder Weltwissen anzureichern. So können für die Eigenschaften und Objektklassen Metaeigenschaften und Beziehungen definiert werden (Siehe dazu auch Abschnitt 2.2.3). Dazu nochmals ein Beispiel:

> Das Restaurant „Bud's Ice Cream & Yogurt" befindet sich in der Stadt „Belvedere Tiburon". Dies wird über die Eigenschaft „restaurant:isInCity" definiert. Die Stadt „Belvedere Tiburon" liegt im Landkreis (engl. *County*) „Marin County" und dieser wiederum in der Region „Bay Area". Die Objekteigenschaften „restaurant:isInCounty", „restaurant:isInCounty" und „restaurant:countyIsInRegion" werden als „Untereigenschaften" von „restaurant:isIn" zugeordnet. Diese Obereigenschaft wird nun als transitiv deklariert. Aufgrund dieser Informationen kann über Deduktion neue Information generiert werden: Bei einer Suche nach Gaststätten in der Stadt „Belvedere Tiburon", in der Region „Bay Area" oder im Landkreis „Marin County" wird nun jeweils das Lokal „Bud's Ice Cream & Yogurt" gefunden werden, obwohl nicht explizit definiert wird, dass sich das Restaurant in einem Landkreis oder einer bestimmten Region befindet.

Die oben definierte Eigenschaft „restaurant:isIn“ wird in *OWL* Objekteigenschaft genannt. Die Eigenschaften, welche eine Untergruppe einer anderen Objekteigenschaft darstellen, werden mit dem Schlüsselwort „rdfs:subPropertyOf“ als solche definiert. Es wird also eine Beziehung zwischen zwei Objekteigenschaften festgelegt, aus der neue Informationen abgeleitet werden können. Neben den im obigen Beispiel verwendeten Beziehungen und Eigenschaften existieren in *RDF* und *OWL* noch verschiedene andere Möglichkeiten, um die erfassten Informationen mit *Weltwissen* anzureichern. So können Objekteigeschaften nicht nur transitiv, sondern auch symmetrisch oder invers zu einer anderen Objekteigenschaft sein. Weiter gibt es neben den Untereigenschaften („rdfs:subPropertyOf“) auch Unterklassen („rdfs:subClassOf“). Für eine vollständige Liste siehe auch [1]. All diese Metainformationen werden von NLP-Reduce verwendet, um Benutzeranfragen zu beantworten. Glücklicherweise muss nicht in NLP-Reduce selbst definiert werden, wie mit diesen Eigenschaftscharakteristika zu verfahren ist: Dies übernimmt der Reasoner von Pellet. NLP-Reduce macht sich die dadurch generierte Information jeoch über die Verwendung von SPARQL zu Nutze. Das heisst, dass mit der Erweiterung der Reasonerfunktionalität und mit der Weiterentwicklung von OWL und RDF auch die Performanz von NLP-Reduce zunimmt.

4.2 Stopwortlisten

Da sich NLP-Reduce möglichst nicht mit der grammatikalischen Struktur von Sätzen befassen will, werden eingegebene Fragen an zwei Stellen im Programm durch *Stopwortlisten* gekürzt[1]. Dies geschieht zum einen während der Vorverarbeitung der Benutzeranfragen[2] (Abschnitt 2.2.5), und zum anderen beim Entfernen unvollständiger Lösungsvorschläge (Abschnitt 2.2.6). Bei letzterem werden alle für die Beantwortung der Frage unwichtigen Wörter aus der Benutzeranfrage gelöscht. Doch die Unterscheidung zwischen für die Beantwortung *wichtigen* und *unwichtigen* Wörtern ist eine nicht-triviale Aufgabe. Ein Wort, welches im einen Fall komplett irrelevant sein kann, kann bei einer anderen Anfrage die einzige Verbindung zwischen den verschiedenen Teilen einer Frage darstellen. Die Stopwortlisten, die verwendet werden, sind lediglich in englischer Sprache verfasst. Somit können zur Zeit auch nur Ontologien mit englischsprachigen Inhalten zufriedenstellend verarbeitet werden.

