UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A mini-thesis submitted for transfer from MPhil to PhD

Supervisor: Prof. David C. De Roure and Dr. Nicholas M. Gibbins
Examiner: Prof. Paul H. Lewis

How Semantic Web Technologies Might Improve Natural Language Querying Systems

by David R. Newman

May 25, 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A mini-thesis submitted for transfer from MPhil to PhD

by David R. Newman

Natural Language (NL) querying has an extensive history, the earliest systems were developed back in the 1960s. This reports reviews a number of these system developed between then and the present to gain understanding of how they work and what are their strengths and weaknesses. Relatively, Semantic Web (SW) technologies are a much more recent invention. These technologies provide mechanisms for storing and representing data in an innovative way, which may be of benefit to NL querying systems.

This report looks at the Comb*e*Chem project as an example of how SW technologies can be used and because it is a domain suitable for a NL querying system. Comb*e*Chem is however a complex domain, therefore an analogous domain around music albums has been developed. Using these two domains to exemplify, this report defines a conceptual framework for a NL querying system that uses SW technologies. It then describes the parts of this framework that have already been implemented, using screenshots to illustrate where applicable.

The report concludes by discussing the benefits of using SW technologies that have already been found. Such as the ability to more logically structure the data the NL querying system requires and to pull in data from disparate sources. It considers how SW technologies and NL querying share similar goals, i.e. both use machine-readable knowledge to produce a human readable output. However, a NL querying system goes further by mirroring this by taking a NL/human-understandable input and representing it in a machine-readable format. Finally this report sets out the work that is still to be done, using a rough timeline to illustrate the key milestones of this work.

Contents

List of Figures

Glossary

3Store A triplestore application developed at the University of Southampton

ANNIE A Nearly-New Information Extraction system

BPEL Businness Process Execution Language. A orcestration language that is part of Open Grid System Architecture.

Chat-80 A NLIDB that uses a database of Prolog statements.

CombeChem A project which had one of it main goals to generate an electronic lab book.

DAML DARPA Agent Markup Language. Was amalgamated with OIL to produce OWL.

DLG Directed Labelled Graph

GATE General Architecture for Text Engineering.

GINSENG Guided Input Natural language Search ENGine. An NL querying system that uses SW technologies.

GUI Graphical User Interface

IP Intellectual Property.

IRLS Intermediate Representation Language System. A system architecture that could be used to design a NL querying system

Janus A NLP system that uses Montague grammar framework.

Jena A triplestore application that allows inferencing through the use of models.

KnOWLer One of the first projects to produce an ontology for Wordnet (v1.7.1).

Kowari A triplestore application.

Lexicon A list of words with additional word-specific information.

Lifer The natural language processing part of the NLIDB Ladder.

Link grammar parser A rule-based parser developed at Carnegie-Mellon University.

Lunar One of the earliest NLIDBs for querying samples of moon rock.

MCF Meta Content Framework. Part of the inspration for RDF.

Minipar A principal-based NLP developed by Dekang Lin. A reduced form of Principar.

MusicBrainz A domain for making up musical album content.

MyExperiment A project to allow scientists to share research. Like MySpace shares photos/documents.

myTea A project that analogises the work done by bioinformaticians.

NL Natural Language

NLIDB Natural Language Interface to a DataBase

NLP Natural Language Parser

OIL Ontology Interchange Language. Was amalgamated with DAML to produce OWL.

OWL-S Web Ontology Language for Services. Language to semantically markup web services.

PC-PATR A unification-based grammar NLP.

Planes A NLIDB to find out information about the flights and maintenance records of aircraft.

Principar A principal-based NLP developed by Dekang Lin.

RDF Resource Description Framework. XML with explicitly defined namespaces. One of the main building block of the semantic web.

RDFS Resource Description Framework Schema.

RDQL Resource Description Query Language. A triplestore query language.

Redland A triplestore application.

RQL Resource Query Language. A triplestore query language.

Sesame A triplestore application.

SPARQL SPARQL Protocol and RDF Query Langauge. A common SW technologies querying language.

SQL Structured Query Language

Stanford parser A rule-based NLP developed at Stanford University.

SW Semantic Web

TNT parser A Head-driven Phrase Structure grammar NLP.

Triplestore SW technologies equivalent of a relational databases data structure.

TURTLE Terse RDF Triple Language

W3C World Wide Web Consortium. An organisation that defines standards for the web.

XML eXtensible Markup Language. Similar to HTML but for marking up real world objects and not just hypertext.

Chapter 1

Introduction

This report considers how Semantic Web (SW) technologies could be used in the design of a Natural Language (NL) querysing system. Can these technologies provide solutions to some of the inherent problems with NL querying systems? Can they make NL querying systems more powerful than they have been in the past?

Chapter 2 considers the pros and cons of NL querying systems as an interface. Through evaluating a number of previous systems, it determines the components required for a NL querying system and how they must be put together. Later in the chapter SW technologies are analysed to determine what features they provide and how they may be used a part of an NL querying system. The chapter concludes by considering a domain, the Comb*e*Chem project[1], that would benefit from a NL querying system and GINSENG, an early implementation of an NL querying system using SW technologies.

Chapter 3 takes what has been learnt from the previous chapter to propose a conceptual framework for a NL querying system that uses SW technologies, but firstly it considers the domains for which it will be tested, i.e. Comb*e*Chem and MusicBrainz[2]. It then provides a diagram, (see Figure 3.2), to illustrate the various components required and how the data should flow through the system from NL query input to human-readable answer output. It then discusses in detail how to go about building these components before summarising the inspiration behind the determined framework.

Chapter 4 explains how a number of the components in the previous chapter have been implemented. Providing screenshots where applicable to illustrate how the components that have been built to conform to the requirements defined in the conceptual framework.

Chapter 5 draws conclusions about what is required to make an effective NL querying system. It considers how SW technologies can be used to achieve this goal. The conclusion also considers the Comb*e*Chem project as a domain and how the research in

[1]http://www.combechem.org
[2]http://www.musicbrainz.org

this report might fit in to develop a system that can be used by the same users as the Comb*e*Chem and even myExperiment[3] projects. This chapter concludes by discussing how work will continue to develop and test the proposed NL querying system.

[3]http://www.myexperiment.org

Chapter 2

Background Research

Any attempt to design a new NL querying system, must consider the history of NL querying and the implementations and techniques that have already been created. This will help determine the benefits of a NL querying system and the inherent problems that have prevented their uptake from being greater. One of the main components of a NL querying system is its NL Parser (NLP) system. Evaluating the different flavours of these and how they fit into a system, is one of the most important considerations in the design of a NL querying system. Some natural language statements are more difficult to evaluate than others, therefore a NL querying system also needs a toolkit of techniques to assist its NLP system in understanding such statements.

In comparison to NL querying, SW technologies are in their infancy[1]. These technologies provide resources for representing, storing and querying data in a new way, that is quite different from existing relational databases, that use Structured Query Language (SQL).

Through reviewing a project implemented using SW technologies, i.e. the Comb*e*Chem project, it is possible to gain a better understanding of how these technologies work. It is not uncommon for these types of project to focus on one particular domain, in this case chemistry. This is a similarity that it shares with some of the more effective NL querying systems that have tended to focus on a single domain. This gives a early hint that the way SW technologies organises its data may benefit a focussed NL querying system. There have already been some early attempts to use SW technologies in building NL querying systems. These must be considered to evaluate their contribution, determine their shortfalls and consider how to integrate the SW technologies most effectively into a NL querying system.

[1]The lunar sciences natural language information system's final report (Woods 72) was published in 1972, whereas one of Berners-Lee's first presentation about the semantic web was not until 2002 (Berners-Lee 02).

2.1 Natural Language Querying

The principles of NL querying are not new. The concept of Natural Language Interfaces for Databases (NLIDBs) has been around since the late 1960s. Lunar, a NL system for querying a database of moon rocks was one of the first NLIDBs (Woods 72). A great number of NLIDBs that have been developed over the last 40 years have used SQL as the machine-readable format to query their database. Some NLIDBs use more novel approaches (Androutsopoulos 95).

Chat-80 uses a database[2] of Prolog statements to store logical relationships(Warren 82). These Prolog statements represent facts, e.g. country(France) implies France is a country and capital(France,Paris) implies Paris is the capital of France. These statements also represent rules such as exists(X,bird(X) & migrates(X)) implies that some birds migrate. The ability to store relationships with this logical format is important in increasing the potential for representing human concepts in a machine-readable way, which is an essential element required for any NL querying system to be successful.

2.1.1 NL Querying Pros and Cons

The first advantage of NL querying and the reason for its conception is to provide the user with an interface where they do not have to follow strict grammatical and often non-intuitive rules, that languages such as SQL require. Instead they allow a query to be expressed much more flexibly, in a language the user understands. A non-NL solution to this problem might be a form-based querying system and although it makes the generation of machine-readable queries trivial, it is liable to limit the number of queries that can be performed, making the data in the database of a lot less useful than it could potentially be. There are also some questions that form-based systems and Graphical User Interfaces (GUIs) struggle with, such as negation, quantification and those with temporal relations (Cohen 92).

Form-based systems may reduce the domain coverage but they may also facilitate quicker more accurate database querying. Depending on the domain, a form-based system may be more appropriate especially if query forms can be automatically generated based on the structure of the database.

NL querying also has other potential weaknesses, one of the most problematic of these is the opacity of domain coverage (Androutsopoulos 95). The output returned by a NL querying system may state that there is no answer, however this may mean one of several things:

[2]This is not a relational database rather a list of rules and facts that are read into the Prolog interpreter when Chat-80 is loaded.

1. The query submitted has either grammatical or spelling defects, so its meaning could not be understood.
2. The grammar and spelling of the query is correct but the NLP's database does not have a record of either the words or grammatical structure used.
3. The query's semantic meaning is understood but it still cannot be transcribed into a MR query.
4. No appropriate data has been recorded in the database for the query to find[3].

The latter two reasons occur because of lack of domain coverage whereas the first two are either due to bad user input or a poor NLP. If the user does not know why their query failed they are likely to try and re-phrase the query in the hope of getting it working. If after several attempts the user still does not get an answer they are liable to become frustrated and disillusioned with the system (Shneiderman 80). An NL querying system must therefore try to reduce this opacity by returning useful error messages, so that the user knows whether they are at error, the query is impossible or if there is just no data pertaining to that query. Lifer is capable of performing spelling corrections and provides specific error messages when queries fail (Hendrix 78). Lifer also uses a technique described as *ellipsis* that keeps a record of at least the previous question so it can try and work out what the current question is, when insufficient detail has been provided. E.g.

1. Who recorded the album Nevermind
2. London Calling

Through ellipsis the system will interpret the second question to be "Who recorded the album London Calling".

Anaphora is also a significant problem with natural language. Anaphora is basically where pronouns both personal and impersonal are used to refer to people or things that have been mentioned previously, e.g. John has a dog. *It* was given to *him* by *his* dad. There are three pronouns in this example, it, him and his because most people would probably assume that John is male, *him* and *his* can be interpreted as referring to John. As no reference is made to the gender of the dog, in general it would be assumed that *it* refers to the dog. ANNIE (Cunningham 06) uses a resolution algorithm to try to solve this problem, (see section 2.1.4.1).

The ability to perform NL querying may give the impression to the user that the system is intelligent with common sense and human reasoning abilities (Androutsopoulos 95).

[3]The open world assumption must be accepted in this case. I.e. Appropriate data may exist but it is not stored in the database.

This may give way to exaggerated disappointment with the system when it does not live up to these expectations.

Natural language may be in many cases the easiest way to express a question/query; however such systems are very error prone due the ambiguity of natural language (Cohen 92). Other forms of querying system do not suffer from this type of ambiguity. NLIDBs in the past have struggled to overcome this and the other problems described here but as NLPs and knowledge technologies continue to evolve it may be possible to minimise some of these disadvantages.

2.1.2 Natural Language Parsers and NLP Systems

There are a good number of NLPs to be found on the web. Many are designed by specialist university research groups such as the Stanford NLP group[4] and the Sheffield NLP group[5]. NLPs should not be confused with pattern-matching systems as these do not parse the natural language they just search the natural language for patterns. This is quite a shallow technique and although it can sometimes perform impressively, it can often make quite spectacular errors (Androutsopoulos 95)(Johnson 85). One of the main attributes of an NLP is the type of grammar it uses. There are two main types of grammar, syntactic and semantic. Syntactic grammars parse the natural language to classify each word as a part of speech, such as noun, verb, adjective, etc. They then usually build trees to describe how the words in a sentence are interlinked. E.g. A verb is linked to one noun that is its subject and one that is its object. Syntactic grammars can be roughly broken down into four sub-categories:

1. Rule-based
2. Principle-based
3. Unification-based
4. Head-driven phrase structure

It is difficult to neatly break down syntactic grammars into sub-categories as a number of grammars are considered to be specialisations of other grammars, as will become clear in sections 2.1.2.1 to 2.1.2.3.

2.1.2.1 Rule-based and Principle-based Grammars

The main difference between rule-based and principle-based grammars is that rules specify how to precisely generate sentences (Spenceley 92) whereas principles use constraints,

[4]http://www-nlp.stanford.edu/
[5]http://nlp.shef.ac.uk/

which provide well-formedness conditions that the sentences of the language should satisfy but do not specify how to generate the sentences of the language. Both rule-based and principle-based grammars have the ability to generate parse trees proving a sentence is valid. principle-based grammars have the advantage of also being able to generate proof for why a sentence is not valid.

2.1.2.2 Unification-based Grammars

Unification-based grammars take feature structures and unify them, if all corresponding register values in the two feature structures are the same. If a register value in one feature structure does not have a corresponding register value it is just copied to the unified feature structure, making a more specific structure (Allen 87). E.g. Consider the two feature structures:

(S **VERB** (VERB **ROOT** LOVE))

(S **VERB** (VERB **FORM** en)
  NUM {3s})

The first states that it must be made up of a verb with the root love. The second again states that it can be made up of a verb, this verb must be in the past participle form, (as represented by en) and this verb must be used in the third person singular. As these two feature structures intersect but the register values do not contradict, they can be unified as follows:

(S **VERB** (VERB **ROOT** LOVE))
        FORM en)
  NUM {3s})

This unified feature structure is more specific as it must now be made up of the past participle of the verb root love, (i.e. loved) and it must be used in the third person singular, e.g. she has been loved.

The main advantage of using a unification-based grammar is that it can maintain partial information about structures and return the same results independent of the order in which grammatical rules are applied (Allen 87). This feature is not guaranteed with a rule-based grammar. Unification-based grammars are basically rule-based grammars with greater formalism, therefore comparison with principle-based grammars is very similar to comparing rule-based grammars with principle-based grammars.

2.1.2.3 Head-driven Phrase Structure Grammars

Head-driven phrase structure grammars are based on generalised phrase structure grammars but provide a much simpler context free grammar, at the expense of a much more complex lexicon. This is achieved by using greater subcategorization on the heads of phrases. What makes head-driven phrase structure grammars interesting is they basically use frame-based representation for knowledge (Vogel 90), which is similar to what SW technologies do, (see chapter 2.2).

2.1.2.4 Semantic Grammars

Semantic grammars are different to syntactic grammars as they do not try to classify each word in a natural language sentence as a part of speech but rather to specific concepts. Often these specific concepts are made of more than one word. These concepts are then combined to build larger concepts, until it gets back to a specific sentence template that has been predefined. E.g. Take the question “Who recorded Thriller?”, this may generate the tree in Figure 2.1. The fact that the parse tree works back to a predefined

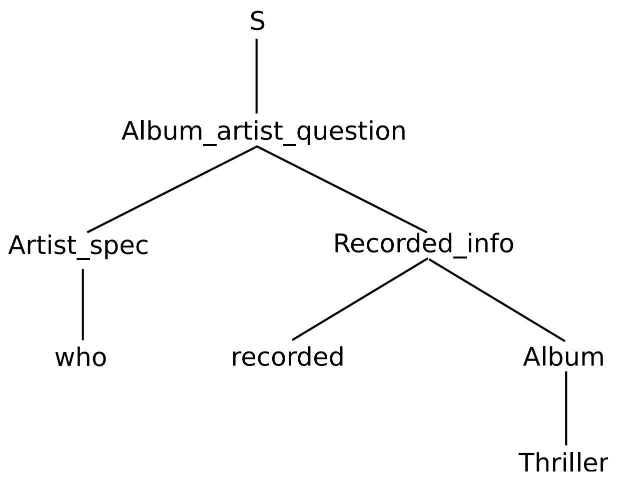


Figure 2.1: Parse Tree for a Semantic Grammar (Based on (Androutsopoulos 95) example)

question is the major flaw in semantic grammars, because as soon as the knowledge base is changed the grammar becomes useless. In well-defined domains with a reasonably small set of possible sentence/question templates semantic grammars work well, e.g. Planes (Waltz 78), which is just for querying about flight and maintenance records of aircrafts or Ladder. (Hendrix 78), which returns information about ships, such as their position, specification and even their complement.

To make semantic grammars useful generically would require being able to swap between knowledge bases/domains quickly and easily. The main problem with this is that the semantic grammar needs to be generated. This is very painstaking to do by hand but it is inevitable that at least some of this will have to be done manually. Tools need to be developed that can limit the amount of manual input required and automate those tasks that can be automated.

2.1.2.5 NLP Implementations

The Stanford parser[6] (Klein 03) and Link grammar parser[7] (Sleator 03) are good examples of NLPs with rule-based grammars. Principle-based grammar NLPs are slightly less common, Principar (Lin 94) and Minipar[8] (Lin 98), which is a smaller version of Principar are quite well known. PC-PATR [9] (McConnel 95) is one of the most well known unification-based grammar NLP. The TNT parser (Torisawa 00), is an example of a NLP that uses an head-driven phrase structure grammar. NLPs for semantic grammars are usually defined specifically for the system itself, as in the case of Planes and Ladder discussed in 2.1.2.4.

One of the problems that all these NLP implementations have had is how to handle finding more than one potential sentence structure. This is an inherent flaw in natural language because of its ambiguity, (see section 2.1.1). The simplest way that an NLP could deal with this is by returning all possible structures but generally a user will want an NLP package that at least returns the most probable structure, if not a weighted list of all possible structures. Various statistical and stochastic methods have been applied in NLPs to try to solve this, some NLP packages even use evolving machine-learning techniques to better tackle this problem.

2.1.2.6 NLP Systems

There is no necessity to use one only NLP in a NL querying system, it is not uncommon to use two or more NLPs to make an NLP system. Using two syntactic parsers that use different types of grammar may be useful, especially if they generate different parse trees. E.g. the Stanford parser (Klein 03) produces a very typical syntax-based parse tree, (see Figure 2.2), whereas Minipar (Lin 98) can produce an output that tries to describe how the words in a sentence are interlinked rather than trying to build them up into larger and larger structures until they are the complete sentence, (see Appendix A.1). This should give a greater richness to the representation making it possible for the NL querying system to generate a more accurate machine-readable / database language query. The extra richness given by using two NLPs is only useful if the two results can be reconciled, with simple sentences it should be possible to design a tool to do this with reasonable ease, as both NLPs will probably have found the same or similar structure. However, when a sentence is more complex this may not be the case. Therefore a

[6]http://ai.stanford.edu/~rion/parsing/stanford_viz.html
[7]http://www.link.cs.cmu.edu/link/index.html
[8]http://www.cs.ualberta.ca/~lindek/downloads.htm
[9]http://www.sil.org/pcpatr/manual/pcpatr.html

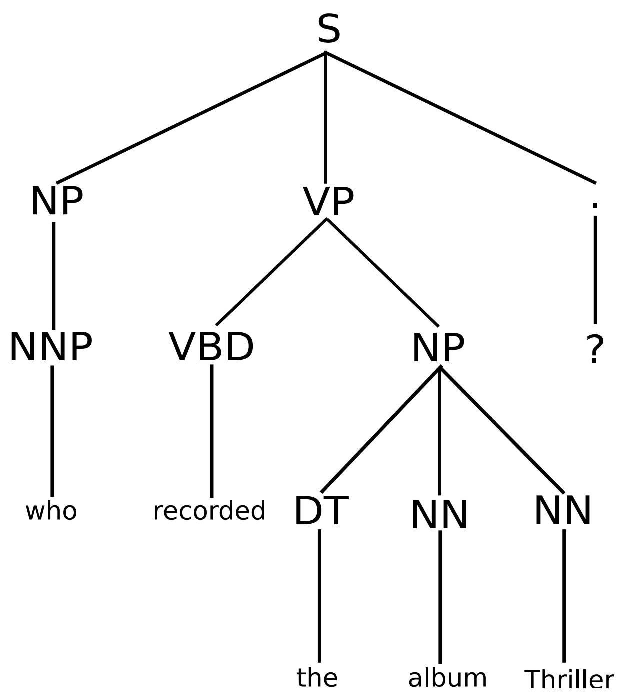


Figure 2.2: Typical Syntactic Grammar Parse Tree

reconciling tool would also need to be designed to evaluate whether two NLP outputs are reconcilable and if not decide which of the two outputs to use.

Using both a syntactic and semantic grammar parser in series should also improve the richness of the interpretation of the natural language. Several systems have been developed that have this hybrid nature. Janus (Hinrichs 88) is a good example of a system that uses several parsers/translators in series to produce an output that can be used to produce a query to a database.

Janus follows the Montague grammar (Dowty 81) framework and first uses a syntactic grammar NLP to produce an output that can then be translated to an English-oriented Formal Language (EFL). In turn this EFL, which is still ambiguous, can be made unambiguous using a domain model to disambiguate lexical meanings. Janus also uses a discourse model to perform ellipsis to overcome ambiguity as well. The output from this step produces a World-Model Language (WML) output. This output could then be mapped directly to a database language query.

A NLP system need not perform its translations as clean and tidily as Janus does; the intermediate outputs do not necessarily need to be so formal that they can be exported for use elsewhere. It may be necessary to extract certain components from a sentence and re-incorporate them later, as a parser/translator may not be able to handle them. E.g. The name of an musical album may well be a phrase or even a complete sentence. If this phrase interjects an NL query, it may make the sentence impossible to parse as it can no longer be mapped to a tree structure. Determining that there is a name of an album in a query and subsituting it for a token would avoid this problem, (see section

2.1.4). By having a symbiotic relationship between the NLP system and the rest of the NL querying system rather than such a distinct separation, it may be possible to achieve more.