Der Ansatz, alle Wörter aus der Frage zu löschen, die nicht in den zugrundeliegenden OWL-Ontologien gefunden werden können, kann bei kleineren Ontologien noch funktionieren, versagt aber schon bald aus verschiedenen Gründen. Die Chance für das Vorhandensein eines Wortes im Lexikon steigt mit der Grösse der Ontologie oder wenn die Sprachdaten mit weiteren Datenbanken (z.B. WordNet) angereichert werden, enorm an. Noch schlechtere Resultate erzielt man mit dieser Lösung, wenn gleichzeitig mehrere Ontologien eingelesen und dadurch die Menge der Wörter im Lexikon um ein Vielfaches vermehrt wird. Des Weiteren sollte das Ziel einer NLP-Suchmaschine sein, nicht nur die gesuchten Resultate bei „korrekt“ gestellten Fragen zu finden,

[1] Stopwortlisten sind Listen mit Wörtern, die zur Geschwindigkeitsoptimierung beim Suchvorgang ignoriert werden.

[2] Hier wird beispielsweise der Artikel „the“ aus den Benutzeranfragen gelöscht.

sondern beim Nichtauffinden von Lösungen dies auch so zurückzumelden. Dies wird natürlich verunmöglicht, wenn einfach alle „unbekannten“ Wörter aus einer Anfrage gestrichen werden.

Die bei NLP-Reduce verwendeten Stopwortlisten basieren auf den Erfahrungen, die bei der Entwicklung des Programmes gemacht wurden. Sie wurden also heuristisch erfasst.

4.3 WordNet

Wie in Abschnitt 2.2.4 beschrieben, können die Synonyminformationen der Ontologien durch die Synonymdatenbank *WordNet* erweitert werden. Für die Anwendung in NLP-Reduce hat sich diese Erweiterung jedoch weniger gewinnbringend erwiesen als ursprünglich vermutet. Da die meisten Texte der Synonyminformationen aus mehreren Wörtern bestehen und jeweils für alle Wortkombinationen in WordNet Synonyme gesucht wurden, ergaben sich dadurch enorm grosse Indizes im Lexikon. Die erhaltene Mehrinformation ist jedoch zu gering, als dass deswegen die massiv höheren Laufzeiten[3] in Kauf genommen werden könnten.

4.4 Sprachmarkierungen

Die Spezifikation von OWL ermöglicht es dem Ersteller von Ontologien, Literale mit Sprachinformation anzureichern. So kann bei jedem Literal genau bestimmt werden, in welcher Sprache der Inhalt verfasst wurde. Das Literal

'mount mckinley'@en

wurde demnach in englischer Sprache erfasst. Leider werden diese Sprachtags vom REGEXP-Prozessor von Jena nicht sauber übersetzt. Das heisst, dass mit *Regular Expressions* nur Literale ohne diese Sprachtags gefunden werden können. Da NLP-Reduce teilweise von der Verwendung von Regular Expressions abhängt, mussten somit zuerst alle Sprachtags aus den zugrundeliegenden OWL-Dateien entfernt werden.

4.5 Nicht unterstütze Fragetypen

NLP-Reduce unterstützt zur Zeit eine Reihe von Fragetypen noch nicht. Im Folgenden wird aufgezeigt, aus welchen Gründen diese nicht verarbeitet werden können.

Aggregatfunktionen: Fragen welche in der SPARQL-Anfrage Aggregatfunktionen[4] verlangen, können nicht verarbeitet werden, weil SPARQL diese noch nicht unterstützt.

[3] Durch die grossen Indizes des Lexikons stiegen die Zeiten, welche für die Durchsuchung dieser gebraucht wurden, enorm an.

[4] Aggregatfunktionen fassen eine Menge von Daten zu einem einzigen Wert zusammen. Beispiele dafür wären die Funktionen Summe, Maximum, Minimum, Durchschnitt oder die Zählfunktion.

Vergleichende Fragen: Damit sind Fragen gemeint, welche in irgendeiner Form Vergleichsoperatoren benötigen. Ein Beispiel dafür ist: „Give me all cities bigger than Wyoming“. Die Beantwortung dieser Fragen bedingt, dass der Fragesatz - wenn auch nur in geringem Masse - syntaktisch analysiert werden muss. Zur Zeit verzichtet NLP-Reduce jedoch auf jegliche linguistischen Analysen, und somit können vergleichende Fragen nicht verarbeitet werden.

„Reflexive“ Eigenschaften: Eigenschaften, welche zwei Instanzen des gleichen Konzepts verbinden, werden zur Zeit noch nicht unterstützt. Ein Beispiel für eine „reflexive“ Eigenschaft ist „geo:borders“. Diese Objekteigenschaft verbindet zwei Instanzen der Klasse „geo:State“ (ein Staat grenzt an einen anderen). Auch für die Beantwortung dieser Fragen wäre eine minimale Analyse des Fragesatzes von Nöten und somit werden auch diese noch nicht unterstützt.

4.6 Zusammenfassung

NLP-Reduce hängt in hohem Masse vom Funktionsumfang der zugrundeliegenden Technologien ab. Dies bringt sowohl Vorteile als auch Nachteile mit sich: Einerseits kann NLP-Reduce vom steigenden Funktionsumfang von Pellet, OWL, RDF und SPARQL profitieren, andererseits limitieren deren Mankos auch die Möglichkeiten von NLP-Reduce. Da zur Zeit noch konsequent auf eine linguistische Analyse der Fragesätze verzichtet wird, werden verschiedene Fragetypen noch nicht unterstützt. Hier ist sicherlich noch Erweiterungspotenzial vorhanden.

Kapitel 5

Künftige Arbeiten

Im folgenden werden Ideen und Erweiterungen für NLP-Reduce vorgestellt, die im Rahmen dieser Diplomarbeit aus zeitlichen oder technischen Gründen nicht verwirklicht werden konnten.

5.1 Interaktive Erweiterungen

NLP-Reduce arbeitet zur Zeit noch ohne erweiterte Benutzerinteraktion. Das heisst, ein Anwender gibt seine Frage ein und erhält darauf die errechneten Resultate. Denkbar wäre jedoch auch, dass sich NLP-Reduce interaktiv mit dem Benutzer „unterhält". So wäre es zum Beispiel möglich, die erfassten Synonyminformationen (Abschnitt 2.2.3) durch Rückfragen beim Benutzer zu erweitern. Falls also zum Beispiel keine oder zu viele SPARQL-Anfragen generiert würden, müsste NLP-Reduce dem Benutzer einer Auswahl an vorhandenen Objekt- und Datentypeigenschaften zur Auswahl unterbreiten. Die somit gesammelten Informationen könnten darauf wieder in die Ursprungsontologie integriert werden.

5.2 Grafische Oberfläche (GUI)

Die grafische Benutzerschnittstelle ist zur Zeit noch sehr einfach gehalten. Wünschenswert wären sicherlich weitere Optionen wie:

1. Popup-Menüs, die dem Benutzer Detailinformationen zu einzelnen *URIs* darstellen können.
2. Die Möglichkeit, Resultate zu speichern.
3. Dynamisches Hinzufügen und Entfernen von Ontologien.
4. Exportfunktionen in Formate wie *CSV* oder *XML*

Des Weiteren würde sich auch ein Zugang über eine Webseite für die Verwendung von NLP-Reduce anbieten. So könnte die Applikation von einer breiteren Benutzergruppe gleichzeitig getestet und verwendet werden.

5.3 Sprachunterstützung

Wie in Abschnitt 4.2 beschrieben, sind die von NLP-Reduce verwendeten Stopwortlisten nur in englischer Sprache verfasst. Diese müssen, um auch Ontologien in anderen Sprachen verarbeiten zu können, für die jeweiligen Sprachen erweitert werden.

5.4 Informationen der Ontologien

Mit dem steigenden Funktionsumfang von OWL und den in Verbindung mit SPARQL zur Verfügung stehenden Reasoner-Programmen werden in Zukunft immer mehr Informationen direkt in den Ontologien abgelegt und über NLP-Reduce verwendet werden können. So können neue Klassen und Eigenschaften auschliesslich aufgrund bereits bestehender Definitionen bestimmt und abgefragt werden. Diese Art der Erweiterung von Ontologien ist deshalb sehr interessant, weil keine Instanzen neu erfasst, sondern lediglich die für alle Inhalte einer Ontologie gültigen Klassen- und Eigenschaftsdefinitionen auf den neuesten Stand gebracht werden müssen.

5.5 Computerlinguistik

Wie in Abschnitt 4.5 beschrieben, können verschiedene Typen von Fragen zur Zeit ohne computerlinguistische Satzanalysen nicht verarbeitet werden. Hier ist zu prüfen, ob durch die Verwendung von robusten Analysealgorithmen die Performanz von NLP-Reduce gesteigert werden kann.

5.6 Zusammenfassung

Obwohl NLP-Reduce mit seinem aktuellen Funktionsumfang schon gute Resultate erzielt, gibt es verschiedene Ansätze, um die Leistungsfähigkeit von NLP-Reduce zu steigern. Diese reichen von einfachen Erweiterungen der grafischen Benutzeroberfläche bis zur Integration von computerlinguistischen Algorithmen.

Kapitel 6

Schlussfolgerungen

Die Konzeption und prototypische Entwicklung von NLP-Reduce wurde im Rahmen dieser Diplomarbeit im Sinne einer Machbarkeitsstudie durchgeführt. Die erreichten Resultate sind wohl ansehnlich, jedoch reicht die Performanz von NLP-Reduce zum heutigen Zeitpunkt noch nicht für den produktiven Einsatz. Im folgenden wird das Erreichte kurz reflektiert und über die Zukunft von NLP-Reduce sinniert.