2.1.3 Thesauruses

In common day to day use a thesaurus' purpose is to find synonyms of words either to gain further understanding of the original word or to find an alternative word for use in a piece of written work. One of the crucial components of a NL querying system architecture, as cited in (Newman 06), is a tool that can standardise the terminology used in an NL query, so that it is possible to generate a machine-readable query. At first these two tasks may appear quite different to each other but a similar data structure could be used to solve them both.

A thesaurus that may be used when writing a piece work will contain many hundreds if not thousands of entries. An entry may be set out as follows:

> **Ball** (Noun. A round object): sphere, globe, orb, spheroid

I.e. The word itself, followed by a definition to put the word in context and then the words that are synonymous. If these entries were stored within a database, they would each have unique identifiers. There would also be less entries in this database, as where two entries are the same, except for the word itself being transposed with one of its synonyms, they could be stored as one entry. The unique identifier for the entry instead of the word itself could be used as the index and each entry would no longer need to have a word and its synonyms, rather a group of synonymous words. Assuming words can be searched for efficiently, this electronic thesaurus provides the same if not better functionality than a hard copy of a thesaurus. It also allows the thesaurus to translate from a natural language to a machine-readable representation.

As already discussed in section 2.1.2.6, Janus converts the parse tree it gets from its parser into EFL. This process is required to provide standard terminology so that the EFL-to-WML translator can use the domain model to produce an unambiguous representation. A thesaurus like the one described previously could be used to perform the same task as the parsetree-to-EFL translator.

By giving an NL query to an electronic thesaurus, each individual word could be looked up and replaced with the unique identifier of the entry that contains that word. This is a extremely simplistic view, as many of the words in the NL query are likely to be in more than one thesaurus entry. A number of these entries could be discounted before the thesaurus performs the search, as the NLP should have identified which part of speech each word is. However it is unlikely that this would always eliminate all but one entry.

Take the example given previously, ball, this is indeed a round object but it is also a description of a formal party. Both these meanings refer to ball as a noun, so which one is correct? This is not an easy question to resolve.

2.1.3.1 Wordnet: A Lexical Database

$Wordnet^{TM}$[10], is an online lexical database (Fellbaum 98). One of the functions of a lexical database is to work like a thesaurus. If you query it with one word, it will give you one or more sets of words that it believes to be synonymous. These sets are derived from the lexical concepts that contain the word queried. A lexical concept is basically an idea, which can be expressed using one or more different words, very much like the thesaurus entry described in section 2.1.3. In Wordnet, a lexical concept is the hub of the lexical database. Each lexical concept can have one of five types: noun, verb, adjective, adjective satellite and adverb. This lexical concept can then have one or more words and commonly one or more definitions.

One of the major features of Wordnet is the linking of lexical concepts together. This is achieved using a number of different relationships, including antonym, hyponym, hypernym, meronym and holonym to mention just a few. Being able to link lexical concepts together like this is particular useful, as it dramatically increases the amount of information that can be stored with minimal extra effort. Hyponyms and hypernyms can generate hierarchical trees and so can meronyms and holonyms. These trees have the potential to be used to help determine how synonymous two words are, based on how far the tree needs to be traversed to get from one word to the other, although this may not be too reliable.

2.1.3.2 OpenThesaurus

OpenThesaurus (Naber 04) is similar to Wordnet except that is designed for German and not English. It also only concerns itself with synonymy and not antonymy, meronymy, etc. However, one of the most significant features that makes it different from Wordnet is that users can contribute. Users can create, discover and edit synonym sets. The history of the synonym sets is recorded so that administrators of the system can rollback any modifications that a user makes that are incorrect. The number of errors made by the users was minimal and the paper found that the general accuracy of the database of synsets imported was also very high, although a few error-correcting sweeps were needed. This suggests that it is possible, at least in part to build a thesaurus using user contributions. The report did however find that although there were over 400 users subscribed, very few of them actually contributed. This is even more significant as the only reason to subscribe is to contribute, as lookups can be done without subscription.

[10]http://wordnet.princeton.edu/

2.1.4 Entity Recognition

Just mapping words to machine-readable representations, is not sufficient to be able to generate machine-readable queries. In many NL queries there are likely to be references to entities. Entities are basically proper nouns but they can often be quite complex, such as the name of a musical album or a chemical compound. Separating entities from the rest of the query is very important, if all or part of the entity is left in the query it may well effect what the query is asking. For example, take the musical album "Standing on the shoulder of giants" by Oasis. This could lead to the query, "Who recorded standing on the shoulder of giants?" This query could be asking who recorded their album whilst standing on the shoulder of giants, which is probably not what the user intended.

Creating a machine-readable representation for every entity may take an inconceivably long time. E.g. defining all possible chemical compounds. Some entities can often be quite long, particularly in the case of musical albums and there is the potential for them to be ill-formed. Take the Oasis album again, it could easily be ill-formed as "Standing on the shoulder**s** of giants." This is an error but it is very minor and it would therefore be preferable for the system to tolerate it rather than return no result.

2.1.4.1 ANNIE

ANNIE is a tool developed by the University of Shefield NLP group[11] as part of the General Architecture for Text Engineering (GATE) project[12] (Cunningham 06). It has three main stages. The first stage tokenizes the phrase. There are several different types of token:

- word
- number
- symbol
- punctuation
- space

A word token has one of the four orthographies: all upper case, all lower case, initial uppercase and mixed, this information is stored as an additional field within the token. A symbol token can be one of two types, currency or other. Punctuation tokens can be one of three types, start, end or other. A space token is a contiguous set of either space or control characters. Like word tokens, the type of symbol, punctuation and space tokens are stored in an additional field.

[11]http://www.nlp.shef.ac.uk
[12]http://www.gate.ac.uk

Once everything has been tokenized a search is done over the gazetteer[13], this will hopefully find matches for words or sequences of words. The manual for ANNIE (Cunningham 06) gives the example of the phrase 800,000 US Dollars. The lookup finds the sequence US Dollars in the list of currency units.

The final stage is to invoke the appropriate grammatical rule. For the US Dollars example the money rule is invoked. This first uses a macro to detect whether US Dollars is proceeded by a word token that represents an amount e.g. million or the letter m representing a million. If not it drops out of that macro and uses a macro that looks for any sequence of number tokens separated by either commas or dots, this extends the entity to include 800,000 and completes the entity recognition. The entity is also annotated with the type of entity it has been determined as being and what rule was used to determine that.

There are many domains where entity recognition is essential. The grammatical rules that ANNIE uses would make it possible to solve the problem set in section 2.1.4 about how to recognise an almost endless list of chemical compounds. It might even be possible to adapt the grammatical rules to allow tolerance of slight illformedness in long and complex entities.

ANNIE also has a resolution algorithm to handle anaphora. Every pronoun, both personal and impersonal will have been tokenized as a pronoun. The algorithm first dismisses all pronouns that it finds to be superfluous. The context is then determined. The context indicates the number of sentences before the current sentence should be search for candidates for the pronoun resolution. From this list of candidates those that are incompatible are removed, e.g. if the pronoun is *she* candidates with gender specified as male are eliminated. From the remaining candidates one is then selected using the criteria specific to the pronoun, which varies depending on whether it is personal or impersonal.

2.1.5 Command Recognition

Using Janus as an example again, there is an assumption that the WFL representation can be mapped to a database language query. Even if the representation is without ambiguity and it is clear, in a machine-readable form, exactly what information is required from the database this mapping is not necessary possible. The reason for this is that the database query language may not be able to express what the WFL representation wants. One of the ways it may not be expressive enough is because database stores specific information but the query requires some processing on this data to get the answer it wants. E.g. In a musical domain the two following question may be asked:

[13]A gazetteer is simply a several lists of terms that are common to a particular domain, e.g. a list of company names, species of birds, make/models of cars, etc.

1. “How many tracks are there on the album Standing on the Shoulder of Giants?”
2. “Is Thriller by Michael Jackson longer than Purple Rain by Prince?”

SQL has many data processing functions such as count, sum, max, etc. (MyS 07) and would be able to answer the first of these questions using its count function. SPARQL Protocol and RDF Query Language (SPARQL, see section 2.2.4.1) does not have a count function (Prud’hommeaux 06) so would not be able to answer this question without some post-processing. The second question would be more difficult for even SQL to give an answer that does not require post-processing. The only way to achieve this would be through a nested query.[14] The only straightforward query that could be used is something like:

```
SELECT album.name sum( track.length ) as albumlength
  FROM album INNER JOIN track ON album.id = track.id
  WHERE ( album.name = 'Thriller' AND album.artist = 'Michael Jackson' )
    OR ( album.name = 'Purple Rain' AND album.artist = 'Prince' )
  GROUP BY album.id
  ORDER BY albumlength DESC
  LIMIT 1
```

This would still require the system to analyse album.name and output yes if is Thriller or no if it is Purple Rain. The structure of this SQL query is quite complex and building it up from the NL query given would be extremely difficult, unless there was a direct mapping from that format of question (i.e. is Album A longer than Album B?) to this SQL query. SPARQL could solve this question more easily using its ask command, assuming both albums has their lengths expressed explicitly. Using just a couple of simple examples it is clear that mapping to a database query language is at best difficult and at worst impossible. Therefore, at least a degree of command recognition and extraction is required to be able to generate the expected result for a user.

2.2 Semantic Web Technologies

One of the first attempts to capture machine-usable descriptions on the web was by using Meta Content Framework (MCF) (Guha 97). MCF used Directed Labelled Graphs (DLGs) comprising of sets of labels, nodes and arcs, where an arc is a triple that connects up two nodes using a label. These graphs can then be represented using eXtensible

[14]A nested query could take something like the SQL statement shown but without the order or limit clauses. By appending to the output a yes on the Thriller record and no on the Purple Rain record. This table could then be queried on this field reusing the order by album.length and limit to one clauses.

Markup Language[15] (XML) (Bray 04). In combination with Minsky (Minksy 74) and others frame-based representation systems, MCF inspired the development of Resource Description Framework[16] (RDF) (Manola 04) (Lassila 98).

The new generation of the web aims increase the amount of machine-readable data it retains. This is unlikely to be achieved without a standard approach for representing knowledge, RDF, many believe could be this standard. The World Wide Web Consortium (W3C) made RDF a recommendation in 1999, (Lassila 99), this was subsequently revised in 2004.

2.2.1 RDF and RDF Schema

Like MCF, RDF uses triples to capture its data and the relations between its data. It can also use XML to represent its data in text format[17]. According to Berners-Lee, RDF is the third layer of his semantic web "layer cake" (Berners-Lee 02). There are many envisioned levels above this to produce a web that stores machine-readable data that can be trusted and can be used to produce more information than the sum of its part, i.e. through inference and other logic-based techniques. The first couple of these layers deal with the structure of the RDF data. The first of these is RDF Schema.

RDF Schema[18] (RDFS)(Brickley 04) allows RDF data to be grouped into a class/property structure, in some ways like an object-oriented programming language, such as Java. Structuring RDF data into classes with properties and instances of those classes, provides the ability to generate additional data above that specified in XML representation of the RDF. The additional data that RDFS can provide is limited. RDFS only provides the properties subClassOf and subPropertyOf to allow hierarchical relationships to be defined. RDFS cannot define functional, inverse or any set theory (e.g. intersections, unions, etc.) relationships. There have been a couple projects to develop a language that could represent these relationships, the DARPA Agent Markup Language[19] (DAML) in the US and Ontology Interchange Language[20] (OIL) in Europe, these two projects eventually merge to form the Web Ontology Language (OWL) (Horrocks 02).

2.2.2 OWL

The complexity of relationships that need to be represented in a domain's ontology varies depending on the domain. It is important that even the most complex logical relationships can be represented but it is as important that if only simple logical relationships

[15]http://www.w3.org/TR/REC-xml/
[16]http://www.w3.org/TR/rdf-primer/
[17]RDF can also be represented in N3, NTriple and TURTLE
[18]http://www.w3.org/TR/rdf-schema/
[19]http://www.daml.org/
[20]http://www.ontoknowledge.org/oil/

are required, only these are used. For this reason there are three main species of OWL[21], that can represent increasingly more complex logical relationships. They are OWL Lite, OWL DL and OWL Full.

OWL Lite is equivalent to the SHIF(D) description logic. OWL DL is equivalent to the SHOIN(D) description logic. (Horrocks 03). OWL Full provides more flexibility than OWL DL but it is neither sound nor complete. There is no reasoning algorithm that is guaranteed to be decidable across all the data that can be expressed in OWL Full. OWL Full should only be used if it is impossible to represent a domain with OWL DL. Surveys have been carried out that have found many ontologies that are defined as OWL Full should really be OWL DL or even OWL Lite (Bechhofer 04).

As OWL stores logical assertions it is possible to reason over these to provide additional information through inference. In general this can be achieved in one of two ways, at the time triples are imported or at the time of querying the triplestore. Both techniques have their advantages, however, more often than not, greater processor time to perform inference at the point of query is at a higher premium than extra storage space.

2.2.3 Data, Information and Knowledge

As already stated one of the main purposes of knowledge technologies is to take human concepts and convert them into machine-readable knowledge. By doing this the machine does not just store data that a human has to interpret to infer knowledge but through using inference and other logical processes can produce knowledge itself. This is far from trivial, especially with a large complex database. One of the greatest problems is that there are several stages that data must go through to become knowledge. Figure 2.3

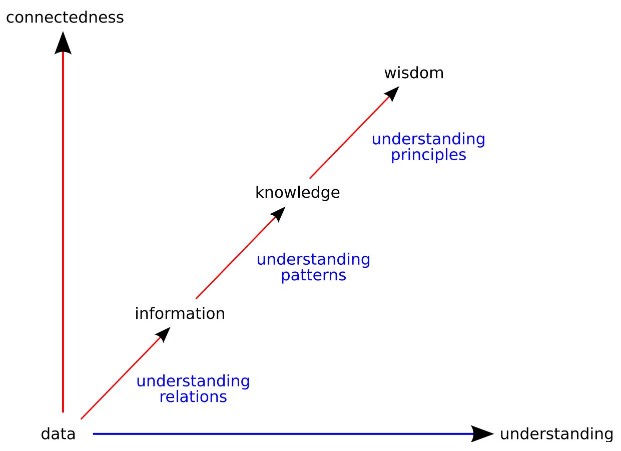


FIGURE 2.3: Data, Information, Knowledge and Wisdom

by (Bellinger 04) shows how both relations and patterns must be understood before

[21]Since the original specification of these three species a subset of OWL Lite, called OWL Tiny has also been defined by European SW Advanced Development (SWAD-Europe) group (see http://www.w3.org/2001/sw/Europe/reports/dev_workshop_report_4/)

data can be transformed into knowledge. What is meant when someone says, "The computer understands..." is subjective. In this report any reference to a computer's understanding is considered to mean that the data has been sufficiently structured to always or almost always return the correct result.

2.2.3.1 How RDF turns Data into Information

RDF uses the principle of a triple which is analogous to the construction of a sentence, each triple must have a subject, an object and a predicate linking the first with the second (Hughes 04). By using such a structure, data is converted to information because the machine understands the relationships as they are explicitly defined.

Humans do not necessarily need relationships to be explicitly defined, take the digital lab book replacement from the Comb*e*Chem project for example, (see Chapter 2.3. (schraefel 04b)). A chemist using their paper-based lab book may record two related results next to each other, it may be clear to them and their colleagues that by their proximity, these results are related but a machine would not be able to make such an inference. It is conceivable that a system could be designed that reads in lab book pages and can determine that results are related. There would however be serious questions on how reliable such a system could be, as only small errors could render a whole database useless.

2.2.3.2 How RDF turns Information into Knowledge

Getting machines to understand relationships is only the first step towards producing knowledge and not just data, the next step is unfortunately much more difficult. As shown by Figure 2.3, it is necessary to understand patterns to convert information into knowledge. Patterns are more difficult to explicitly define than relationships. Triples do provide some help in determining these patterns, as long as the correct design ethos is used. Triple predicates such as RDFS's subClassOf enforce a hierarchy on a triplestore (Manola 04). It is almost inevitable that an ontology designer will enforce some sort of hierarchy, as without this structure the data within the triplestore would become almost impossible to manage.

A RDF triplestore on its own cannot express patterns explicitly like it does relations. It stores relations (i.e triples as physical records within a database table or similar data structure but there is no physical storage of patterns. In general, a pattern can be considered as a set of relations that can be grouped together using one or more logical expressions. Using logical expressions allows reasoning to be performed, which generates inferenced relations. Inferenced relations are a crucial part of RDF as they give extra meaning to data without making consistency preservation more complex.

RDF requires OWL[22] to be able to define logical expressions to generate groups that share implicit relations. OWL provides properties such as unionOf, complementOf and inverseFunctionalPropertyOf (see section 2.2.2). By using RDF and the properties provided by the OWL ontology it is possible to start tackling the problem of converting information into knowledge by designing ontologies that define patterns for a particular domain.

2.2.4 RDF Triplestores

A relational database has its data and its structure, i.e. the tables and the relationships between them, separate. A database that stores RDF, i.e. an RDF triplestore, is considered to store its data and structure together, as the structure itself is stored as data. However, care must be taken with how structural and instantiation data intermingle. Often instantiation data is more contentious, e.g. there may be disagreement with exactly what toppings make up a four seasons pizza. Including too much instantiation data within and ontology can often make it less flexible, it is also more unlikely that people will use your ontology if they disagree with you instantiations. A potential solution to this is to provide an ontology for defining a domain's structure and then an ontology extension to define useful instantiation data.

RDF triplestores provide a more flexible way to manage data than relational databases because they make storing relationships as generic as it is possible to achieve, i.e. a flat list of one-to-one relationships. This can be highly beneficial as whenever some new triples are generated they can just be added directly to this list. This is not always the case in relational databases as the data may not be compatible with the tables' enforced structure. It can be said that in itself RDF captures information not just data (Hughes 04). It is important to understand how an RDF triplestore works to know why the previous statement can be asserted.

2.2.4.1 Querying a Triplestore

The whole purpose of having a triplestore is to be able to query it. SPARQL (Prud'hommeaux 06) is designed specifically for querying RDF Triplestores. SPARQL uses a syntax similar to SQL, which is intuitive considering a number of RDF triplestores use an almost flat-file SQL database to store their triples. SPARQL syntax also resembles a TURTLE/N3 syntax. In its simplest form a SPARQL query will allow the user to search for certain patterns of RDF triples, where one or more elements of the triples are unspecified. The following example is a query to find a group of human siblings:

[22]This could also be achieved with DAML+OIL but OWL is the most current W3C recognised standard.

```
PREFIX rdf: <http://www.w3.org/2000/02/22-rdf-syntax-ns#>
PREFIX exp: <http://www.example.com/ontology#>

SELECT ?sibling WHERE {
  ?sibling exp:hasParent <http://www.example.com/people/Bob_Smith> .
  ?sibling exp:hasParent <http://www.example.com/people/Jane_Smith>
}
```

SPARQL is not the only language that has been designed to query RDF triplestores, Resource Description Query Language (RDQL) (Seaborne 04) and Resource Query Language (RQL) (Karvounarakis 02) are also commonly used query languages.

The best way of understanding how a triplestore can be used is to look at an example system. The Comb*e*Chem project's digital lab book is backed by a triplestore (schraefel 04b) and this is a good example of how a triplestore can be used and why it can be better than using a relational database system, (see section 2.3.1.3).

2.2.5 Wordnet on the Semantic Web

2.2.5.1 The KnOWLer Project

Wordnet was originally created as a lexical database stored in a relational database model, i.e. an SQL database, (see section 2.1.3.1). There have been several attempts to implement it using SW technologies, i.e. RDF, OWL and triplestores. The University of Neuchtel's KnOWLer project[23] has defined an ontology for Wordnet 1.7.1. Figure A.2 is a diagram representing this ontology. A key for this diagram can be found in Appendix A.2.

The ontology produced by the KnOWLer project does provide most of the functionality defined for Wordnet 1.7.1, as discussed in section 2.1.3.1 but it does not give a solution to the problem posed in section 2.1.3. The KnOWLer project was one of the earlier projects to try to produce a Wordnet ontology, since then quite a lot of work has been done in this area. In fact, in July 2006, the W3C produced a working draft for a Wordnet ontology(van Assem 06).

2.2.5.2 W3C Wordnet Working Draft

There are a number of significant differences between the W3C working draft of a RDF/OWL representation of Wordnet and the KnOWLer project's Wordnet ontology. One of the most significant is the use of synonym sets (synsets). KnOWLer's ontology does

[23]http://www2.unine.ch/imi/page11291_en.html

make reference to them, as a lexical concept is a Wordnet synset but they are not defined explicitly. This difference is not just semantic, the way that synsets can be used in the working draft is slightly more sophisticated than in KnOWLer. There is not just synsets but also word senses. These reify the relationship between the synsets and the word objects. A synset may contain many word senses but a word sense must only be part of one synset, also many word senses may refer to the same word object. The main purpose of this is to keep the sense of words separate from their forms. I.e. words may have the same spelling but different meanings and vice-versa using word senses normalises this many-to-many relationship.

Reification using word senses makes it possible to markup a sentence much more richly. E.g. The sentence "He went to the ball", could mean one of two things. Usually a human would be able understand what was meant by the sentence by listening to sentences that preceded and followed it. A computer may initially do this to determine the word sense but as this can be complex, it would be preferable to only have to do this once. Using word senses, this identical sentence could be stored using to slightly different lists of word senses. If word senses did not exist, to markup the sentence with the same richness, each word would have to represented by a word object and synset pair. Therefore word senses are useful if the sentence meaning is being marked up to be preserved but less so if it being processed and then disposed of, which would be the case when generating machine-readable queries in a NL querying system.

A second difference is the use of three classifiedBy properties (classifiedByTopic, classifiedByUsage and classifiedByRegion). Although these only subtly change the schema, they have the potential to make a lot of difference to how the ontology can be used. In the KnOWLer ontology the only way to determine whether two words are from the same topic is to traverse the hyponym or meronym trees. As this is not their intended purpose, there is a good chance the results will be inaccurate and incomplete. One of the main benefits classification properties could give in a NL querying system is improved entity recognition, (see section 2.1.4). E.g. If it known that filter is classified as a process in chemistry, determining any entities within this query can be quicker and more accurate because the rules used to find entities can be targeted towards finding chemical compounds rather than the names of pop bands.