6.1 Reflexion des Erreichten

Das Ziel der vorliegenden Diplomarbeit war es, ein System zu entwickeln, das natürlichsprachliche Anfragen erlaubt, gleichzeitig jedoch die typischen Nachteile der Verarbeitung von natürlicher Sprache umgeht. Des Weiteren sollten verschiedene OWL-Ontologien verarbeitet werden können und ein Interface für die Benutzung durch gelegentliche Nutzer implementiert werden. Alle diese Ziele konnten erreicht werden. Die in der Evaluation erreichten Resultate sind gut und zeigen klar auf, dass die gewählte Strategie Potential für die Verwendung in weiteren Projekten aufweist.

6.2 Ausblick

Mit der Weiterentwicklung des Semantic Web werden Suchsysteme für dessen Datenbestände immer wichtiger. In Zukunft werden weitere Suchsysteme entwickelt und evaluiert werden müssen, um die semantischen Informationen für den Benutzer in einer angenehmen Form verwendbar zu machen. NLP-Reduce stellt dafür eine gute Basis dar. In Kapitel 5 wurden veschiedene Teilbereiche beschrieben, in denen noch Verbesserungspotential liegt.

Wie Christian Kaiser in seiner Arbeit „Ginseng - A Natural Language User Interface for Semantic Web Search“[10] schreibt, ist fraglich, ob die Suchsysteme der Zukunft wirklich natürliche Sprache werden verarbeiten müssen. Die meisten Anwender des heutigen Internet sind sich gewohnt, Inhalte über die Beschreibung durch Stichworte zu suchen. NLP-Reduce zwingt den Benutzer jedoch nicht zur Verwendung von natürlichsprachlichen Fragen: Da lediglich die in einer Frage vorkommenden Wörter für die Suche verwendet werden,

kann NLP-Reduce auch mit stichwortartigen Fragen umgehen und gleichzeitig die semantischen Informationen von Ontologien verwenden.

Literaturverzeichnis

[1] D. B. Brian McBride and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/2002/WD-rdf-schema-20021112/, 2002.

[2] G. K. u. J. J. C. Brian McBride. Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/, 2002.

[3] T. Burners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), May 2001.

[4] A. S. Eric Prud'hommeaux. Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/.

[5] M. L. R. Group. Machine learning research group - university of texas at austin. http://www.cs.utexas.edu/~ml/.

[6] T. R. Gruber. *A translation approach to portable ontology specifications. In: Knowledge Acquisition*. 1993.

[7] J. Heflin. Owl web ontology language: Use cases and requirements. http://www.w3.org/TR/webont-req/, 2004.

[8] J. Hendler. Frequently asked questions on w3c's web ontology language (owl). http://www.w3.org/2003/08/owlfaq.html, 2006.

[9] P. I. Androutsopoulos, G.D. Ritchie. Natural language interfaces to databases - an introduction, 2003.

[10] C. Kaiser. Ginseng - a natural language user interface for semantic web search, 2004.

[11] D. L. McGuinness and F. van Harmelen. Owl web ontology language: Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.

[12] M. Porter. The porter stemming algorithm. http://www.tartarus.org/martin/PorterStemmer/.

[13] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[14] W3C. Semantic web. http://www.w3.org/2001/sw/, 2006.

[15] Webseite. Java. http://java.sun.com/.

[16] Webseite. Jena - semantic web framework. http://jena.sourceforge.net/.

[17] Webseite. Pellet - an open-source java based owl dl reasoner. http://www.mindswap.org/2003/pellet/.

[18] Webseite. Wordnet - a lexical database for the english language. http://wordnet.princeton.edu/.

[19] Webseite. The world wide web consortium (w3c). http://www.w3.org/.

[20] Webseite. The dublin core metadata initiative. http://www.dublincore.org/, 2006.

[21] Wikipedia. Metainformationen. http://de.wikipedia.org/wiki/Metainformation, 2006.

Anhang A

Technische Informationen

A.1 Inhalt der CD-Rom

Zu dieser Arbeit wurde eine CD-Rom erstellt, welche dieses Dokument, alle verwendeten Daten und Sourcen in digitaler Form enthält. Die Tabelle A.1 gibt eine Übersicht über die Inhalte, die auf der CD-Rom abgelegt wurden.