A problem with using classification properties is that the relationships are often subjective. ClassifyByRegion may be objective but classifyByTopic and classifyByUsage are potentially more controversial. This problem could be significant because of the number of relationships between synsets. One erroneous classification could lead to a lot of inaccurate inferences. Keeping poor classifications to a minimum is essential but eliminating them completely is a very difficult task. A further complication is that there are not just poor classifications, there are also improbable classifications. Therefore an NL querying system would not benefit from making hard inferences rather making probabilistic ones, as sometimes the less likely meaning of a query is the correct one.

Another difference is the use of tag counts, these record the number of times a particular word sense is found in the corpus. This property has the potential to be quite useful assuming the corpus used is sufficiently large and is not an unusual selection. If the NL querying system is of a generic design but is then customisable for a particular domain, the tag count values could be set by using a corpus skewed towards type of text used in that domain. This may well improve the probability of determining the right word sense first time, however producing a suitable corpus that has all its words marked up with the correct word senses may be very time consuming. The last significant difference from the KnOWLer ontology is the use of the frame property for verb word senses. This property is designed to show what the makeup of the sentence must be for a particular verb word sense to be used. E.g. teach has three frames:

1. Somebody —-s
2. Somebody —-s somebody something
3. Somebody —-s something to somebody

The —-s is replaced with the verb in question, i.e. teach(es) and the somebodies and somethings can be replaced with the appropriate nouns. The frame property has the range rdfs:literal, this is not ideal because some additional knowledge is required to know how to decode the text string stored as an rdfs:literal into something useful. Unless this is done, the property is rather useless for anything other than display purposes, as it cannot check that the sentence uses the verb in a valid way.

2.3 Implementations Using SW Technologies

There is wide array of projects that have produced implementations using SW technologies. There are several that are interesting and worth focusing on. The Comb*e*Chem project is interesting because it focuses on one domain, chemistry experimentation, that could benefit from having a NL querying system. Comb*e*Chem has had a number of spin-off projects, SmartTea and myTea, that have focused more on the knowledge capture through analogy. The use of analogy is also important for NL querying systems, as it is unlikely that an effective system could be developed for a complex domain, such as chemistry experimentation, without a more straightforward analogous domain being used to gain a greater understanding of how the system is working. Another project of interest is GINSENG because it uses SW technologies to create a quasi-NL interface.

2.3.1 The Comb*e*Chem Project

The Comb*e*Chem project has been a partnership between the School of Chemistry and the School of Electronics and Computer Science at the University of Southampton, as part of the eScience research stream. It has involved people with a wide range of expertise. In particular the Comb*e*Chem project has concentrated on producing a digital replacement for the chemistry lab book (schraefel 04b). Paper-based lab books have several inherent flaws to them, which have become more apparent and significant in the 21st Century.

- Difficult to share information, particularly in a widely distributed community.
- Difficult to back up information recorded.
- Limited guarantee of being able to prove Intellectual Property (IP) rights.
- Limited structure of data, leading to the potentially crucial information not being recorded.

All of the above issues have the potential to be resolved by the implementation of a digital lab book replacement backed by a database capable of storing all the data captured by the device.

- A database gives immediate access to the data for any permitted chemist.
- Data is stored on a server and therefore can easily be backed up regularly.
- When data is recorded it can be timestamped giving increased strength to intellectual property rights. Other chemists being able to view the data immediately through querying the datastore adds further strength IP rights.
- Structured forms are provided for experiment planning and execution that will remind the user to submit data they may have previously forgotten[24].

A paper-based lab book has its drawbacks as well as its benefits (schraefel 04b), such as:

- Ease of access, i.e. flipping back and forth through pages.
- Simplicity of data entry, i.e. no complex computer applications to learn.
- Ability to draw freeform sketches.
- Portability.
- Resilience to damage, e.g. chemical spills, dropping on the floor, etc.
- Security of storage.

[24]These structured forms are sufficiently flexible to not frustrate the user with rigidity of the data input.

- Minimal effort required to capture data.

For the digital lab book replacement to be a success it needed to maintain as many of these benefits as possible. Simplicity of data entry and minimising the effort required to capture data are two of the most important features of the paper-based lab book and much effort was given to maintaining these features.

2.3.1.1 The SmartTea Project

One of the biggest problem with designing a database, whether it be a relational database or an RDF triplestore is understanding the specific domain that it is to be used for. This is why the SmartTea project was developed.

SmartTea[25] uses a technique developed by the designers of the digital lab book replacement to try and elicit information about how a chemist plans and carries out an experiment (schraefel 04a). The main problem with creating a digital lab book replacement for a chemist is actually understanding what they put in it. This is particularly difficult if the chemist is carrying out a complex chemistry experiment that the designer has no understanding of.

SmartTea uses the analogy that making a cup of tea is like conducting a chemistry experiment, this benefits the designer because he/she understands the planning, process and expected outcome and then can observe how and what the chemist records to their lab book through the course of the experiment. The SmartTea analogy was inspired by (Dix 03), this advocates deconstructing a process to a point where its exact purposes can be understood, (i.e. from a complex chemistry experiment down to making a cup of tea), before reconstructing the process in a different medium, i.e. from paper-based to digital lab book.

Even by using this deconstructing technique it is still unlikely that all the information about the domain will have been elicited. Therefore if a triplestore with an ontology is used, there is greater flexibility to alter the domain's structural representation later, as it would be much less likely, than with a relational database, that data already stored would be affected.

2.3.1.2 The myTea Project

The myTea project[26], is similar to SmartTea to the extent that it uses an analogy to a common simple activity to represent the more complex task a researcher is trying to

[25]http://www.smarttea.org
[26]http://www.mytea.org.uk

tackle. myTea was designed to help bioinformaticians better integrate their work electronically, to save time and make it easier to share their results with others (Gibson 05). The analogy of assembling a jigsaw puzzle is used to represent the work of bioinformaticians.

The work that bioinformaticians do is quite different to that done by chemists in the SmartTea project. All the work is done in silico, so they do not need an electronic lab book for the same reasons as the chemists do. At the moment there are a lot of applications that the bioinformaticians use, the problem is they are badly integrated, this causes several problems. Firstly, it slows the progress that the bioinformaticians can make because they need to copy and paste, download and upload information between applications. Secondly because of this piecemeal transfer of data it is very difficult for the user to keep track of their data, e.g. whether they a have changed a copy of the file they downloaded and therefore need to download it again to get a clean copy. The reason that this problem is caused is because despite there being plenty of web applications available, these are not integrated within the desktop where the bioinformaticians do their work. myTea attempts to try to solve these problems.

The equivalent to the electronic lab book in the SmartTea project, is a virtual lab book that allows the user to interact with web-based applications seamlessly. It also allows the analysis that the user performs on this virtual lab book to be stored and processed within Web-accessible/sharable SW technologies such as triplestores and ontologies (Gibson 05). The system is also designed to generate reports from this analysis which can be annotated by the owner and then shared with other users of the owner's choice. Annotation is an important component of the project to allow the user to specify how reliable the data is, terms like *finished*, *unsatisfactory*, *useful* and *needs attention* help keep track of the work. As well as the user annotations, provenance data is automatically captured. By building this framework it should help tackle the problems cited in the previous paragraph. The framework should also be extensible to the world of cheminformatics.

For this report, the two most relevant features of this project is its use of SW technologies and like SmartTea, the use of an analogy so that the systems designers have some common ground with the domain experts. Implementing a NL querying system with such a complex domain, such as the Comb*e*Chem domain, would be extremely difficult. Therefore, the use of an analogy where both side share cpmmon ground should make an initial implementation easier.

2.3.1.3 The Comb*e*Chem Triplestore

The Comb*e*Chem triplestore is the database described in section 2.3.1. The Comb*e*Chem triplestore uses JenaRDF[27] implementation to store its data. The Comb*e*Chem triplestore currently uses an specifically designed RDFS schema but there are plans to use an OWL ontology in the future (Hughes 04).

The current schema defines processes as well as substances, using RDFS's subClassOf property to define the hierarchy of each. e.g. FiltrationWithBuchnerFunnel is a subClassOf Filtration (Hughes 04). By using the triplestore and Comb*e*Chem schema it is possible to define an experiment plan, which can then be instantiated when the experiment is carried out to make sure the plan is followed correctly. When an experiment plan is instantiated it also provides the ability to annotate and make observations that are then directly associated to that experiment instantiation. It is possible to have more than one instantiation of the same experiment plan that allows either the same chemist or a different chemist to repeat the experiment to check the original results.

The design of an experiment plan uses a process-product pairing. This scheme requires each experiment plan to start with a process, where one or more of the experiment ingredients go through this process, to produce a product. This product is then used in the next process, possibly with one or more other experiment ingredients. This spine is continued until the final experiment product is produced (Hughes 04). At the moment this is an over simplified process with just one central spine. It is likely that one process may produce more than one product that needs continued processing or that multiple initial experiment ingredients will need to be processed before they can be combined. This would generate a graph with more than one spine. This was not designed for in the original schema, as the experiments that the system would be used for were very unlikely to be of this nature. Learning lessons from OWL-S (Martin 04) and the Business Process Execution Language, WS-BPEL) (Alves 06) may make it possible to design an ontology that is capable of generating graphs like previously described (Hughes 04).

The Comb*e*Chem triplestore has a ModelServer with it own API (Hughes 04). This provides two useful bits of functionality:

1. Querying, in this case using RDQL.
2. Changing the experiment model.

These two pieces of functionality allow software to be designed for the digital lab book to allow a chemist to design and modify an experiment plan, instantiate an experiment and make observations, annotations and recordings on this experiment instantiation.

[27]http://jena.sourceforge.net/

Through the use of an ontology to structure the triplestore data, it is possible to make inferences about the data recorded. As only an RDFS schema was designed this inference is a somewhat limited. Inference mainly comes from the subClassOf property but this still gives a lot of free data. A typical experiment's number of triples increases ten-fold when Jena's RDFS inferencing model is used (Hughes 04). If an OWL ontology had been used instead, the increase in the number of triples would have been even more significant. This is because the complex logical relationships that OWL can represent, make it possible to use a considerably more sophisticated inferencing model on the RDF data to produce a much greater number of inferenced triples.

2.3.2 GINSENG: Guided Input Natural language Search ENGine

GINSENG (Bernstein 06) is one of the first systems that attempts to provide a NL/quasi-NL search engine that uses SW technologies. It guides the user when they enter words for their query to try and ensure it gets an NL query that uses known words and structure. This helps overcome one of NL querying's greatest problems, i.e. domain opacity, (see section section 2.1.1).

By having a guided input means the NLP system and techniques required to produce a machine-readable query can be much simpler. GINSENG just has three main parts to its architecture, a multi-level grammar, which contains about 120 static domain-independent rules and a dynamic grammar rule for every entity contained within the loaded ontologies, that is basically the popup options provided in the guided input interface. The second part is the incremental parser, as well as specifying all the parsable sentences it also provides the information required to generate the machine-readable (SPARQL) queries. The third part is Jena's SPARQL interpreter that executes the query and returns the result.

Although GINSENG performed well in preliminary user testing there were a few limitations/issues with the system. First, any queries that could not be transcribed into SPARQL could not be handled, such as "How many cities named Austin are there in the USA?" Second, the grammar of GINSENG is limited, meaning that perfectly acceptable NL query may not be understood, although trial users did not find this too big a problem. Because the grammatical rules are static if any common structures are missing they have to be added manually. A further problem is that a user must use certain terms, if they try and use a synonym the query will most likely fail. The paper states that there are plans to use Wordnet to facilitate using synonyms. Last, The user may find popup guidance annoying, as can often be the case with popups. It may also be frustrating if they keep typing in a query and the popup keeps telling them there is no way of completing the query within a valid grammatical structure. It may also increase ambiguity, as the user strives to construct their query so it is valid, they have try to express it slightly more simply than they would if they had more freedom. This may take away some of the context of the query making it less likely to return the correct result.

2.4 Summary

There have been many attempts to build NL querying systems. Some have been more successful than others. In general, the more successful have been those that have focused on only being able to query one domain at a time, e.g. Lunar (Woods 72), Planes (Waltz 78) and Ladder (Hendrix 78). Focusing on more than one domain seems to cause major problems with ambiguity. To avoid ambiguity, a NL querying system would need to be designed generically, so that it can be used for more than one domain but not at the same time.

Even when an NL querying system focuses on only one domain there are still many issues with natural language that must be handled to ensure the query is interpreted correctly, the correct database queries are made and the appropriate information is returned to the user. Various implementations have developed tools and techniques to overcome these issues. Wordnet can be used to determine synonyms and has the potential to be adapted to perform lexical to domain mappings. ANNIE can be used to recognise entities and can provide help against anaphora, through its pronoun resolution. Ellipsis, used in Lifer(Hendrix 78) as well as other implementations can help when sentences are ambiguous because of a lack of information being provided. Tools for command recognition and other areas where transcription from natural language to machine-readable queries is difficult need to be considered when building an effective NL querying system. In particular the problem of domain opacity, that was a strong focus in GINSENG (Bernstein 06), needs to be tackled to ensure that the system is user-friendly.

Wordnet has been implemented so that it can be used on the semantic web, (see section 2.2.5). In some ways this is an intuitive step for two reasons. First, Wordnet is intended to be a resource that is available globally, that is why there is a web interface[28] and can be downloaded to be used locally. Making it available as an ontology and RDF data file is just the next logical step. The Wordnet ontology design from KnOWLer, (see section 2.2.5.1), uses RDFS's subClassOf property along with OWL's inverse, transitive, symmetric properties. These are much more succinct and intuitive ways of representing certain relationships, for example setting hyponymOf as the inverse of hypernymOf, as well as making both of these properties transitive.

GINSENG (see section 2.3.2), has also demonstrated how SW technologies can fit nicely into a NL querying system. In particular how OWL ontologies can be uploaded into the Jena ontology model that can be then used to generate the dynamic grammar rules.

[28]http://wordnet.princeton.edu/perl/webwn

The fact that a whole domain can be wrapped up into a single file that can be globally accessible and easily loaded into a system to assist that system in solving a problem.

The Comb*e*Chem provide a useful example of a domain that has been represented using SW technologies. It provides insight into a system that would benefit from a NL querying system. Comb*e*Chem and its associated projects, SmartTea and myTea, consider the problem of understanding an expert domain, this must also be considered in the design of a NL querying system. Evaluating such a system within a complex domain would be hard, as it would difficult determine whether errors where due to flaws in the system or a lack of understanding of the domain.

As determined in section 2.1.1, there are a number of disadvantages with using a NL querying system over other form-based or GUI systems. This chapter has reviewed a number of different tools and techniques to solve these problems. A number of these are already implemented using SW technologies. Those that do not already use these technologies may benefit from their use to help represent the data that they use, e.g. ANNIE's gazetteers and grammatical rules or the rules for recognising and extracting commands.

Chapter 3

A Conceptual Framework for a Semantic Web-based NL Querying System

The NL querying systems and SW technologies discussed in Chapter 2 have provided some insight, making it now possible to design a conceptual framework for an NL querying system, that takes advantage of SW technologies. This chapter describes how to design such a system. It first considers the domains it may be used for. It then describes the components required and how they should be constructed together, as illustrated in Figure 3.2.

3.1 Domains for the NL Querying System

For widespread adoption of a NL querying system it needs to be generic. To achieve this, all domain dependent knowledge needs be uploadable to the system, which will in turn determine the behaviour of a number of the system components. To best explain how each of these components will work it is necessary to define a couple test domains so that examples can be given in context.

3.1.1 Comb*e*Chem Domain

As previously discussed in section 2.3.1, Comb*e*Chem is a project that was intended to provide users with an electronic lab book backed by a triplestore to store all the information submitted. As this project already has semantic web aspects it is well-suited to be a test domain for the NL querying system. There already exists an RDFS schema that helps structure the data that has been collected from the electronic lab book. Comb*e*Chem would not be the most suitable test domain to use first. It is rather complex and it has a lot of content that is difficult for those that are not domain experts to understand. A more straightforward domain is needed that can be understood by both the NL querying system domain designer and by its users.

3.1.2 MusicBrainz Domain

Section 2.3.1.2 stated how both SmartTea and myTea used analogies to help the system designers and domain experts develop a system that would meet the latters needs. A similar analogy is needed to provide a simpler domain to test the NL querying system.

MusicBrainz[1] is a website designed to allow users to submit information about musical albums. It stored in a large database and can be queried via an online form. The database can be downloaded and imported into a PostgreSQL[2] relational database. MusicBrainz also provides an ontology of sorts for its database structure[3]. This is not particularly well-formed but it does give a good outline of how to define an ontology for such a domain. MusicBrainz provides a good analogy for the Comb*e*Chem domain (see section 3.1.1) as it shares similar features:

1. Both domains have complex entities. Comb*e*Chem has chemical compounds, MusicBrainz has the names of albums, tracks and bands.
2. Both have actors, actions and things that can be acted upon. Comb*e*Chem has the concept of a user defining a chain of process-product pairs (see 2.3.1.3) and MusicBrainz has artists that can perform actions, such as sing, play guitar on tracks and albums.
3. Both have an almost limitless number of actions that can be performed, which are organised in a hierarchical structure. For Comb*e*Chem there is:

   $$filter \rightarrow filterwithfunnel \rightarrow filterwithbuchnerfunnel$$

   for MusicBrainz there is:

   $$playinstrument \rightarrow playguitar \rightarrow playbassguitar$$

4. Comb*e*Chem has the concept of creating a plan and then instantiating the plan each time the experiment is carried out. MusicBrainz has albums, for each album there could be different releases of that album, e.g. US/UK release, special editions, etc. Therefore instantiations of that album need to be made.

[1]http://musicbrainz.org
[2]http://www.postgresql.org/
[3]http://www.ldodds.com/projects/musicbrainz/schema/index.rdf

3.1.2.1 Designing the MusicBrainz Ontology

The MusicBrainz ontology not being particularly well-formed, as shown by parsing it through the University of Manchester's OWL validator[4], where it validates as OWL Full but with many additional messages. A further problem with this ontology is that it does not sufficiently express the domain, apparently an extended ontology does exist but it has not been possible to find it. Therefore it was decided to design a new ontology based on the original MusicBrainz ontology, that would be more expressive but also validate to OWL DL.

MyMusicOntology v1.1 (mmo1.1) [5], is this reworked version of the MusicBrainz ontology, it has increased domain coverage and does validate to OWL DL using the University of Manchester's OWL validator. Figure 3.1 shows the domain coverage of mmo1.1. In designing the mmo1.1 ontology the emphasis has been on ensuring that it only represents the domain's structure and not any instantiation data. This is a requirement of Figure 3.3, that defines how domain data should be defined so that it can be swapped in and out to allow the NL querying system to be used for different domains.

There are three classes Type, Status and MusicalActionType, that may have any number of instantiations. It is unlikely that Type and Status would ever need more than ten instantiations each and in the original MusicBrainz ontology these instantiations were defined within it. However new instantiations may become applicable in the future and the only way to represent these would be by re-defining the ontology each time.

MusicalActionType has an almost infinite number of instantiations for every action it is possible to perform on an album, e.g. playing any instrument, mixing, writing lyrics, composing etc. By ensuring the instantiations of these classes can be defined by the user and are not hard-coded by the designer provides more flexibility within the domain. A disadvantage of this is the reliance on users to define instantiations sensibly and to annotate and label these instantiations appropriately. As data input is not the major concern of the implementation, this problem can be somewhat overlooked for the time being.

3.2 Data Flow for a NL Querying System

There are many steps between the input of a NL query and the output of a human-readable answer. Careful consideration is needed at each step to ensure seamless data flow through the system. Figure 3.2 is partially adapted from the Intermediate Representation Language System (IRLS) proposed in (Androutsopoulos 95). It also observes how Janus generates a machine-readable query, that represents the NL query but is not a

[4]http://phoebus.cs.man.ac.uk:9999/OWL/Validator
[5]http://www.ecs.soton.ac.uk/~drn05r/musicbrainz/mmo1.1

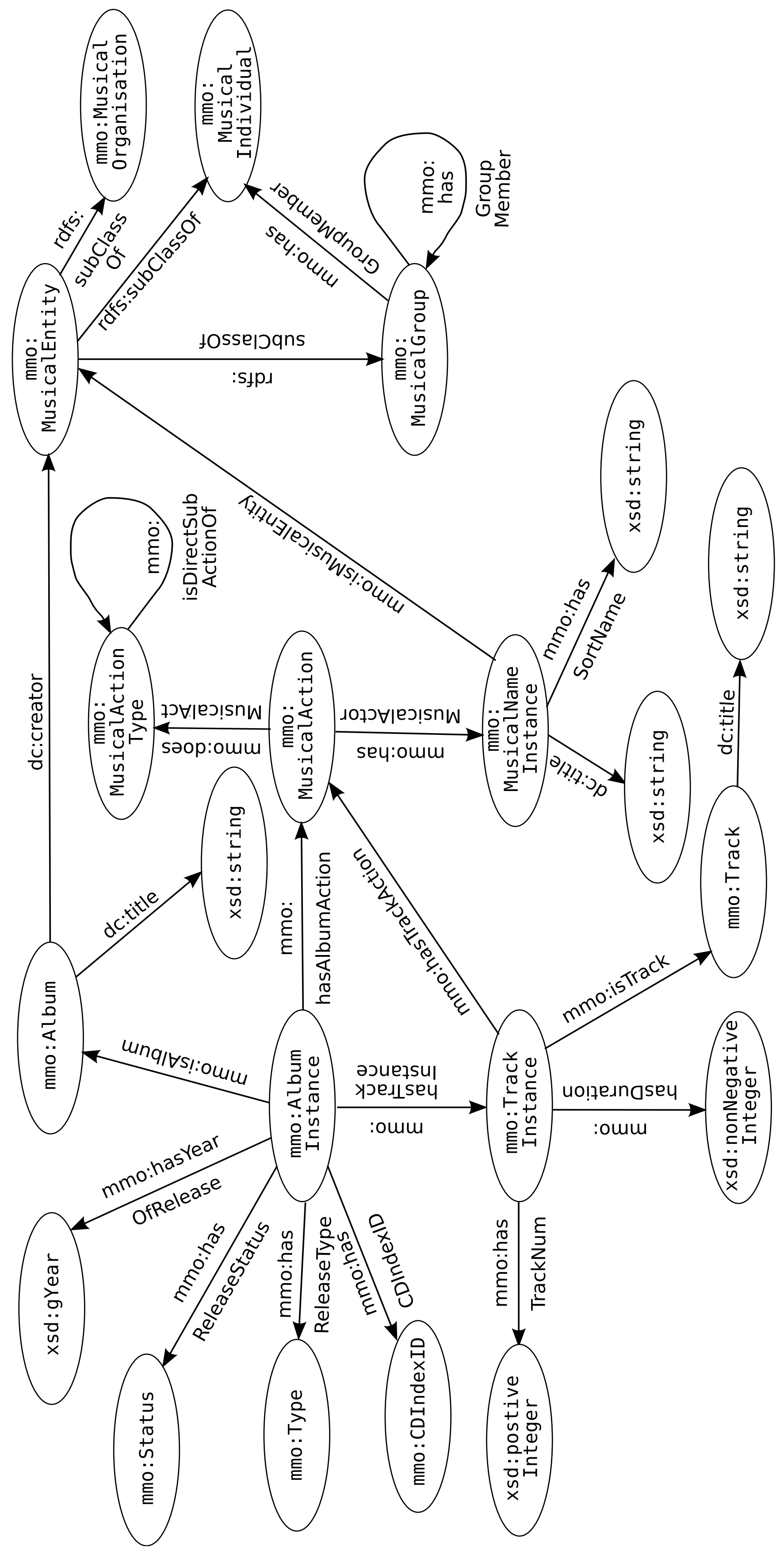


FIGURE 3.1: Domain Map based on mmo1.1 Ontology

database query and how this is useful for the purpose of incorporating a database interface into the NL querying system. The backbone of the NL querying system is made up

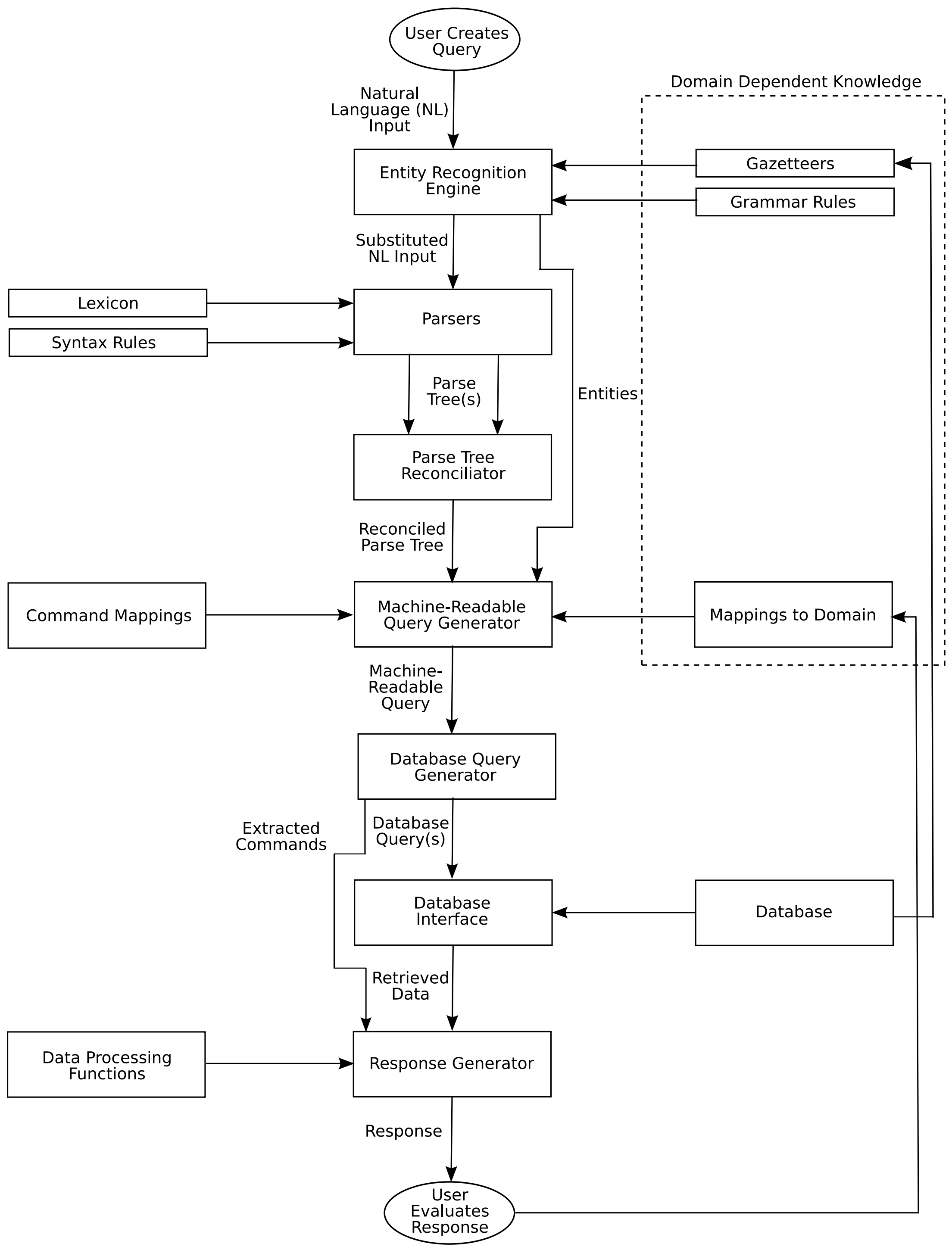


FIGURE 3.2: The Flow from NL Query to Human-Readable Answer

of data stored in a triplestore. The way that this data is organised is important so that the system can be generic and allow domain data to be swapped in an out depending on the domain that it is being used for. Figure 3.3 shows how this data must be arranged to allow data from a new domain to be imported.

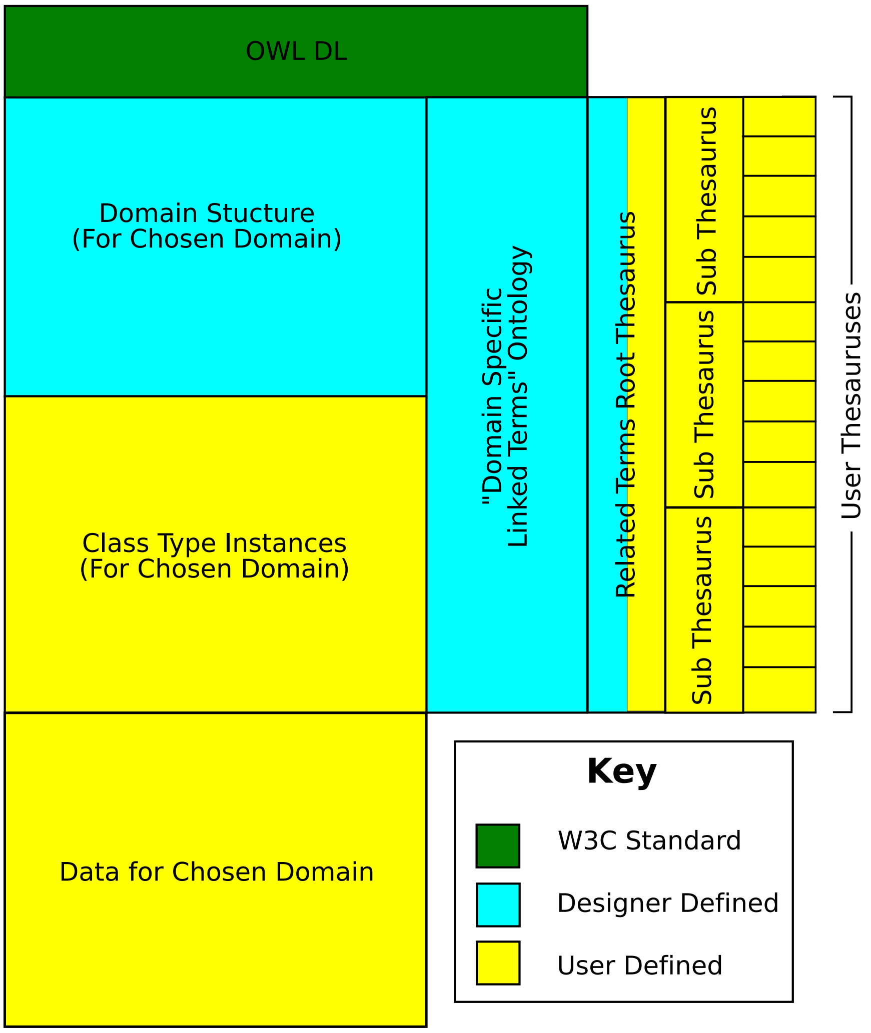

OWL DL
Domain Stucture (For Chosen Domain)
Class Type Instances (For Chosen Domain)
Data for Chosen Domain
"Domain Specific Linked Terms" Ontology
Related Terms Root Thesaurus
Sub Thesaurus
Sub Thesaurus
Sub Thesaurus
User Thesauruses
Key
W3C Standard
Designer Defined
User Defined

FIGURE 3.3: Architecture for Defining a NL Querying System Domain

3.3 Entity Recognition Engine

The framework for the NL querying system will take the initial NL query input for the user and input it into an entity recognition engine much like ANNIE, (see section 2.1.4.1), if not an adapted version of it. ANNIE requires two domain dependent components, gazetteers and grammar rules.

To save time gazetteers could be generated from the instantiation data in the database. If the gazetteers are generated in this way it would be useful to preserve the link between the gazetteer and the database. This would make it possible to store a richer representation of the extracted entities. It may not always be possible to determine exactly what entity from the database an extracted entity is, but it should be possible to generate a limited list of candidates. Recognised entities may not just be the entity itself but also some associated value, e.g. 800,000 US Dollars. Therefore the extracted entity also needs the ability to be able to store that as a separate part of the entity that also needs to be extracted.

3.3.1 For MusicBrainz Domain

In the Musicbrainz domain there are three main entities that need to be recognised:

1. Actors: These may be musicians but could be producers, engineers, composers, etc. An actor could represent an organisation or a group not just an individual.
2. Recordings: This can be broken down into two further categories, album names and track names.
3. Actions: This will generally be a musical action such as singing or playing an instrument but this can also includes non-musical actions like being the recording label or production engineer.

Three/four gazetteer lists would be required to do this. The generation of these lists should straightforward as when new albums are submitted, triples are created that store the album and track names. When new actors and actions are submitted similar triples are created for these as well. This data could be used to directly generate the gazetteer lists. This does not solve the whole problem. E.g. If a user is performing a search to see who recorded a particular album, if this album title is quite long it is quite likely they may not get the title absolutely correct. Frank Zappa's "Ship Arriving Too Late to Save a Drowning Witch." There are many ways that a user could type in this title incorrectly or because it is so long they may not even give the full title. In any case with a title so long, one mistake in the title should not prevent the album being found. This is why the grammar rules that ANNIE uses are required, so that is a perfect match cannot be found, if a very good match is found at least it can be considered.

It should be possible to design reasonably generic rules for detecting entities that are ill-formed. I.e. If you cannot find a perfect match look for entities that are a very close match to the entity in the query. Even though these rules could be made generic, this does not make the task straightforward. This is because measuring what is a close match and what is not depends on a number of factors. How many words are there in the entity? How common are the words in the entity? How disordered are the words in the user generated entity? etc. This could be taken even further, e.g. how misspelt are the individual words are in the entity. This is not something that ANNIE is capable of doing and until the other measures are implemented it is difficult to determine how useful this measure would be.

3.3.2 For Comb*e*Chem Domain

The first gazetteer that would be required for the Comb*e*Chem project is for words that make up chemical compound/element names. Another gazetteer that may be required is for measurements, most of these would probably be SI units but there may be others that are chemistry specific. In chemistry there is a large number of acronyms taken from different sources (Brown 96), these would also need to be encapsulated in one or more gazetteers.

Some quite complex grammatical rules will also be required for the Comb*e*Chem domain. INChIS [6] are a complex sequence of numbers, letters and symbols, e.g. C6H6 1-2-7 2-3-8 3-4-9 4-5-10 5-6-11 7-12, represents benzene, a relatively simple organic compound. This should be definable in ANNIE assuming it will allow any legitimate context-free grammar rule to be defined.

Chemical compounds present another potential problem, take 1,1,1-trichloroethane as an example. Unless ANNIE had trichloroethane as a word in one of its gazetteers, it would not be able to find it. This is because trichloroethane is a type of ethane with three bonds to chlorine rather than hydrogen atoms. Storing trichloroethane, trichloroethene and dichloroethane, etc. would be excessive and it would also be very difficult to be comprehensive. To expect the user to break up all these words with hyphens, so trichloroethane becomes tri-chloro-eth-ane, making it possible for ANNIE to tokenize it, is optimistic and goes against the NL querying system ethos. To solve this problem ANNIE would have to be modified in the tokenizing phase. The tokenizer would need to tokenize certain sequences of letters, this should not make it significantly slower because the NL queries are relatively short. A greater problem would be deciding when and when not to tokenize.

A gazetteer or similar would be needed to store the letter sequences. Every word would need to be checked to determine whether it was an entity. An algorithm would start at the beginning each word. If it found a sequence of letters it would remove them and using the remainder of the word search for another sequence of letters. This would be repeated until either a sequence of letters could not be found in the remaining string or the remaining string had no letters. If the latter occurred this would indicate that it was an entity or at least part of an entity. This algorithm may not be absolutely reliable as it may choose the wrong sequence of letters, if one sequence was the head of another sequence, e.g. ol as in eth-an-ol and oleic. To avoid this letter sequences could be tagged, stating whether they can appear at the beginning, middle, end or a combination of locations in a word. It may also be useful to link shorter letter sequences with longer ones so that if an entity is not found the algorithm can work back to try to find the longer sequence instead.

One problem with using the algorithm is that it may classify certain words as chemical compounds even if they are not. E.g. take the query “Who tried experiment X with trichloroethane?”. If both ‘tri’ and ‘ed’ may be defined letter sequences, then ‘tried’ could be considered as an entity. To avoid this, all words that have been determined

[6]IUPAC-NIST Chemical Identifiers (http://www.iupac.org/inchi/)

as potential entities must be checked against a dictionary. If a word has definitions for it but none of these definitions are nouns, then this word should be discounted as an entity. This may still not find all the words that are chemical compound entities or discard all those that are not. However when dealing with natural language, certainty is never guranteed. Once the NL querying system is implemented this algorithm may need to be modified, if it misses to many entities or finds to many false positives.

As well as the algorithm finding entities that are verbs, adverbs or adjectives, it may also find entities that are not real chemical compounds, this is not necessarily a bad thing. E.g. 1,1,1-trichloroethene looks like a perfectly acceptable chemical compound, but it is actually chemically impossible. There is no way of avoiding these false positives without making the grammar rules very specific. In some ways, detecting such words as chemical compounds is a good thing, as there may be data stored on such entities stating that they are chemically impossible. Also allowing them to be parsed as an entity but then not finding any data on them, will mean that the error message returned can inform the user that they have probably misformed the chemical compound and not that the system has not understood the query.

As previously stated individual words may not be whole chemical compound entities. Therefore a grammatical rule is needed to build these chemical compound entities. Only a simple rule should be needed, which groups all adjacent chemical compound entity words into one entity.

3.4 NL Parser(s)

Section 2.1.2.6 explains how by using more than one NLP it is possible to obtain a much richer output than could be achieved with one NLP. It also makes it clear that for it to be worthwhile these two NLPs must go about generating their parse trees in different ways and ideally produce different types of output so that a greater richness is produced. Section 2.1.2.6 showed how the Stanford parser and Minipar use different types of grammars, rule-based and principal-based respectively and produce different types of output, concrete syntax parse tree versus dependency parse tree/triples.

Using two NLPs means the system requires an additional component to reconcile the parse trees. This component is discussed in section 3.5. Building an NL querying system is not straightforward, using an incremental implementation should simplify some of the complexity. The use of multiple NLPs is one aspect that makes the system more complex, therefore the first implementation increment of the system may be better served with the use of only one NLP. Both the Stanford parser and Minipar have been used successfully in the past (Klein 03) (Lin 98). Minipar however produces an output synonymous to that of a SPARQL query, i.e. a set of triples that can be linked together in a tree structure. Although it is unlikely that these triples could be able to be transposed directly into a SPARQL, it gives the potential to represent data in a consistent fashion, which may be helpful in the future. It may be useful in the next implementation increment of the system to use the Stanford parser, as this may assist in the design of the parse tree reconciliator because it will provide data on which NL queries each NLP handles best.

3.5 Parse Tree Reconciliator

Implementing the parse tree reconciliator in the first increment of the implementation is probably infeasible. Data is first needed to determine how to best reconcile different parse trees; using earlier implementations of the system where only use one or other NLP is used is a way of collecting some of this data. There are several different approaches that could be used to reconcile a parse tree.

1. Use both parse trees separately to produce two outputs.
2. Determine which parse trees is more appropriate and use that one.
3. Merge the two parse trees.

The first approach is the most straightforward, all the reconciliator need do is invoke the rest of the system twice. It does however create the potential for a problem at a later stage. If both parse trees generate successful SPARQL queries that return results, which result should be returned to the user. If both have found the same result then there is no problem and the response can be even more assured. If the responses are different, what should the user be shown. It is unlikely that one result could be discounted for being infeasible, because measuring feasibility would not be possible across such a wide range of potential responses. Using this approach, the best strategy is probably to return both results, as this gives the user an impression of the confidence of the response.

Approach two requires more effort by the reconciliator. How should it go about determining which parse tree is more appropriate. Data collected from earlier increments of the implementation where the NLPs have been used separately will be needed. The best way to use this data to is to try to determine whether there is a correlation between certain traits the parse trees have and whether ultimately they produce a correct response. If this is possible these traits could be scored and the higher scoring parse tree should then be chosen.

Approach three is again more complicated. This approach could only be implemented in one of the later increments. It would be sensible to first implement approach two and assess its performance, as this third approach may be excessive. Merging the two parse tree could be achieved with a technique similar to that used in unification-based grammar NLPs (see section 2.1.2.2). Components of the two parse trees that are the same would be kept, those that do not contradict would also be copied to this reconciled parse tree and those that do would have to be resolved. As the format of the two parse trees from the Stanford parser and Minipar are quite different, this task may be difficult. In particular resolving contradictions will be problematic. However, proposing a more complete solution is infeasible until data from earlier implementations has been collected.

3.6 Machine-Readable Query Generator

The NL query is now a richly-represented and reconciled parse tree, but it is still only a machine-readable representation of an NL query and not a machine-readable query. To do this the words in the parse tree need to be converted into machine-readable entities. This can be achieved through a process of standardisation. Section 2.1.3 discussed how electronic thesauruses, use a machine-readable entity to represent a group of words that are synonymous. This is not quite powerful enough to generate sufficiently accurate machine-readable queries. This is why it is necessary to modify KnOWLer's Wordnet ontology to include features to make this possible. These additional features are illustrated in Figure 3.4.

3.6.1 Design of Domain Concept Mapping

To make an electronic thesaurus useful in a NL querying system context it needs to be able to map machine-readable entities representing lexical concepts (i.e. synonym groups) to real-world concepts that have been conceptualised using a domain ontology. This feature is not provided by Wordnet. Being able to do this is necessary to convert an NL query into a machine-readable query, as these domain concepts are the machine-readable representation used in the database.

Creating a mapping between a lexical concept and a domain concept is similar to the mappings between lexical concepts and word objects in KnOWLer's Wordnet ontology, except these mappings need to be slightly more sophisticated. They must be more sophisticated because only one mapping can be used by the system at a time, so the most probable mapping must be determined. The mapping between lexical concepts and word objects is binary, it is either there or it is not, mappings between lexical and domain concepts needs to be quantifiable. This quantification would probably have to be intially done by the creator of the mapping, based on how often they believe the mapping is will be correct. User quantification is unlikely to always be perfectly accurate, therefore a technique is needed for adjusting the correctness probability as the system is used. Ideally, this technique should not require excessive user input, as this could quickly become tiresome.

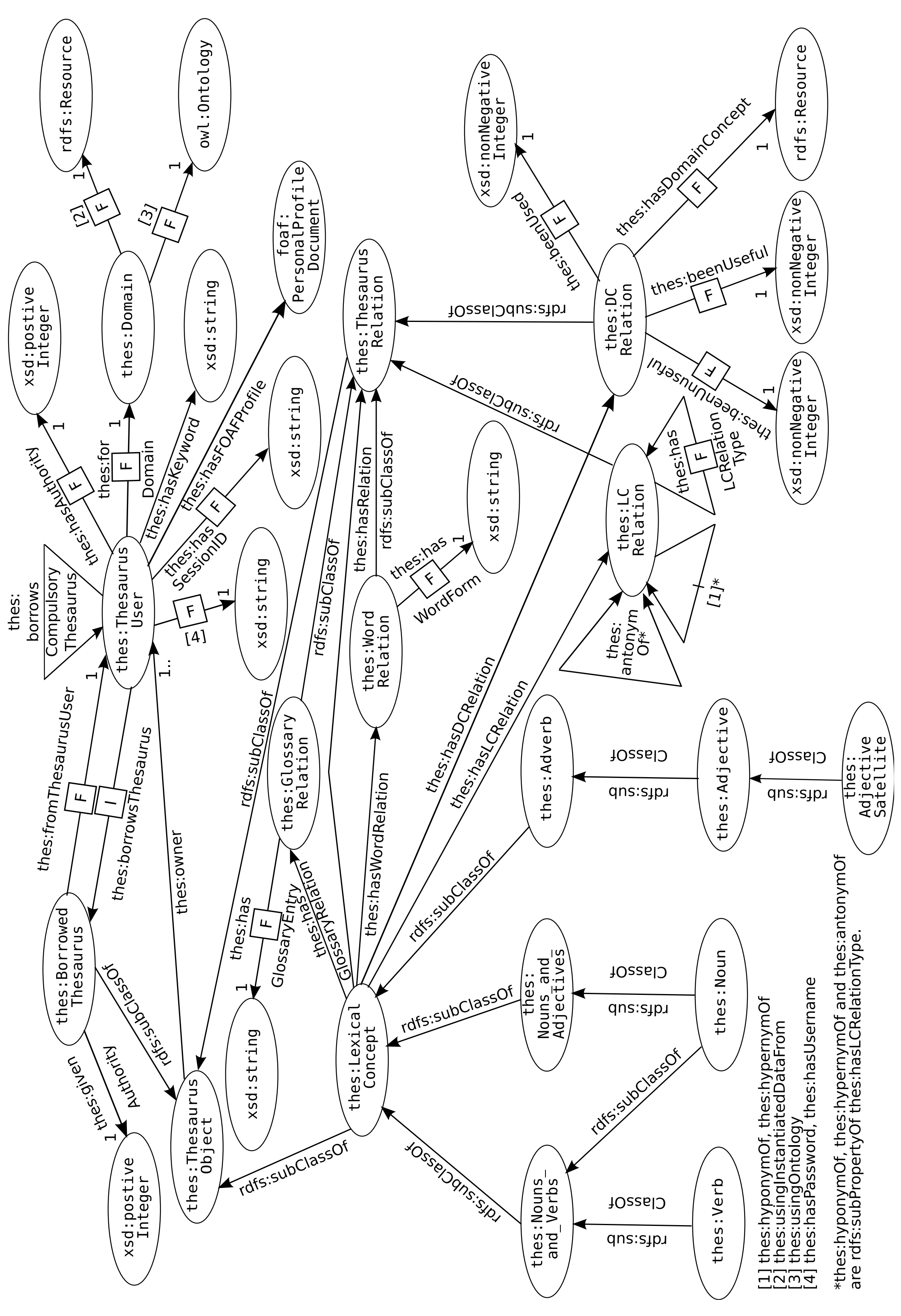


FIGURE 3.4: Thesaurus Ontology

Imagine that a user submits a question to the NL querying system. The system determines what the most probable lexical to domain concept mappings for each word or set of words are and generates a machine-readable query. After completing the remaining processes in the system, a response is given to the user. The user could then react to this response in a number of ways.

1. They could get the answer and start a new query / close the interface.
2. They could find that it is the answer they required and inform the system of this.
3. They could find that it is not the answer they required and ask the system to re-execute the query using the next most probable mappings.

These three different reactions can be used to help adjust the correctness probability of a mapping. The first reaction is neutral because starting a new query or closing the interface, does not prove whether the answer was what the user required. The second reaction is positive and the final reaction is negative. These three reactions can be used within a reinforcement learning algorithm to adjust the correctness probability. This reinforcement learning algorithm requires three parameters that can be stored as properties of the lexical to domain concept mapping. These three parameters are:

1. Been used
2. Been useful
3. Been unuseful

Each time a query is executed that uses a particular domain concept mapping, the been used property is incremented, depending on the reaction of the user the been useful, been unuseful or neither properties are incremented[7]. These three parameters are then combined in equation 3.1 to generate a correctness probability for the domain concept relation:

$$Correctness = \frac{BeenUseful - BeenUsed - BeenUnuseful}{BeenUsed \times 2} \tag{3.1}$$

Equation 3.1 will always produce a result between zero and one, zero being when every time the domain concept mapping has been unuseful and one when everytime it has been useful. This algorithm is not necessarily ideal, as the concept that the three reactions described previously are neutral, positive or negative, is a little oversimplistic. When a user finds the answer they are looking for there is no way of compelling them to inform the system, this would mean some positive reactions would be lost and considered as neutral. Negative results are less likely to be lost in quite such high numbers, as there is a motivation to inform the system so it can re-execute the query using different mappings

[7]Reaction 1: no increment. Reaction 2: been useful incremented. Reaction 3: been unuseful incremented.

to hopefully find the correct result. It may therefore become necessary to adjust the weighting of been useful parameter to correct for this.

There must be accounting for these lexical to domain concept mappings to keep track on who is adding and modifying them. It may only take one malicious/inexperienced user to irrevocably damage the system.

3.6.2 Design of User Accounts

Having user accounts will make it possible to keep track of objects within a thesaurus, as they will be assigned ownership to particular users, this however requires an additional class to be added to the original KnOWLer Wordnet ontology, a thesaurus user class. This class needs to be able to represent a user account. There are a number of requirements for a user account.

- A unique identity
- Security
- Maintaining consistency
- Classification of purpose
- Identification of the possessor
- Quantification of importance

All these requirements must be defined within the ontology so they can be exploited by any system that uses it. Figure 3.4 diagrams how these requirements have been designed into the ontology.

3.6.2.1 Unique Identities and Security

A unique identity within an ontology may seem implicit, as when a new user account object is created it is given it own URI. This although sufficient is not particularly user-friendly. A user would probably expect to be given or choose a username. Although the username could be extracted from the URI this places the limitation of never being able to change the username. It also makes the system less secure as it would be possible to guess a particular user's URI and then potentially use it to add malevolent entries to their thesaurus. A username property is therefore defined that has the user account class as its domain. Security is given to the user account by creating a password property in the same way.

3.6.2.2 Maintaining Consistency

Preventing two users from using the same account at the same time is what is meant by maintaining consistency. To ensure this, when a user account is logged in a session ID must be associated with that account to avoid a second login. For this reason a property entitled hasSessionID has been defined.

3.6.2.3 Classification of Purpose

As one of the purposes of thesaurus ontology is to map lexical concepts to domain concepts, a property needs to be associated with each user account to tell it where to find its domain concepts. This is not a straightforward as just having a property that points at the domain ontology. This is because as well as there being domain concepts within the ontology, there may also be instantiated domain concepts, see 3.1.2.1. To accomplish this a new class called Domain needs be defined, which has two properties, a pointer to the domain ontology and a pointer to the location of the directory that holds all instantiated data. Having a property that points at a ontology object means that the thesaurus ontology cannot be OWL DL. This can be tolerated as this property will not need to be reasoned over as it is only required as a unique URL of the domain ontology.

The property defining the domain of the thesaurus user account is only part of the classification of purpose, the other part is the keywords property. This property allows the user account to be classified more finely, so that it is tailored for a particular sub-domain, e.g. in chemistry the keyword organic may be ascribed to one user account and inorganic to another. There is no limit to the number of keywords that can be defined. Classification of purpose is essential as one of the features of the thesaurus ontology is the facility to borrow thesaurus objects created by other user accounts. Keywords provide a searchable method of discovery to help find user accounts that have the appropriate thesaurus objects to be borrowed. Keywords are a basic and generic searchable discovery method. As systems for certain domains are developed it should become more apparent what are other useful properties to search for user accounts on. How to introduce these other classification properties to the ontology may be a problem, as they will probably be both numerous and evolving. Hard-coding them into the ontology would be wrong as it would require regular amendments to the ontology. Therefore, a solution could be to introduce a generic classification property which could be instantiated to define the name and the range of each new property.

3.6.2.4 Identification of the Possessor

Identification of the possessor of the user account would be necessary so that a user could borrow thesaurus terms from colleagues who work in the same area. This could work both ways, after discovering a useful thesaurus a user would then be able to find out who that user is, as they may be a useful contact. Just a username is not sufficient to identify a user, especially as some user accounts may have multiple users. The thesaurus ontology allows Friend-Of-A-Friend (FOAF) profiles to be associated to a user account. This property also prevents the ontology from being OWL DL but as the purpose of this property is only to define the FOAF profile URL this will not need to be reasoned over using a DL reasoner.

3.6.2.5 Quantification of Importance

If a user account borrows many thesauruses, there may be words that maps to several different domain concepts. As the NL querying system only wants to avoid confusion by only returning one result at a time, choosing which thesaurus to use first is an issue. Therefore two properties are defined in the ontology to allow importance to be ascribed. The property used depends on whether the thesaurus is borrowed compulsorily or voluntarily. If a thesaurus is compulsorily borrowed then the importance property value defined at its creation is used by the borrowing user account. If a user decides voluntarily to borrow a thesaurus they may choose the importance. Currently these authority properties must have positive integer values. This will probably need to be improved or abstracted so that these values have some meaning, e.g. 5 = most authoritative, 1 = least authoritative.

3.6.3 Command Mappings

As described in section 2.1.5, commands need to be recognised because it is unlikely that any database query language will be able to cope with all types of commands they may be required to perform. In particular, as triplestore query languages are relatively new, few of these commands have been implemented. This is why there needs to be mappings from lexical concepts to commands, so that these commands can be determined and handled. These can then be used with domain mappings to build an machine-readable representation of the query. The following machine-readable query would represent the NL query, “Is Bohemian Rhapsody by Queen longer than Rhapsody in Blue by George Gershwin?” something like:

```
ASK(
  track(Bohemian Rhapsody).length WHERE artist(Queen)
  GREATER_THAN
  track(Rhapsody in Blue).length WHERE artist(George Gerswin)
)
```

ASK and GREATER_THAN are commands that need to be extracted, WHERE is just a joining command that does not need to be extracted. Track and artist are structural domain concepts that have been determined from mapping the entities to instantiated domain concepts, i.e. specific tracks and artists. length is a domain concept property that has been determined from the term longer than in the original NL query.

Although the gazetteers will be generated from instantiation data in the database, the recognised entities may not necessarily map to one entity in the database. Take the example above, if the query had just had "Rhapsody in Blue by Gershwin", Gershwin would have been determined as an entity but this entity may refer to at least two different entities in the database, i.e George Gershwin and Ira Gershwin. At this point, a decision needs to be made on how to proceed. Choosing one of the two is not an option as there is no criteria to choose on, like there is for domain concept mappings because these are non-quantifiable mappings. It would be possible to capture data over time using the reinforcement learning algorithm, but this would not help initially. A better option is to perform two queries one with each entity candidate. The majority of the time, only one query will be successful and that one will be used. If both queries return results, it may be possible to merge the results if they are the same or just return both results. This is discussed in greater detail in section 3.10.

3.7 Database Query Generator

The output produced by the machine-readable query generator is an intermediate representation, which cannot be processed directly by a database query engine. This is because it is necessary to have a full machine-readable representation of the query but not all of the query is processable by the database query engine, i.e. it can not perform certain data processing functions. This is why the database query generator is required. This component is designed to extract and substitute parts of the machine-readable query that can be converted into database queries. These are then given to the database query engine to execute.

Using the example given in section 3.6.3, track(Bohemian Rhapsody).length WHERE artist(Queen) and track(Rhapsody in Blue).length WHERE artist(George Gerswin) can be converted into SPARQL queries, somewhat like:

```
SELECT ?tracklength WHERE {
  ?trackinstance exp:trackInstanceOf
    <http://www.example.com/track/Bohemian_Rhapsody> .
  ?trackinstance exp:hasArtist <http://www.example.com/artist/Queen> .
  ?trackinstance exp:hasTrackLength ?tracklength
}
```

```
SELECT ?tracklength WHERE {
  ?trackinstance exp:trackInstanceOf
    <http://www.example.com/track/Rhapsody_in_Blue> .
  ?trackinstance exp:hasArtist
    <http://www.example.com/artist/George_Gerswin> .
  ?trackinstance exp:hasTrackLength ?tracklength
}
```

It is possible to see how parts of the machine-readable query map into the SPARQL queries. Assuming that the finalised machine-readable representation language is sound and complete it should be possible to scale this for more complex queries.

3.8 Database Interface

The database interface and the database itself are be pre-built components. The amount of time spent of developing these components is extensive, so developing bespoke components for the NL querying system would not be sensible. The purpose of the database query generator is so that the system can be seamlessly integrated with a pre-built database with its own interface. Choosing the correct database and the query interface is an important decision and one that must be made early in the design, as it may place restrictions on other components in the system. As the system is to be designed to demonstrate how SW technologies might improve NL querying systems this database will be a triplestore.

3.9 Choosing a Triplestore Application

There are many triplestore applications available for download, such as Jena[8] (Carroll 04), Sesame[9] (Broekstra 02), Kowari[10] (Wood 05), Redland[11] (Beckett 01) and 3Store[12] (Harris 03). Several factors must be considered to determine which system is the best to use:

- How easy is it to find and communicate with the developers / other users of the system?
- Does the system provide interfaces with a familiar query language?

[8]http://jena.sourceforge.net/
[9]http://www.openrdf.org/
[10]http://www.kowari.org/
[11]http://librdf.org/
[12]http://sourceforge.net/projects/threestore/

- What is the triplestore implemented on top of, i.e. is it based on top of a relational database such as MySQL or Oracle or does it have some other lower-level structure? A familiar method is preferable so that the working of the triplestore can be understood. This may help determine why some queries may not work and aid design of faster queries.
- How fast is the system, both in importing and querying?

Jena uses a familiar SPARQL query interface, which we are familiar with. Jena can also be configured to store RDF data in a familiar MySQL database. However we do not know any of the developers and very few users. In comparison to 3Store the general speeds of Jena seems to be quite a bit slower (Lee 04).

Like Jena we do not know the developers of Sesame and we do not know many users. Sesame did not originally have a SPARQL query engine, it has it's own Sesame RDF Query Language (SeRQL) but a plug-in[13] has been built for it. Sesame can use MySQL as a repository (Broekstra 02). Using MySQL as a repository, reading in roughly the equivalent number of triples as for Jena in (Lee 04), Sesame appears to be almost three times as fast ($\approx$ 300 seconds compared to 844.247 seconds) (tri 05).

The developers of 3Store considered using both Jena and Sesame before deciding to build their own triplestore (Harris 03). In particular they found that Jena was not good at importing large amounts of RDF data and Sesame was quite slow at querying. Considering these factors, 3Store is a very strong contender because we know some of the developers, as well as a number of users; It supports the familiar query language SPARQL; It is based on top of a familiar MySQL database. 3Store was purposefully designed to store large amounts of RDF data so it should scale well.

3Store does not support inferencing especially well and therefore desoite using it initially it may not be wholly suitable for the task at hand. Jena does provide more support for inferencing and may well be a better candidate. This is discussed further in Chapter 5.

3.10 Response Generator

The response generator receives two separate inputs. One is results returned from the database queries, the other is the extracted commands. The extracted commands are basically the machine-readable query with result ID tags substituting the parts of the query that could be converted into database queries. These two inputs are then used with the data processing functions to generate a response for the user. Using the example given in section 3.6.3. The two inputs would look like this:

[13]http://sparql.sourceforge.net/design-spec/design.html

- **Extracted Commands:** ASK(#Result1 GREATER_THAN #Result2)
- **Database Query Results:**

```
<sparql>
  <head>
    <variable name="tracklength"/>
  </head>
  <results ordered="false" distinct="false">
    <result>
      <binding name="tracklength">
        <literal>355000</literal>
      </binding>
    </result>
  </results>
</sparql>
```

```
<sparql>
  <head>
    <variable name="tracklength"/>
  </head>
  <results ordered="false" distinct="false">
    <result>
      <binding name="tracklength">
        <literal>1028000</literal>
      </binding>
    </result>
  </results>
</sparql>
```

The values between the literal tags in the two database queries are then inserted in the extracted commands in the machine-readable query. First the GREATER_THAN command is executed, this returns false. Then the ASK command is run, this simply converts false into an answer that would be understood by a user, e.g. No. This response is then returned to the user.

Just returning a single word response to the user is sufficient to answer the user's query but it may not be enough to satisfy them that the answer is definitely correct. To increase the user's confidence in the answer, returning some of the processing steps would be useful. The machine-readable query would be a useful step to display. This is because it shows the domain concept mappings the system has chosen, which will help the user in determing whether their intended NL query has been answered. This will also assist them if and when they give feedback for use by the domain concept mapping reinforcement learning algorithm.

Displaying the machine-readable query would also be useful if more than one is generated. In section 3.6.3, it was discussed that an NL query may not provide enough information about an entity to determine for certain which entity it was referring to, so a list of entities may be used to generate a number of queries. Displaying all these queries to a user would be beneficial, as it would make the user aware that they did not define the entity conclusively enough, as well as showing them how they could do so in the future. At this point, how these multiple queries were resolved would also need to be displayed, to make the user aware what final response they receive is in reference to.

Along with the response given to the user a set of options will be given to the user. One requesting that the query be re-executed using the next most probable set of domain concept mappings. The second allowing the user to confirm that the response is correct and a third allowing the user to start a new query. As stated in section 3.6.1, these three reactions, have a negative, positive and neutral effect respectively on the reinforcement learning algorithm.

3.11 Summary of Conceptual Framework

The conceptual framework uses many of the ideas and techniques discussed in the background research in Chapter 2. The ideas adopted vary from using or adapting pre-built components, such as the NLPs, Stanford parser and Minipar, the entity recognition engine ANNIE and Wordnet and OpenThesaurus for domain concept mapping. To approaches on how to construct the system as a whole, such as the architecture suggested for an Intermediate Representation Language System (IRLS) in (Androutsopoulos 95) and the incremental representation demonstrated in Janus (Hinrichs 88).

The conceptual framework is designed to demonstrate how SW technologies can be used throughout a NL querying system, in an integrated manner, that makes it possible to interpret more sophisticated queries and then represent them in a richer way. SW technologies also allow the data for a domain to be stored in a logical manner, making it more straightforward to transform an NL query into query(s) that can be executed on a triplestore. SW technologies can also be used to help build up data for the system, as demonstrated in 3.6.2.

From this conceptual framework it is now possible to envisage how this NL querying system can be implemented. Early stages of this implementation are discussed in Chapter 4.

Chapter 4

Implementation

Having a conceptual framework now makes it possible to start building the components of the NL querying system. As stated several times in Chapter 3, the implementation of the system needs to be performed in an incremental manner. This begins by first building the components that are least dependent on the rest of the system. A number of these have already been implemented and are discussed in this chapter.

After these base components are built it becomes possible to build the components that rely on these, e.g. the parse tree reconcilator cannot be built until there is a component that provides NLP(s) output. Once all the individual components have been built, the incremental implementation can focus on making certain components more sophisticated, e.g. the parse tree reconcilator, as described in section 3.5, has increasingly sophisticated ways of reconciling the parse trees it receives.

Producing a intricate plan of these incremental implementations would be difficult and probably not unproductive. What is important is to consider is what other components a component will need to use once it is implemented. If these are not already implemented they should be completed first. A further problem with a intricate plan is that it reduces adaptability if a certain component behaves unexpectedly. Therefore keeping a flexible plan will allow the implementation to evolve over time.

4.1 Parser Interface

Minipar and the Stanford parser are the two NLPs that have been chosen to for use in the NL querying system, (see section 3.4). During the system development it will be useful to see the parse trees that each NLP returns for various NL queries. Therefore a parser interface has been built that returns the outputs of both Minipar and the Stanford parser. This parser interface is a web page that simply displays the text-form of each parse tree, as can be seen in Figure B.1.

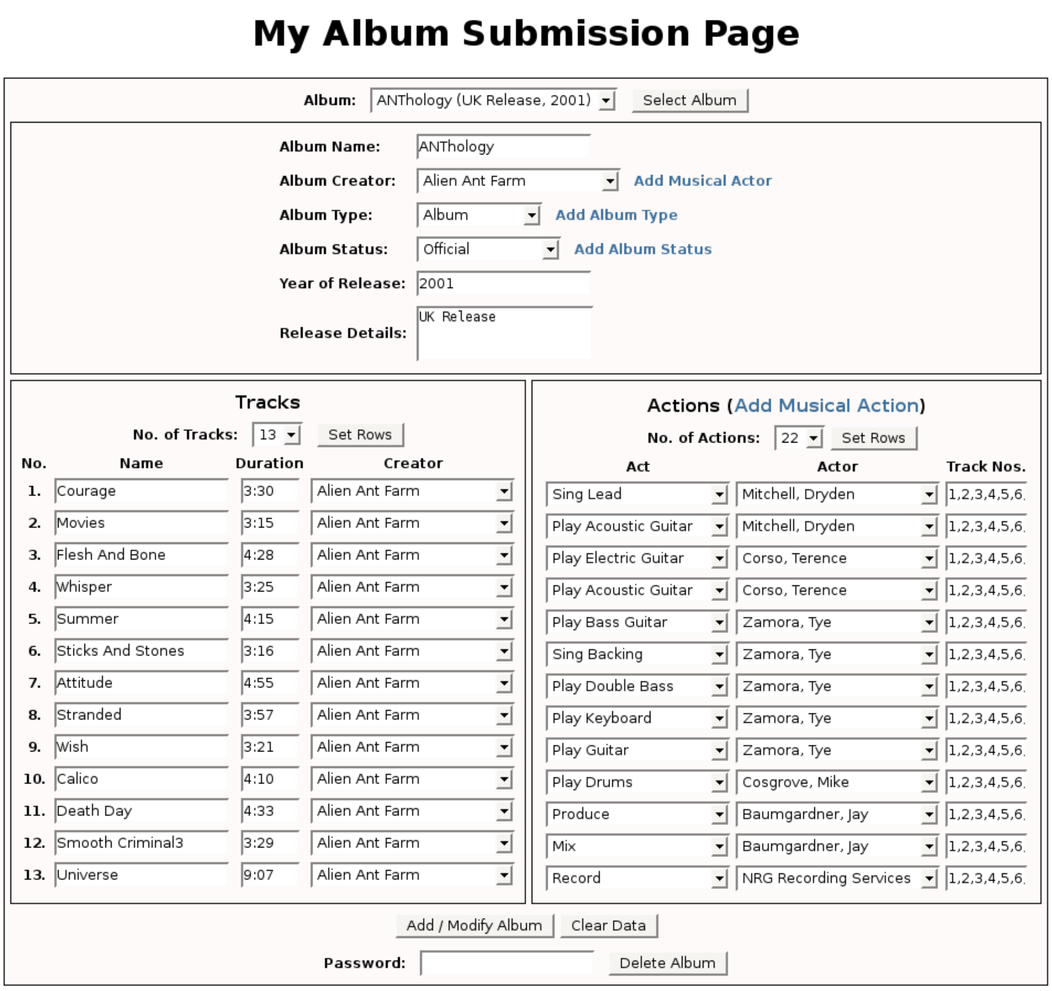
The parser interface is slightly more sophisticated than it may first appear. The Stanford parser uses the English factored lexicon, that is quite large ($\approx 250MB$ of memory) and takes 15-20 seconds to load on a 3GHz hyper-threaded Pentium 4. To avoid having to load the lexicon each time a query is executed, a client-server relationship has been set up. This dramatically decreases the processing time by allowing the server side to keep the lexicon in memory and the client to connect each time it has a NL query to parse.

4.2 Importing Data

Importing data is liable to be a time-consuming task. It is important to be aware of the data needs to be imported to ensure an appropriate method is used. The first implementation of the NL querying system will be for the MusicBrainz domain. The ontology for this domain, as described in section 3.1.2.1, has been heavily adapted from the original ontology provided by the MusicBrainz website. This means that porting data provided by the MusicBrainz database is not feasible. Another option is to screen-scrape data from music websites, this however gives very little guarantee of reliable or complete data. The screen-scraping tool may struggle to retrieve all the data because the format varies so widely. If the tool could retrieve the data perfectly, there is a high probability that data will be unreliable, as is often the case with web data.

The best solution for importing data is to use manual form-based data entry. It may be more time-consuming but the data will be more trustworthy as it has not been created by a third party. By having the original data, it does not matter if it is not absolutely correct because the answers given by the system can be checked against this original data. If these two tally then the system can be said to have passed the test for that particular NL query.

As form-based entry can be quite slow, it is essential that thought is given to designing the forms to facilitate fast and accurate input. The form displayed in Figure 4.1 is laid out in a logical way so that a user must first enter the main details about the album, i.e. its name, creator, year of release, etc. They can then enter more specific details that are of variable length, such as the tracks and the actions performed upon the album. The form tries to use drop down boxes wherever possible, using SPARQL queries to generate the options that need to be displayed. This prevents the user from mistyping an entry, it should also reduce the time taken to submit an entry. The form provides links to submit data for other entities in domain, such as musical actors, album types, album statuses and musical actions. The entry forms for these can be found in the Appendix, from Figure B.2 to Figure B.5.



My Album Submission Page

Album: ANThology (UK Release, 2001) | Select Album

Album Name:	ANThology
Album Creator:	Alien Ant Farm — Add Musical Actor
Album Type:	Album — Add Album Type
Album Status:	Official — Add Album Status
Year of Release:	2001
Release Details:	UK Release

Tracks

No. of Tracks: 13 | Set Rows

No.	Name	Duration	Creator
1.	Courage	3:30	Alien Ant Farm
2.	Movies	3:15	Alien Ant Farm
3.	Flesh And Bone	4:28	Alien Ant Farm
4.	Whisper	3:25	Alien Ant Farm
5.	Summer	4:15	Alien Ant Farm
6.	Sticks And Stones	3:16	Alien Ant Farm
7.	Attitude	4:55	Alien Ant Farm
8.	Stranded	3:57	Alien Ant Farm
9.	Wish	3:21	Alien Ant Farm
10.	Calico	4:10	Alien Ant Farm
11.	Death Day	4:33	Alien Ant Farm
12.	Smooth Criminal3	3:29	Alien Ant Farm
13.	Universe	9:07	Alien Ant Farm

Actions (Add Musical Action)

No. of Actions: 22 | Set Rows

Act	Actor	Track Nos.
Sing Lead	Mitchell, Dryden	1,2,3,4,5,6,
Play Acoustic Guitar	Mitchell, Dryden	1,2,3,4,5,6,
Play Electric Guitar	Corso, Terence	1,2,3,4,5,6,
Play Acoustic Guitar	Corso, Terence	1,2,3,4,5,6,
Play Bass Guitar	Zamora, Tye	1,2,3,4,5,6,
Sing Backing	Zamora, Tye	1,2,3,4,5,6,
Play Double Bass	Zamora, Tye	1,2,3,4,5,6,
Play Keyboard	Zamora, Tye	1,2,3,4,5,6,
Play Guitar	Zamora, Tye	1,2,3,4,5,6,
Play Drums	Cosgrove, Mike	1,2,3,4,5,6,
Produce	Baumgardner, Jay	1,2,3,4,5,6,
Mix	Baumgardner, Jay	1,2,3,4,5,6,
Record	NRG Recording Services	1,2,3,4,5,6,

Add / Modify Album | Clear Data

Password: | Delete Album

FIGURE 4.1: Album Submission Form

4.3 A System for Creating Domain Mappings

The ontology defined in section 3.6 is only the specification of the system for creating domain concept mappings, this specification needs to be implemented to make the system. The first aspect of the system that needs to be implemented is the user accounts. In section 3.6.2, a list of six properties that these user accounts require was discussed and it was explained how these were to be conceptualised within the ontology. These properties now exist but they need to be implemented within the system.

4.3.1 Unique Identities and Security

The system implements unique identities and security by providing a login for the system. A screenshot of the login screen is provided in Figure B.7 in the Appendix. Once a user submits their account details they are checked against the entries in the triplestore to see if they match. The string entered into the password field is masked and when submitted is MD5 hashed and compared to the MD5 hash of the password for that user to maintain security. The system does not currently use secure HTTP so the password is sent in plaintext but this can be implemented later if necessary. Once the user is logged in any additions or modifications they make to thesaurus objects will cause those objects to be tagged as being owned by them, although not necessarily solely by them.

4.3.2 User Account Consistency

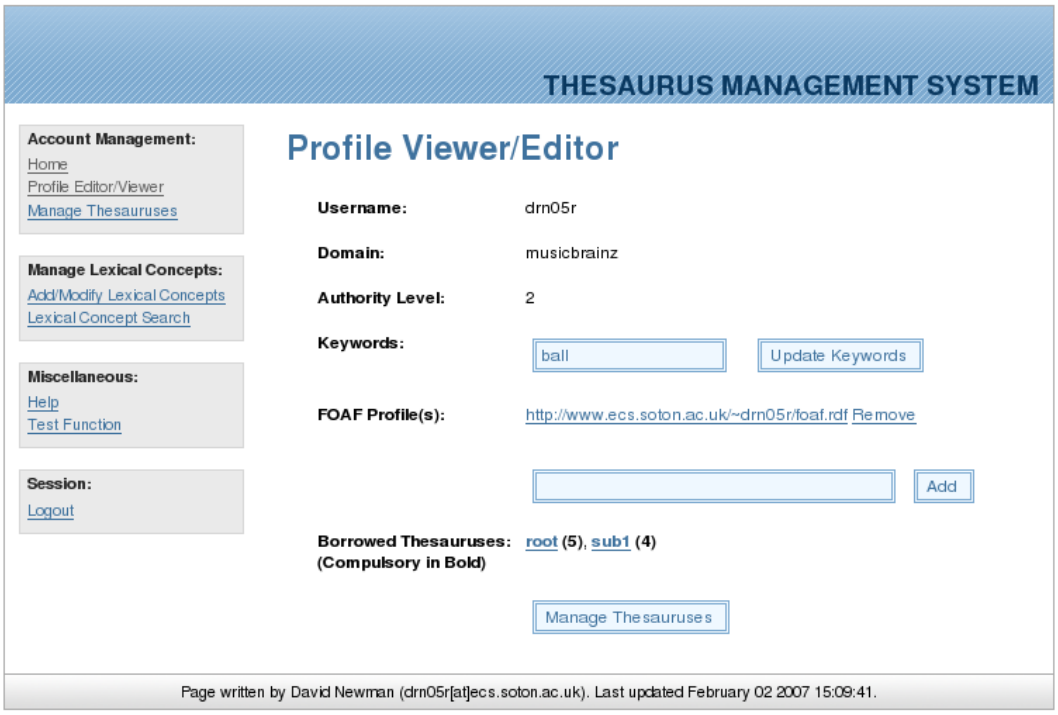
The session ID property is provided in the ontology to prevent two users logging in to the same account at once. Once a login is successful, an approximately unique session ID is created for the session and stored in the triplestore. It is also stored as the value of a cookie, identified as an encrypted version of their user ID. Everytime the user performs an operation the system checks to see whether the session ID the user was given at login, which is passed in the header of the web page, matches that stored in the cookie. If so the cookie's time to live is reset to an hour. If however there is no match or the cookie has timed out the user will be automatically logged out, before any operation is performed. Having the session ID stored in the triplestore allows recover of a user's username, user ID and domain. This reduces the the number of variables being passed in the header, improving both tidiness and security.

If a user tries to login while another session is still active or has not been logged out of properly, a warning is displayed to the user. If there is no cookie associated with the session, then the warning informs the user that either the session is running on another computer or it has become non-active, otherwise the user is warned that the session is still active. The user can then either choose to terminate the session or not. Both options will return the user to the login page but the former will also expire any active cookie and delete the session ID from the triplestore, preventing the user getting the same warning message if they try to log in again. In this case it is necessary to login twice to maintain security. This is to prevent a malicious user from creating a page that links to this warning message page with the appropriate values passed in the GET and POST headers because then they could simply click on a button to gain access to the whole system.

Using cookies prevents an unauthorised user using a session that has been not logged out of properly as it times out after one hour and the user cannot perform URL injection to gain access. The way this has been implemented prevents any possibility of two users being logged in at the same whilst avoiding deadlock if a session is not logged out of properly.

4.3.3 Classification of Account Purpose

As already described in section 3.6.2.3, a thesaurus user account can be classified for purpose using keywords and in the future possibly other more specific classification mechanisms. The user profile editor provided by the system, (see Figure 4.2), allows these keywords can be managed.



Username:	drn05r
Domain:	musicbrainz
Authority Level:	2
Keywords:	ball
FOAF Profile(s):	http://www.ecs.soton.ac.uk/~drn05r/foaf.rdf Remove
Borrowed Thesauruses: (Compulsory in Bold)	**root** (5), **sub1** (4)

Page written by David Newman (drn05r[at]ecs.soton.ac.uk). Last updated February 02 2007 15:09:41.

FIGURE 4.2: User Account Management for Thesaurus Management System

4.3.4 Identification of the Possessor

This user profile editor also allows the user to add or remove FOAF profiles for the purpose of identifying the possessor(s) of the account. The user profile editor can also be used to view other user's profiles. Whilst viewing a user profile the user can click on the user's FOAF profile link. This loads a page that displays a formatted version of their FOAF profile, as illustrated in Figure B.6 in the Appendix. This formatted FOAF profiles gives details of friends a user has and projects that they have or are working on, which may assist the user in determing whether it would be useful to borrow their thesaurus or not.

4.3.5 Quantification of Importance

The profile editor also gives useful information such as the thesauruses they have borrowed and whether they are compulsorily or voluntarily borrowed. It displays the authority level of the thesaurus, which is their quantification of importance.

The profile editor provides a link, that can also be found via the main menu, to a form allowing the user to manage their borrowed thesauruses. A screenshot of this form can be found in the Appendix as Figure B.8. This form allows the users to add and remove voluntarily borrowed thesauruses. When a user adds a voluntary thesaurus they must also specify its authority level, this quantifies its importance to this particular user. Compulsory thesauruses have a fixed authority level that is specified for that thesaurus' account when it is created. At present there is no facility to create accounts except by manually creating the RDF/XML and importing it into the triplestore.

4.3.6 Thesaurus Discovery

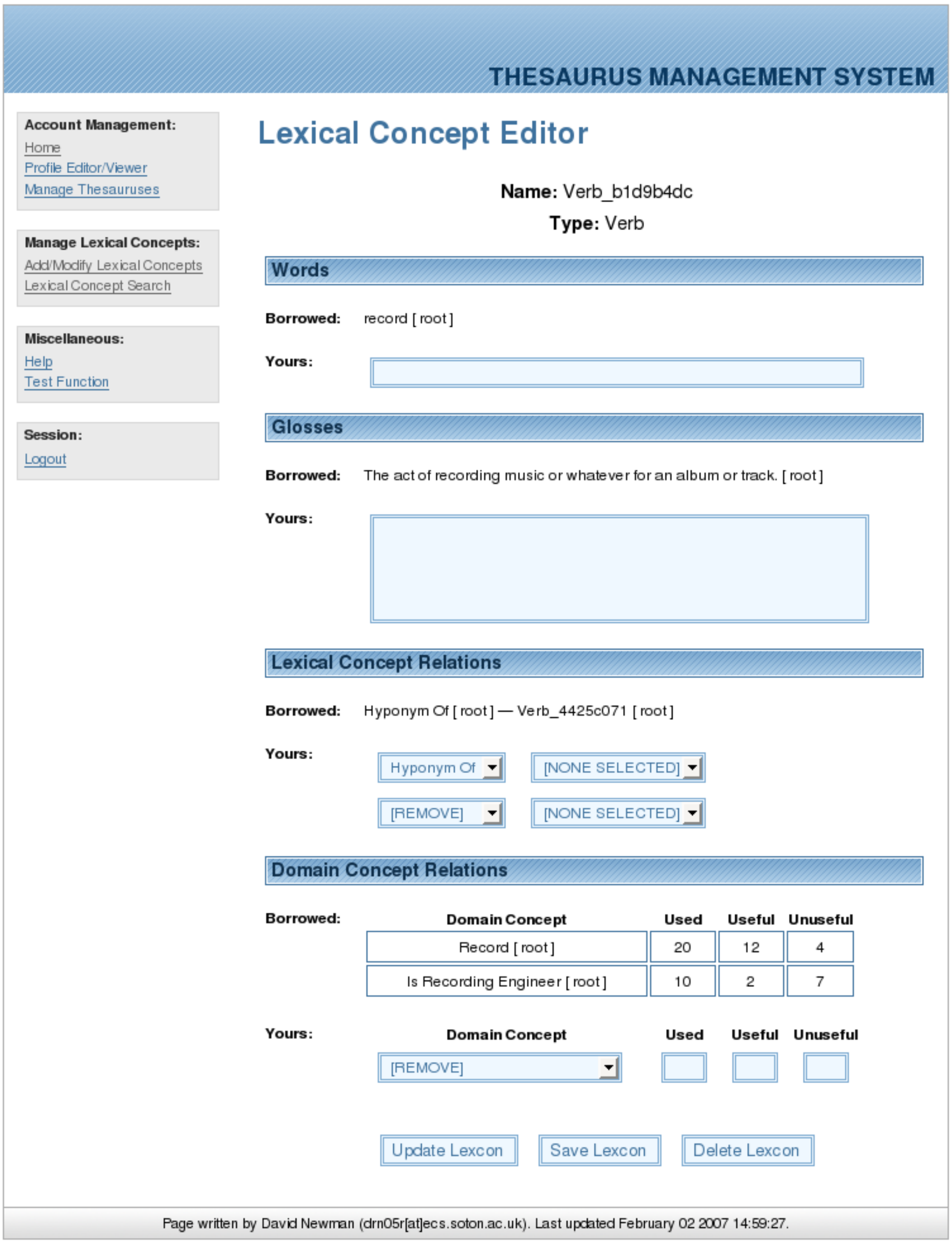
To help users find thesauruses, the manage thesauruses form, (see Figure B.8), provides a search engine based on keywords. From these search results the user can view the profile for the account and even the FOAF profiles associated with that account to determine whether they wish to borrow that account's thesaurus.

4.3.7 Managing Lexical Concepts

User accounts and thesaurus discovery is only part of the system. One of the most important features is the facility to manage lexical concepts. Figure 4.3 is a screenshot of the form provided to manage lexical concepts. At the top of the form there is the name of the lexical concept, this is also, when put with URL folder[1] for lexical concepts, the URI for this particular lexical concept. Next, the type of the lexical concept is defined. Following this there are the four main components of the lexical concept, its words, glossary entries (glosses), lexical concept relations and domain concept relations. For each of these four components the borrowed relationships are displayed followed by the user's relationships. The user can edit or create their own relationships for any of the four different components. This is straightforward for the first three components, although defining lexical concept relationships is presently not especially user-friendly. Currently the user can select from a list of lexical concept names, which is not particularly useful as the name does not state what the lexical concept represents. This could be achieved by allowing the user to search for a word in-form, then a list of lexical concepts that contain that word could be given to the user to choose from. To make it easier to select the correct lexical concept from this reduced list, a subset of words associated to it could also be provided.

Adding or editing domain concept relations is slightly more sophisticated. When defining the domain concept to link to, the user must also define the three parameters, used, useful and unuseful, which make up the reinforcement learning formula in equation 3.1. It may be possible to simplify this for the user but to do this in a way that still captures both the probabilities and the certainty, i.e. both the ratio and size of the values, may be difficult.

[1]http://sanjay.ecs.soton.ac.uk/ drn05r/musicbrainz/lexcon/



THESAURUS MANAGEMENT SYSTEM

Account Management:
Home
Profile Editor/Viewer
Manage Thesauruses

Manage Lexical Concepts:
Add/Modify Lexical Concepts
Lexical Concept Search

Miscellaneous:
Help
Test Function

Session:
Logout

Lexical Concept Editor

Name: Verb_b1d9b4dc
Type: Verb

Words

Borrowed: record [root]

Yours:

Glosses

Borrowed: The act of recording music or whatever for an album or track. [root]

Yours:

Lexical Concept Relations

Borrowed: Hyponym Of [root] — Verb_4425c071 [root]

Yours: Hyponym Of | [NONE SELECTED]
[REMOVE] | [NONE SELECTED]

Domain Concept Relations

Borrowed:

Domain Concept	Used	Useful	Unuseful
Record [root]	20	12	4
Is Recording Engineer [root]	10	2	7

Yours:

Domain Concept	Used	Useful	Unuseful
[REMOVE]			

Update Lexcon | Save Lexcon | Delete Lexcon

Page written by David Newman (drn05r[at]ecs.soton.ac.uk). Last updated February 02 2007 14:59:27.

FIGURE 4.3: Lexical Concept Management for Thesaurus Management System

4.4 Summary

So far the implementation has concentrated on three main areas:

- NLP interface
- Data acquisition
- Lexical to domain concept mapping (through thesauruses)

The purpose of implementing the first of these components early has been to gain an understanding of how an NLP represents an NL query as this underlies the whole NL querying system. Being able to play around using the interface described in section 4.1 should provide insight for how to build the parse tree reconcilator, the machine-readable query generator and even the entity recognition engine.

The last two components also needed to be implemented early as they are required to capture data that is necessary to built and test most the components in the system. As a large amount of data needs to be collected to gain enough domain coverage to make it possible to user test the system, having the facility to import data early in development should make importing this data less painful.

One of the next components that can be built is the entity recognition engine. Firstly the gazetteers need to be compiled. This should be helped by already having some data acquired from the form-based submission system discussed in section 4.2. The grammar rules may be more difficult to construct, as they will probably have to be designed ad hoc dependent on the domain.

Two further components that could be constructed next are the command mappings and the data processing functions. Neither of these rely to heavily on other components, although the former relies on an increased understanding of the likely NL queries. However it will not be possible to complete these in one increment of the implementation. As work progresses further command mappings or data processing functions will probably be required. In this first increment it should be possible to formalise how these command mappings and data processing functions will be represented and then used by the system.

Providing an intricate plan of implementation would be difficult and probably counter-productive. Discussing exactly how and when each and every component for the system will be built would be foolhardy. Section 5.1 provides a rough timeline for when components for the NL querying system will be built, it is quite vague as a more specific timeline would be unlikely to be accurate.

Once the implementation has completed a sufficient number of increments it should be possible to user test it, that will help to start drawing some conclusions to the main research question. How these user tests will be designed is yet to be determined. Simply giving a test user a number of questions to find answers to would be no real test for a NL querying system at all.

Chapter 5

Conclusions

In the background research carried out in Chapter 2 a large number of existing NL querying systems / NLIDBs have been investigated. What is apparent about those that have been successful is that they concentrate on only one domain, e.g. Lunar (Woods 72), Planes (Waltz 78), etc. Even Ask (Thompson 83) that was designed generically, cannot be used for more than one domain concurrently. The main reason for this is due to the ambiguity that can arise in terms when they are applied to more than one domain. Therefore like NLIDBs that have gone before, the NL querying system proposed concentrates on one domain but to make it widely applicable the framework should allow swapping between domains with ease. This task is made easier through the use of SW technologies, as they facilitate wrapping up a domain into a nice bundle, i.e. an ontology. This has been exemplified by the GINSENG project (Bernstein 06).

Chat-80 is a system that uses a database of Prolog statements to perform inference to resolve NL queries; a triplestore can be used in a similar way. As previously described in Chapter 2, OWL is a language that can define logical relationships. It can be used to generate logical statements much like the Prolog statements used in Chat-80. By being able to create logical statements it should be possible to exploit the data to answers more complex NL queries than would be possible otherwise.

Currently the NL querying system uses 3Store as its chosen triplestore, this has little support for inferencing logical statements. It may be possible to find a tool that can be incorporated into the system that can provide this inference. A better solution may be to use a different triplestore application. GINSENG used Jena and its RDFS inferencing model. Jena also has an OWL DL inferencing model that can provide more sophisticated inferencing. This could be used over the MusicBrainz and Comb*e*Chem ontologies to allow the NL querying system to use inference as one of its tools to help find the correct answers to the more sophisticated NL queries submitted. At this stage a final decision on which triplestore application should be used needs to be made. Although we have less knowledge of Jena, it does provide features that are essential in producing a powerful NL querying system, making it the better option.

One of the other main advantages of using SW technologies is their support for amalgamating data from distributed sources. Currently the conceptual framework for the NL querying system takes advantage of this in a number of ways. Domain ontologies can be pulled in from different sources. FOAF profiles can be linked to from a user profile in the thesaurus management system. The system probably does not take advantage of this feature of SW knowledge technologies as much as it could. As the system is implemented more scenarios where this feature is beneficial may become apparent.

It was observed in section 2.3.1, that the Comb*e*Chem domain with its digital lab book, would benefit from having a NL querying system. This could be used for retrieving information whilst planning and conducting experiments and also for reviewing past experiments. A NL querying system provides advantages over a GUI; when it comes to finding specific values from a data book, a single typed query might well be quicker than a number of clicks to find the required table that still may need to be studied to find the particular piece of data required. Despite the benefits that SW technologies provide, developing a generic NL querying system for this domain would be difficult because of its complexity and the degree of expert knowledge. Therefore, taking inspiration from the analogies used in the SmartTea and myTea project, (see sections 2.3.1.1 and 2.3.1.2), a simpler analogous domain has been determined, i.e. the MusicBrainz domain, (see section 3.1.2).

The Comb*e*Chem project ended in 2005. Since then, a related project has been started, entitled myExperiment[1] (De Roure 07). This project intends to assist scientists from varying different fields: bioinformatics, chemistry, social statistics and astronomy. The main goal of the project is to provide a Virtual Research Environment (VRE) to allow scientists to share and discuss their experiments, which is similar to one of Comb*e*Chem's goals. Part of the inspiration for myExperiment comes from the popular social networking site MySpace[2]. Scientists would like to be able to swap workflows and publications much like current users of MySpace share documents, photos and videos. Currently the project is in its early stages and bioinformatics is the main focus. As the project develops, more work related to chemistry and the Comb*e*Chem project will be completed making it possible to determine how a NL querying system for Comb*e*Chem could fit in.

One application of the NL querying system could be to provide a remote interface to the digital lab book when performing long timescale experiments. For example, a user may want to check the current temperature within the lab, whilst they sort out some paperwork in their office. A simple NL query could elicit this information without

[1]http://myexperiment.org/
[2]http://www.myspace.com

the user having to return to the lab. A NL querying system is far superior when the interfacing device is small, such as in a cellphone, because it does not require a large display or a high resolution for it to be used. A NL querying system using a standard messaging system such as Short Message Service (SMS) or even an instant messenger client over General Packet Radio Service (GPRS) would provide a useful way for chemists to keep track of their experiments in almost any location.

The main purpose underlying the Semantic Web paradigm, is to change the current web from one that stores human-readable knowledge as machine-readable data to one that can produce a greater amount human-readable knowledge by storing machine-readable knowledge, (see section 2.2.3). By design, a NL querying system that uses SW technologies meets these criteria but it also goes one step further by providing a natural language interface to find this knowledge.

5.1 Future Work

As already stated, producing a precise plan of when and in which order components of the system should be built is infeasible, as it is difficult to be certain of the dependencies each component until just before they are built. The command mappings and data processing functions are two components that have been cited as being suitable to develop next, as they only depend on components that have already been built. Once these are completed, a review process will be required to determine the components that no longer have dependencies on other incomplete components, these can then also be implemented. This review process will need to be repeated until all the components in the system are completed to an extent where data can flow through the whole NL querying system. At this point, designing and conducting user tests for the MusicBrainz domain would be useful. These tests would help determine how to modify the system as part of the incremental implementation process to produce a more effective NL querying system. With these modifications it may then be possible to conduct user tests for the system using the Comb*e*Chem domain.

5.1.1 Future Work Plan

A future work plan is necessary to structure the time between now and the completion of the PhD thesis. The plan should ensure that various stages of the the required research are completed promptly to allow enough time to write the thesis report.

5.1.1.1 May - July 2007

1. Install and get use to using Jena.

2. Import MusicBrainz data for complete components. (See section 4.2).
3. Write initial data processing functions library. (See section 3.10).
4. Determine how to represent natural language to machine-readable command mappings. (See section 3.6.3).
5. Compile an initial list of command mappings. (See section 3.6.3).

5.1.1.2 August - December 2007

1. Use ANNIE to build the entity recognition engine. (See section 3.3).
2. Build machine-readable query generator. (See section 3.6).
3. Build database query generator. (See section 3.7).
4. Build response generator. (See section 3.10).
5. Conduct self-testing.
6. Re-implement system using Stanford Parser.
7. Conduct further self-testing.
8. Implement basic parse tree reconcilator. (See section 3.5).
9. Conduct more self-testing.

5.1.1.3 January - June 2008

1. Compile and conduct first user tests.
2. Evaluate first tests.
3. Tweak components of the system as part of the incremental implementation, based on the results of the user tests.
4. If time permits, import data for Comb*e*Chem domain.
5. Compile and conduct second user tests, possibly on both domains.

5.1.1.4 July 2008 - Completion

1. Evaluate second user tests.
2. Write up thesis.

Appendix A

A.1 Minipar Output from a Question

Minipar can be set to produce two distinctly different outputs, a dependency parse tree and a set of dependency triples. This two outputs displayed below ate for the question

Dependency Parse Tree -

```
(
E0 (() fin C * )
1 (Who ~ N E0 whn (gov fin))
2 (recorded record V E0 i (gov fin))
E2 (() who N 2 subj (gov record) (antecedent 1))
3 (the ~ Det 5 det (gov thriller))
4 (album ~ N 5 nn (gov thriller))
5 (Thriller ~ N 2 obj (gov record))
6 (? ~ U * punc)
)
```

Dependency Triples -

```
fin C:whn:N who
fin C:i:V record
record V:subj:N who
record V:obj:N thriller
thriller N:det:Det the
thriller N:nn:N album
```

A.2 Key for Ontology Diagramming

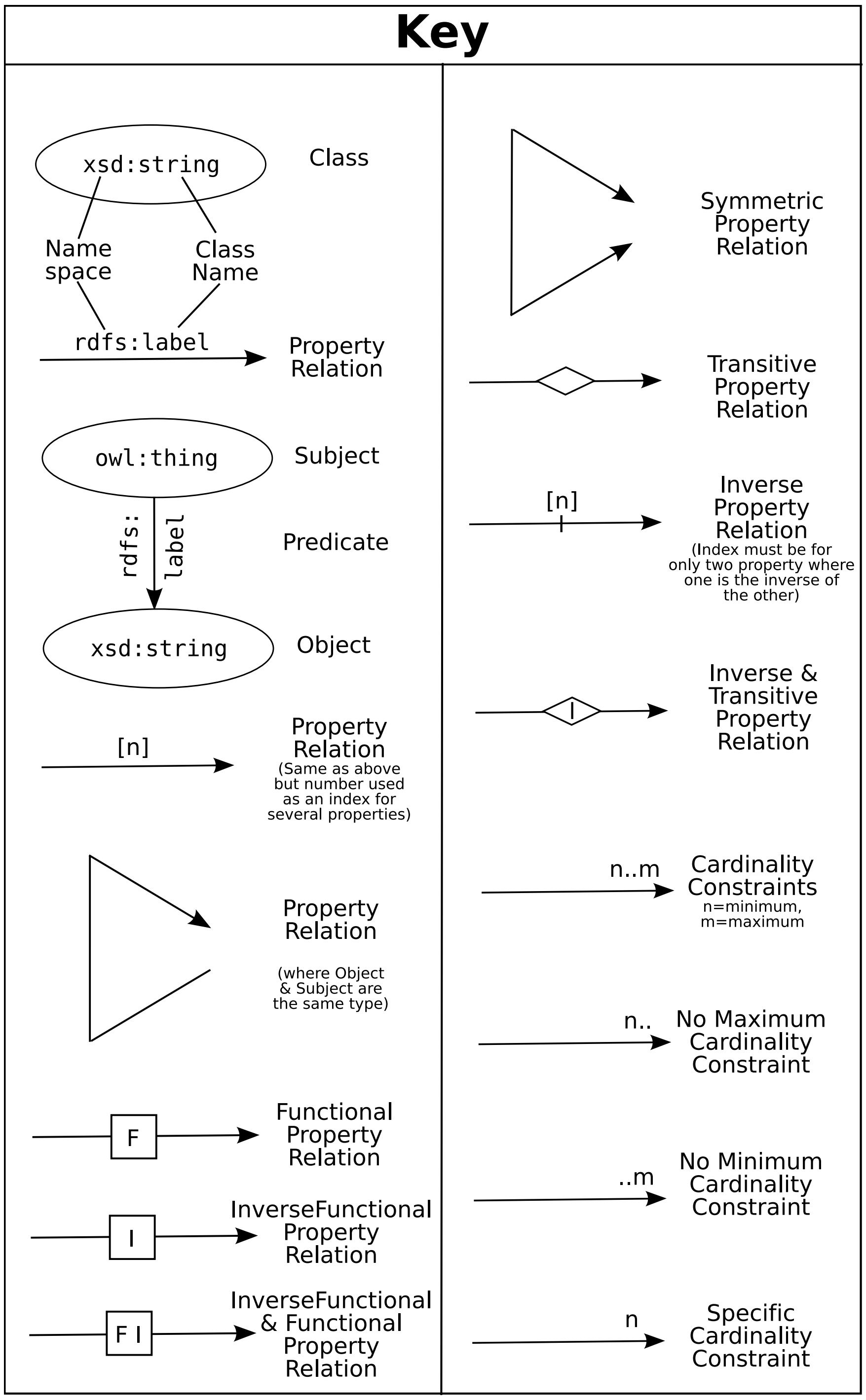

Key
xsd:string
Class
Name space
Class Name
rdfs:label
Property Relation
owl:thing
Subject
rdfs: label
Predicate
xsd:string
Object
[n]
Property Relation
(Same as above but number used as an index for several properties)
Property Relation
(where Object & Subject are the same type)
F
Functional Property Relation
I
InverseFunctional Property Relation
F I
InverseFunctional & Functional Property Relation
Symmetric Property Relation
Transitive Property Relation
[n]
Inverse Property Relation
(Index must be for only two property where one is the inverse of the other)
I
Inverse & Transitive Property Relation
n..m
Cardinality Constraints
n=minimum, m=maximum
n..
No Maximum Cardinality Constraint
..m
No Minimum Cardinality Constraint
n
Specific Cardinality Constraint

FIGURE A.1: Key for Ontology Diagramming

A.3 Diagram for Wordnet Ontology

This is a visualization diagram of the Wordnet Ontology created by the KnOWLer project at the University of Neuchâtel, based on Wordnet 1.7.1. It uses entities described in the key in Appendix A.2.

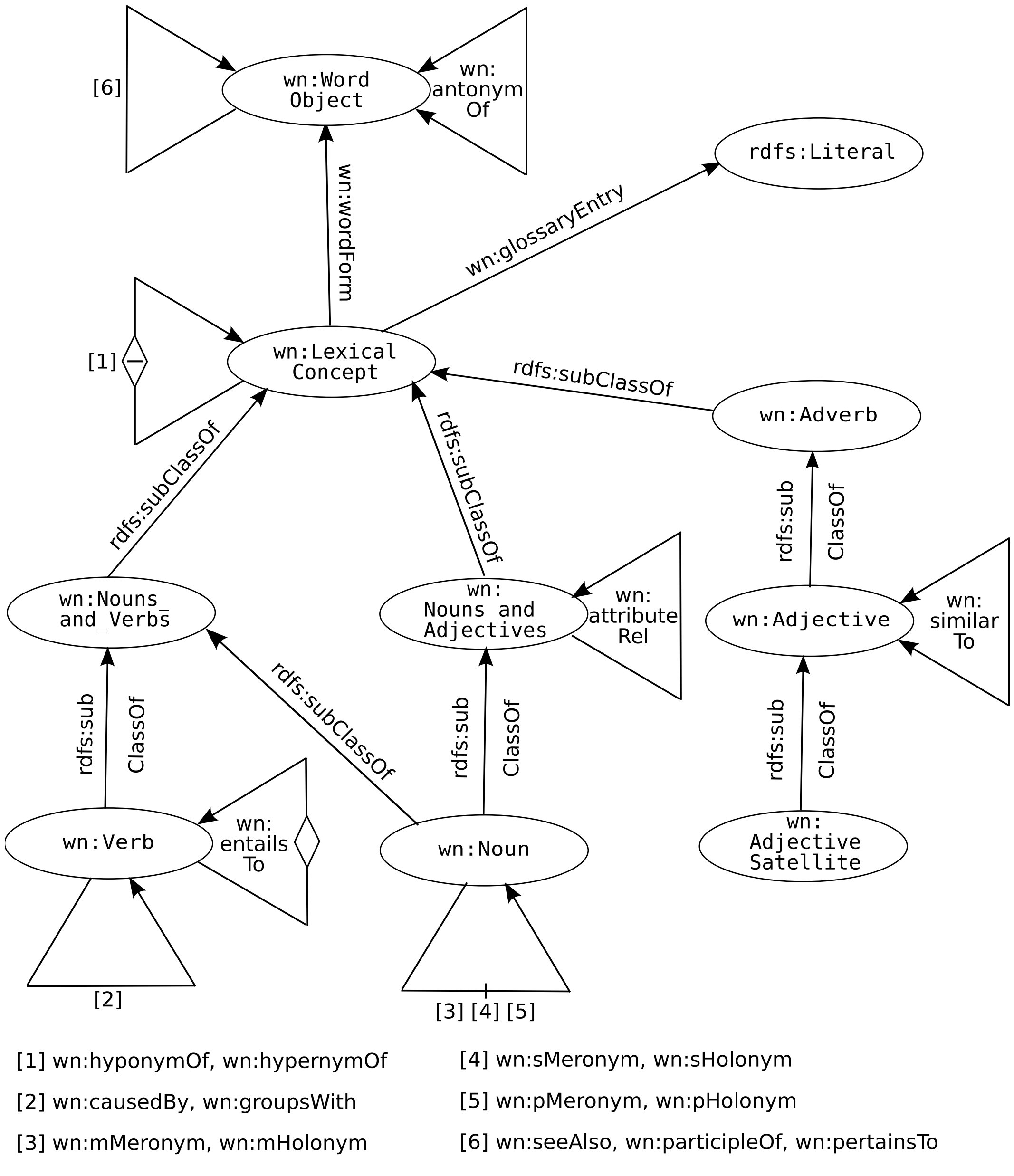
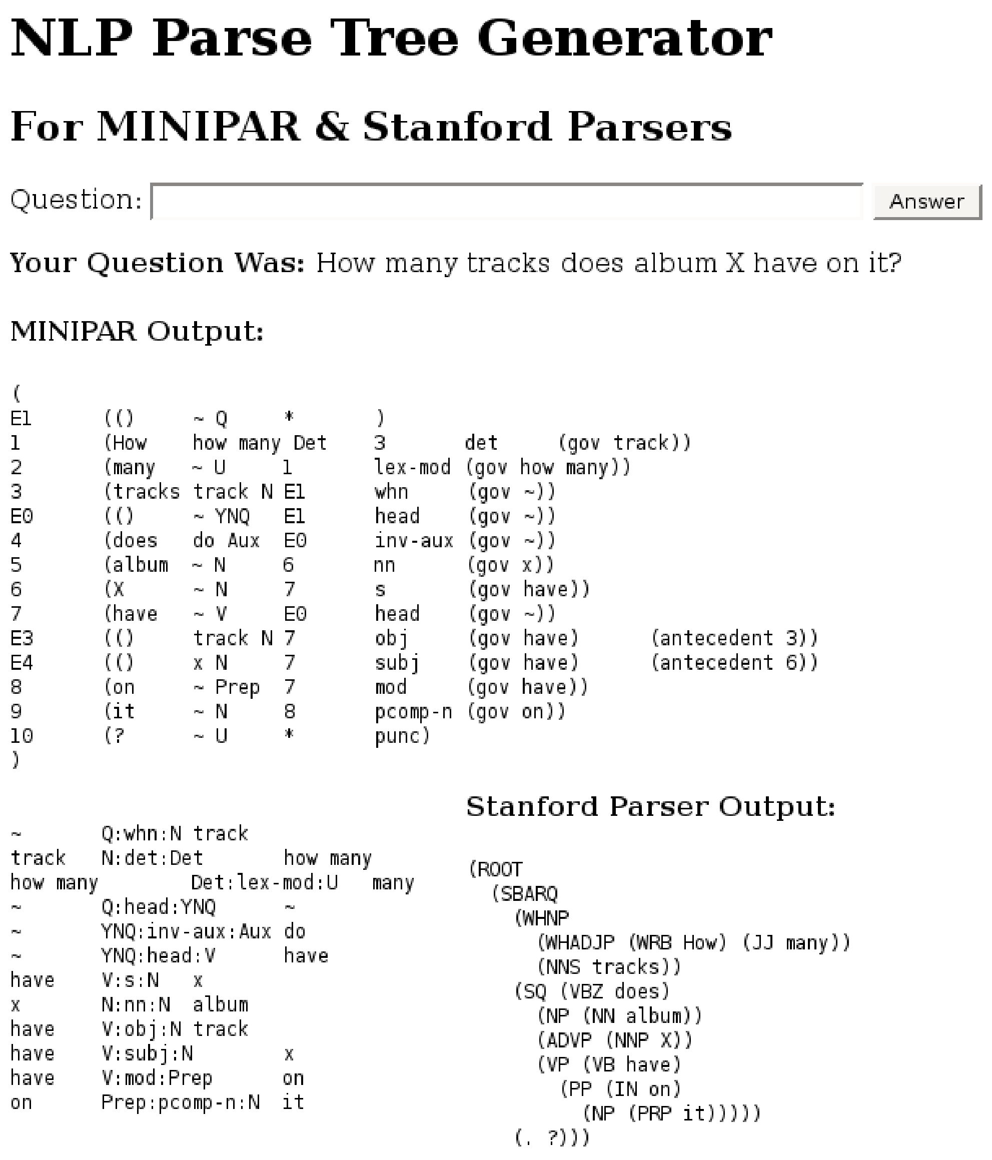


FIGURE A.2: Visualization Diagram of the Wordnet 1.7.1 Ontology

Appendix B

B.1 Screenshot of Parser Interface



NLP Parse Tree Generator

For MINIPAR & Stanford Parsers

Question: Answer

Your Question Was: How many tracks does album X have on it?

MINIPAR Output:

```
(
E1      (()     ~ Q     *       )
1       (How    how many Det    3       det     (gov track))
2       (many   ~ U     1       lex-mod (gov how many))
3       (tracks track N E1      whn     (gov ~))
E0      (()     ~ YNQ   E1      head    (gov ~))
4       (does   do Aux  E0      inv-aux (gov ~))
5       (album  ~ N     6       nn      (gov x))
6       (X      ~ N     7       s       (gov have))
7       (have   ~ V     E0      head    (gov ~))
E3      (()     track N 7       obj     (gov have)      (antecedent 3))
E4      (()     x N     7       subj    (gov have)      (antecedent 6))
8       (on     ~ Prep  7       mod     (gov have))
9       (it     ~ N     8       pcomp-n (gov on))
10      (?      ~ U     *       punc)
)
```

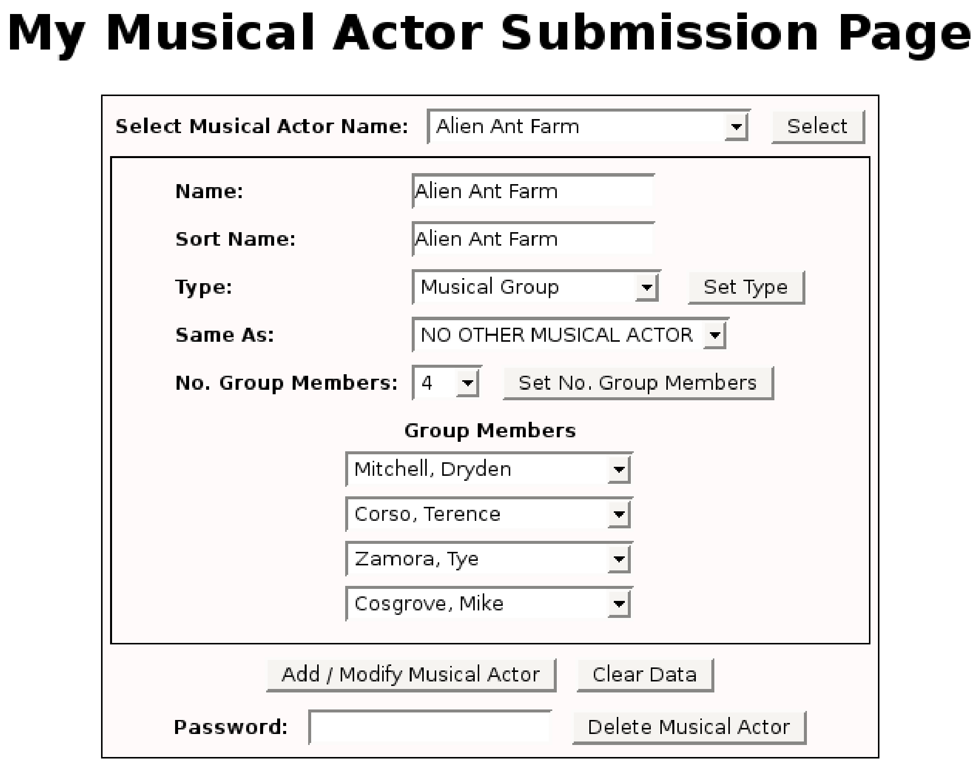
```
~       Q:whn:N track
track   N:det:Det       how many
how many        Det:lex-mod:U   many
~       Q:head:YNQ      ~
~       YNQ:inv-aux:Aux do
~       YNQ:head:V      have
have    V:s:N   x
x       N:nn:N  album
have    V:obj:N track
have    V:subj:N        x
have    V:mod:Prep      on
on      Prep:pcomp-n:N  it
```

Stanford Parser Output:

```
(ROOT
  (SBARQ
    (WHNP
      (WHADJP (WRB How) (JJ many))
      (NNS tracks))
    (SQ (VBZ does)
      (NP (NN album))
      (ADVP (NNP X))
      (VP (VB have)
        (PP (IN on)
          (NP (PRP it)))))
    (. ?)))
```

FIGURE B.1: Parser Interface for MINIPAR and Stanford Parsers

B.2 Screenshots of Musical Data Submission System



My Musical Actor Submission Page

Select Musical Actor Name:	Alien Ant Farm	Select
Name:	Alien Ant Farm	
Sort Name:	Alien Ant Farm	
Type:	Musical Group	Set Type
Same As:	NO OTHER MUSICAL ACTOR	
No. Group Members:	4	Set No. Group Members

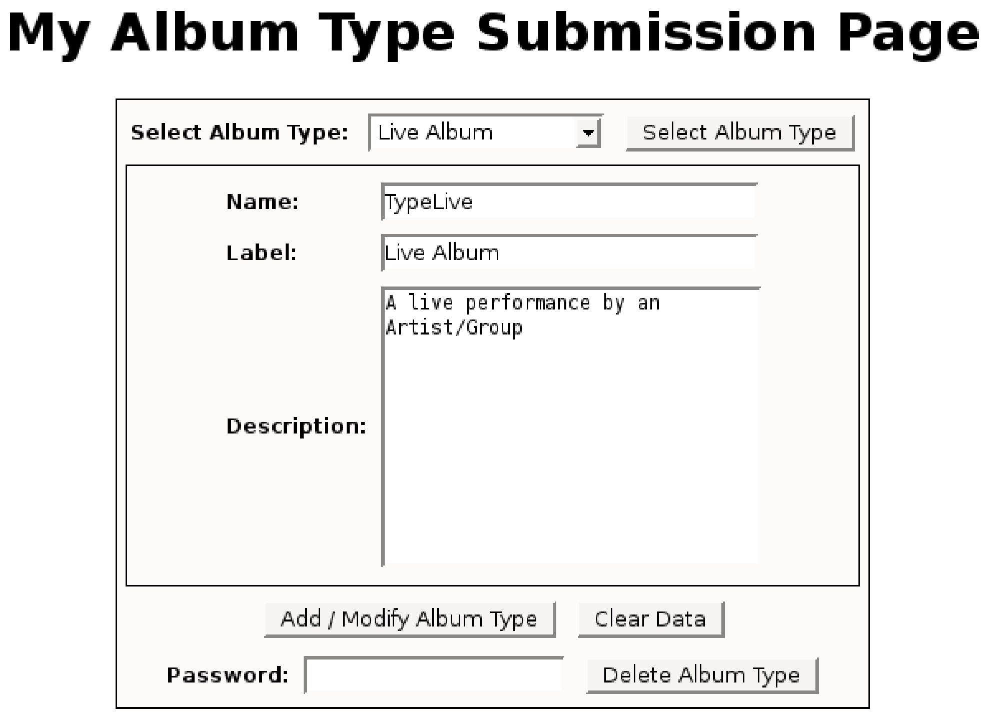
Group Members

- Mitchell, Dryden
- Corso, Terence
- Zamora, Tye
- Cosgrove, Mike

Add / Modify Musical Actor | Clear Data

Password: | Delete Musical Actor

FIGURE B.2: Musical Actor Submission Form



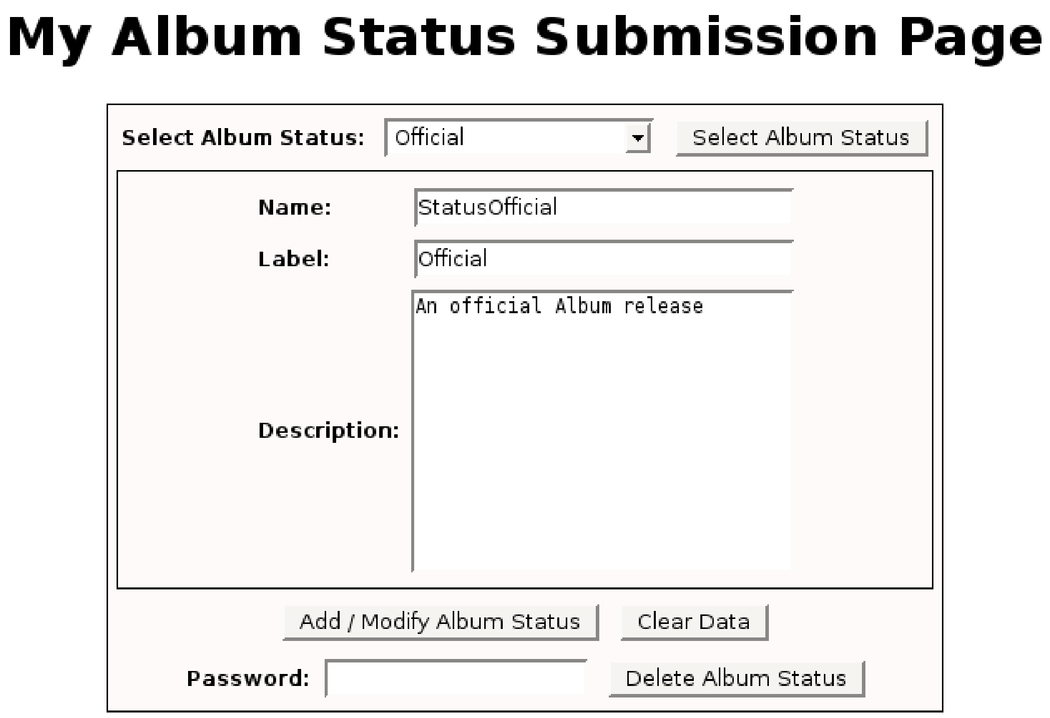
My Album Type Submission Page

Select Album Type:	Live Album	Select Album Type
Name:	TypeLive	
Label:	Live Album	
Description:	A live performance by an Artist/Group	

Add / Modify Album Type | Clear Data

Password: | Delete Album Type

FIGURE B.3: Album Type Submission Form



My Album Status Submission Page

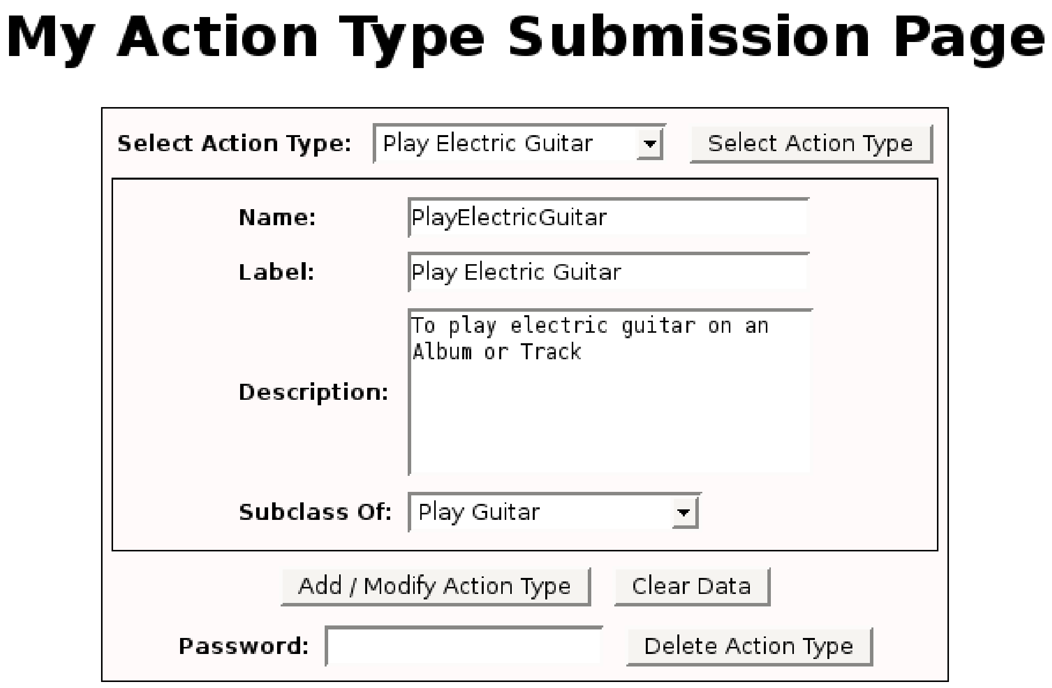
Select Album Status: Official [Select Album Status]

Name:	StatusOfficial
Label:	Official
Description:	An official Album release

[Add / Modify Album Status] [Clear Data]

Password: [Delete Album Status]

Figure B.4: Album Status Submission Form



My Action Type Submission Page

Select Action Type: Play Electric Guitar [Select Action Type]

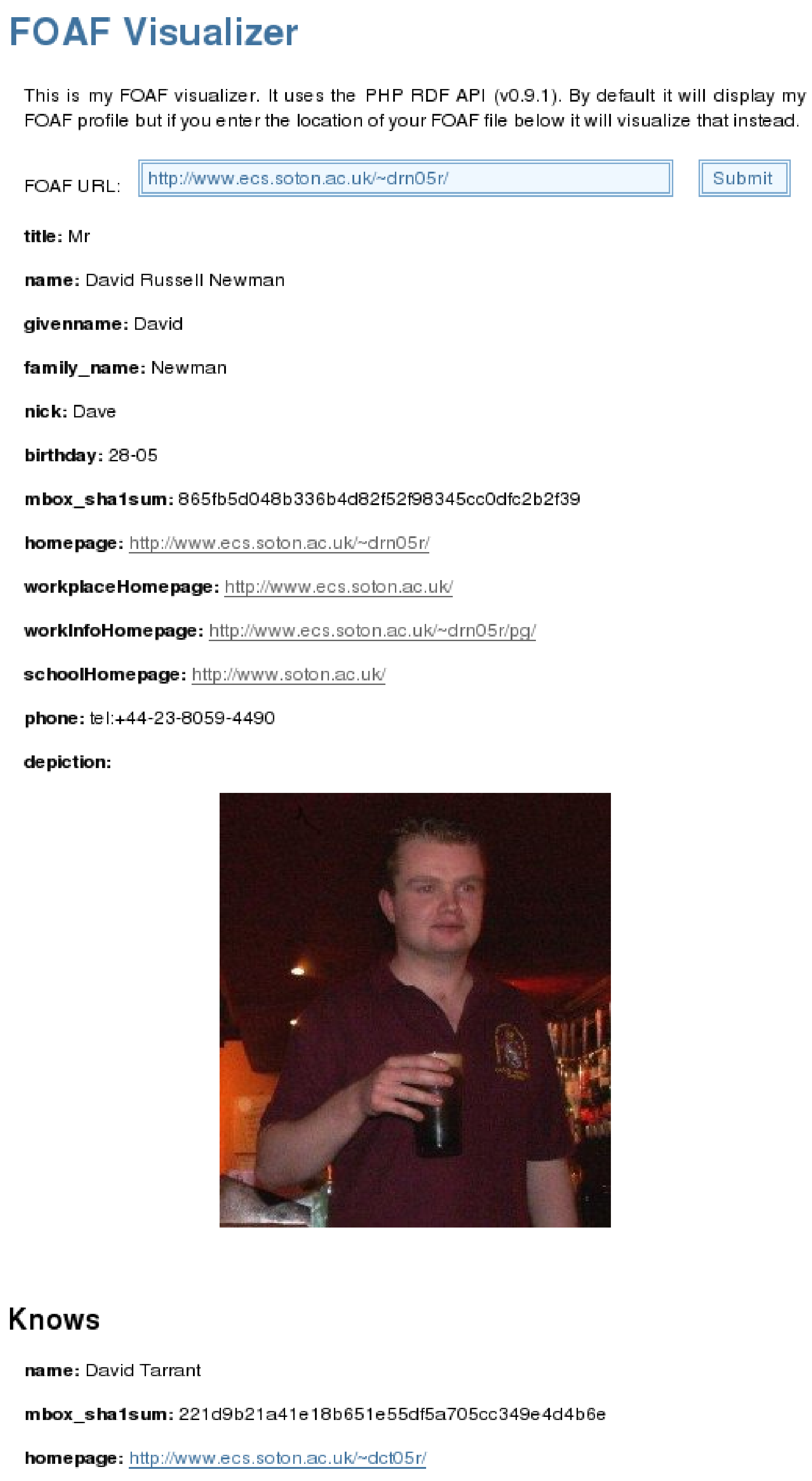
Name:	PlayElectricGuitar
Label:	Play Electric Guitar
Description:	To play electric guitar on an Album or Track
Subclass Of:	Play Guitar

[Add / Modify Action Type] [Clear Data]

Password: [Delete Action Type]

Figure B.5: Musical Action Submission Form

B.3 Screenshots of Thesaurus Management System



FOAF Visualizer

This is my FOAF visualizer. It uses the PHP RDF API (v0.9.1). By default it will display my FOAF profile but if you enter the location of your FOAF file below it will visualize that instead.

FOAF URL: http://www.ecs.soton.ac.uk/~drn05r/ Submit

title: Mr

name: David Russell Newman

givenname: David

family_name: Newman

nick: Dave

birthday: 28-05

mbox_sha1sum: 865fb5d048b336b4d82f52f98345cc0dfc2b2f39

homepage: http://www.ecs.soton.ac.uk/~drn05r/

workplaceHomepage: http://www.ecs.soton.ac.uk/

workInfoHomepage: http://www.ecs.soton.ac.uk/~drn05r/pg/

schoolHomepage: http://www.soton.ac.uk/

phone: tel:+44-23-8059-4490

depiction:

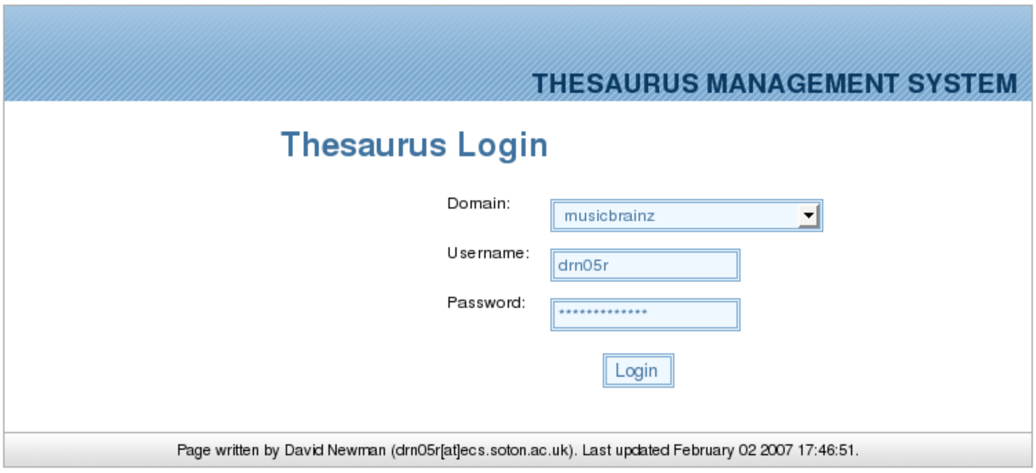
Knows

name: David Tarrant

mbox_sha1sum: 221d9b21a41e18b651e55df5a705cc349e4d4b6e

homepage: http://www.ecs.soton.ac.uk/~dct05r/

FIGURE B.6: FOAF Profile Visualizer

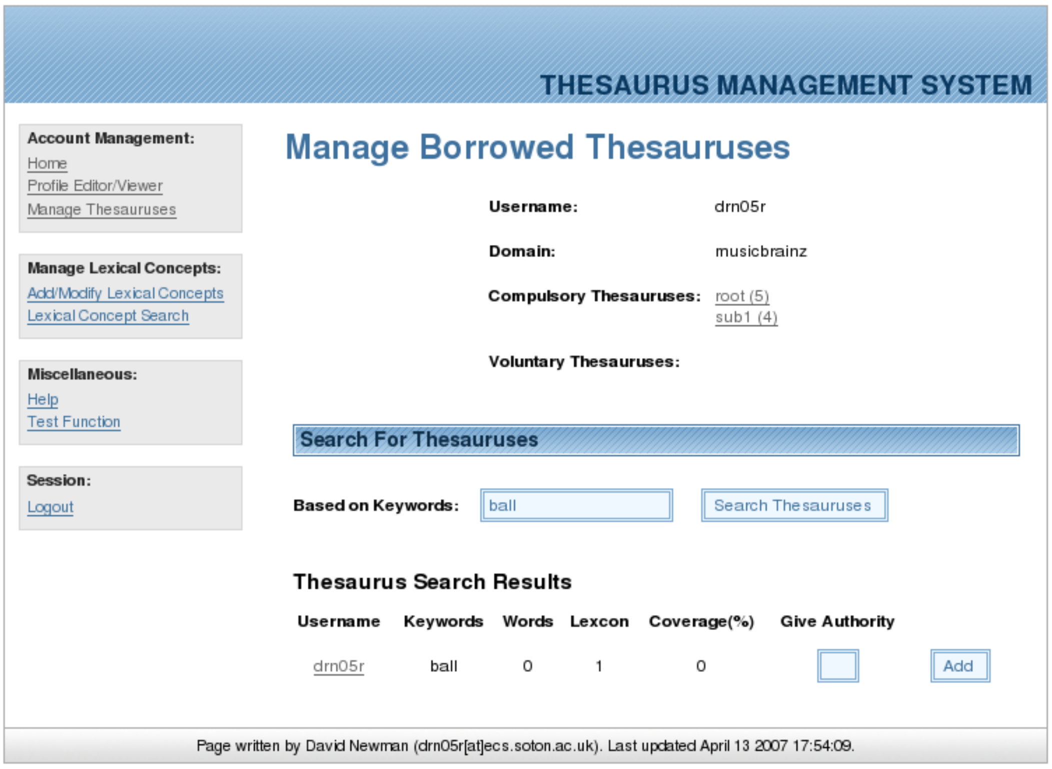


Thesaurus Login

Domain:	musicbrainz
Username:	drn05r
Password:	*************

Page written by David Newman (drn05r[at]ecs.soton.ac.uk). Last updated February 02 2007 17:46:51.

FIGURE B.7: Login Form for Thesaurus Management System



Manage Borrowed Thesauruses

Username:	drn05r
Domain:	musicbrainz
Compulsory Thesauruses:	root (5) sub1 (4)
Voluntary Thesauruses:	

Search For Thesauruses

Based on Keywords: ball

Thesaurus Search Results

Username	Keywords	Words	Lexcon	Coverage(%)	Give Authority
drn05r	ball	0	1	0	

Page written by David Newman (drn05r[at]ecs.soton.ac.uk). Last updated April 13 2007 17:54:09.

FIGURE B.8: Manage Thesauruses for for Thesaurus Management System

Bibliography

[Allen 87] J. Allen. Natural lanaguage understanding. Apt, A. and Weisel, J., 1987.

[Alves 06] A. Alves, A. Arkin, S. Askay, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, V. Knig D. amd Mehta, S. Thatte, D. van der Rijn, P. Yendluri & A Yiu. *Web Services Business Process Execution Language Version 2.0.* http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm, May 2006. OASIS Committee Draft.

[Androutsopoulos 95] I. Androutsopoulos, G. D. Ritchie & P. Thanisch. *Natural Language Interfaces to Databases–an introduction.* Journal of Language Engineering, vol. 1, no. 1, pages 29–81, 1995.

[Bechhofer 04] S. Bechhofer & R. Volz. *Patching Syntax in OWL Ontologies.* In Proceedings of ISWC2004, 2004.

[Beckett 01] D. Beckett. *The Design and Implementation of the Redland RDF Application Frameworke.* In WWW10, 2001.

[Bellinger 04] G. Bellinger, D. Castro & A. Mills. *Data, Information, Knowledge, and Wisdom.* web: http://www.systems-thinking.org/dikw/dikw.htm, 2004.

[Berners-Lee 02] T. Berners-Lee. *The Semantic Web.* W3C Presentation, April 2002.

[Bernstein 06] A. Bernstein, E. Kaufmann, C. Kaiser & C. Kiefer. *Ginseng: A Guided Input Natural Language Search Engine for Querying Ontologies.* In Proceedings of the 2006 Jena User Conference, May 2006.

[Bray 04] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler & F. Yergeau. *Extensible Markup Language (XML) 1.0.* W3C, third edition edition, February 2004. W3C Recommendation.

[Brickley 04] D. Brickley & R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema.* http://www.w3.org/TR/rdf-schema/, February 2004. W3C Recommendation.

[Broekstra 02] J. Broekstra & .and van Harmelen F. Kampman A. *Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema.* In ISWC2002, 2002.

[Brown 96] R. D. Brown, J. E. Boggs, R. Hilderbrandt, K. Lim, I. M. Mills, E. Nikitin & M. H. Palmer. *Acronyms Used in Theoretical Chemistry.* Pure Appl. Chem., vol. 68, no. 2, pages 387–456, 1996.

[Carroll 04] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne & K. Wilkinson. *Jena: Implementing the Semantic Web Recommendations.* In WWW2004, 2004.

[Cohen 92] P. R. Cohen. *The Role of Natural Language in a Multimodal Interface.* In Proceedings of the 5th annual ACM symposium on User interface software and technology, pages 143–149, New York, NY, USA, 1992. ACM Press.

[Cunningham 06] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani & I. Roberts. *Developing Language Processing Components with GATE Version 4 (a User Guide).* University of Sheffield, October 2006.

[De Roure 07] D. De Roure & C. Goble. *myExperiment 2̆013 A Web 2.0 Virtual Research Environment.* In Submitted to International Workshop on Virtual Research Environments and Collaborative Work Environments, Edinburgh, UK, 2007.

[Dix 03] A. Dix. Funology: From usability to enjoyment, chapitre 13: Deconstructing Experience - pulling crackers apart, pages 165–178. Kluwer Academic Publishers, Dordrecht, 2003.

[Dowty 81] D. R. Dowty, R. E. Wall & S. Peters. Introduction to montague semantics. Kluwer Academic Publishers, 1981.

[Fellbaum 98] C. Fellbaum, G. A. Miller, R. Al-Halimi, R. C. Berwick, J. F. M. Burg, M . Chodorow, J. Grabowski, S. Harabagiu, M. A. Hearst, G. Hirst, D. A. Jones, R. Kazman, K. T. Kohl, S. Landes, C. Leacock, K. J. Miller, D. Moldovan, N. Nomura, U. Priss, P. Resnik, D. St-Onge, R. Tengi, R. P. van de Riet & E. Voorhees. Wordnet: An electronic lexical database. MIT Press, 1998.

[Gibson 05] A. Gibson, R. Stevens, R. Cooke, S. Brostoff & m. c. schraefel. *myTea: Connecting the Web to Digital Science on the Desktop.* Rapport technique, ECS, University of Southampton, October 2005.

[Guha 97] R. V. Guha & T. Bray. *Meta Content Framework Using XML.* W3CNote, June 1997.

[Harris 03] S. Harris & N. Gibbins. *3store: Efficient Bulk RDF Storage.* In 1st International Workshop on Practical and Scalable Semantic Systems, pages 1–15, Sanibel Island, Florida, 2003.

[Hendrix 78] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz & J. Slocum. *Developing a Natural Language Interface to Complex Data.* ACM Transactions on Database Systems, vol. 3, no. 2, pages 105–147, June 1978.

[Hinrichs 88] E. W. Hinrichs. *Tense, Quantifiers, and Contexts.* Computational Linguistics,, vol. 14, no. 2, June 1988.

[Horrocks 02] I. Horrocks & P. F. Patel-Schneider. *Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web.* http://www.cs.man.ac.uk/ horrocks/Publications/download/2002/AAAI02IHorrocks.pdf, 2002.

[Horrocks 03] I. Horrocks & P. F. Patel-Schneider. *Reducing OWL Entailment to Description Logic Satisfiability.* In Proc. of the 2nd International Semantic Web Conference (ISWC), 2003.

[Hughes 04] G. Hughes, H. Mills, D. De Roure, J. G. Frey, L. Moreau, m. c. schraefel, G. Smith & E. Zaluska. *The Semantic Smart Laboratory: A system for supporting the chemical eScientist.* Org. Biomol. Chem., vol. 2, pages 1–10, 2004. DOI: 10.1039/b410075a.

[Johnson 85] T. Johnson. Natural language computing: The commercial applications. Ovum Ltd., 1985.

[Karvounarakis 02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis & M Scholl. *RQL: A Declarative Query Language for RDF.* In WWW2002, May 2002.

[Klein 03] D. Klein & C. D. Manning. *Accurate Unlexicalized Parsing.* In 41st Meeting of the Association for Computational Linguistics, pages 423–430, 2003.

[Lassila 98] O. Lassila. *Web Metadata: A Matter of Semantics.* IEEE Internet Computing, vol. 2, no. 4, pages 30–37, July/August 1998.

[Lassila 99] O. Lassila