Pfad	Inhalt
/	Im Wurzelordnet liegen drei Dateien: Zusfsg.pdf (Zusammenfassung in deutscher Sprache) Abstract.pdf (Zusammenfassung in englischer Sprache) Diplomarbeit-01-714-013.pdf (Dieses Dokument)
/JavaProject/	Dieser Ordner enthält das Eclipseprojekt mit sämtlichen Java-Sourcen und Javadoc-Dateien.
/Resources/	Dieser Ordner enhält die Ressourcen, auf die NLP-Reduce zurückgreift. Also alle verwendeten OWL-Ontologien sowie die Fragen und Antworten zu den Ontologien.

Tabelle A.1: Inhalt der CD-Rom

A.2 Verwendete Komponenten

Für die Entwicklung von NLP-Reduce wurden verschiedene Produkte von Drittanbietern verwendet. In Tabelle A.2 sind diese aufgelistet.

A.3 Kompilieren und Ausführen

Der komfortabelste Weg, um NLP-Reduce auszuführen, ist über die Entwicklungsumgebung Eclipse[1]. Der Eclipse-Projektordner befindet sich auf der CD-

[1] Der dieser Arbeit beiliegende Prototyp von NLP-Reduce wurde mit der Version 3.1 von Eclipse entwickelt.

Produkt	Url
Java	`http://java.sun.com/`
Jena	`http://jena.sourceforge.net/`
Pellet	`http://www.mindswap.org/2003/pellet/`
Wordnet	`http://wordnet.princeton.edu/`
JWNL	`http://jwordnet.sourceforge.net/`
Eclipse	`http://www.eclipse.org/`

Tabelle A.2: Verwendete Produkte

Rom im Verzeichnis „/JavaProject“. Alternativ kann NLP-Reduce auch über den direkten Aufruf in der Kommandozeile gestartet werden. Da die Indices des Lexikons sehr gross werden können, muss beim Aufruf der Applikation darauf geachtet werden, dass genügend Arbeitsspeicher zur Verfügung steht. Dies kann über die Paramter „-Xms512m -Xmx512m“ erreicht werden. Weitere Parameter werden benötigt um *Pellet* anzuzeigen, wo die Konfigurationsdateien zu finden sind. Der vollständige Aufruf für die Manuelle Ausführung von NLP-Reduce (unter Verwendung der Ontology „geo.owl“ im Verzeichnis „res“) sieht wie folgt aus[2]:

```
java -Djava.ext.dirs=lib _
-cp bin _
-Xms512m -Xmx512m _
-Dpellet.configuration=file:res/pellet.properties _
-Dlog4j.configuration=file:res/log4j.properties _
nlpreduce/NlpReduce res/geo.owl _
"give me all cities of illinois."
```

Die Befehlszeile, um die grafische Benutzeroberfläche zu starten, sieht analog dazu folgendermassen aus:

```
java -Djava.ext.dirs=lib _
-cp bin _
-Xms512m -Xmx512m _
-Dpellet.configuration=file:res/pellet.properties _
-Dlog4j.configuration=file:res/log4j.properties _
nlpreduce/NlpReduceGUI res/geo.owl
```

Der Evaluationsmodus von NLP-Reduce lässt sich über folgenden Programmaufruf starten:

```
java -Djava.ext.dirs=lib _
-cp bin _
-Xms512m -Xmx512m _
-Dpellet.configuration=file:res/pellet.properties _
-Dlog4j.configuration=file:res/log4j.properties _
nlpreduce/NlpReduceGUI res/geo.owl
```

[2]Es wird davon ausgegangen, dass das aktuelle Arbeitsverzeichnis „CDROM/JavaProject/“ ist. In der Datei „readme.txt“ im Wurzelordner des Javaprojekts wurden die drei Aufrufmethoden für die Verwendung per „Copy&Paste“ abgelegt.

Um die von *NLP-Recuce* generierten Antworten automatisiert bewerten zu können, muss sich in dem Verzeichnis, in dem sich die Ontologie befindet, auch je eine Datei mit Namen *#Dateiname-Ontologie#.answers.txt* beziehungsweise *#Dateiname-Ontologie#.ignore.txt* befinden.

A.4 Dokumentation des Sourcecodes

Im Ordner *#CDROM#/JavaProject/doc/javadoc/* befindet sich die vollständige Dokumentation zu den Java-Sourcen. Diese sind, wie auch sämtliche Kommentare und Variabelnamen, in englischer Sprache gehalten. Dies mit dem Ziel, dass die entwickelte Software auch von Programmierern ohne Deutschkenntnisse weiterentwickelt und verwendet werden können soll.

Anhang B

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis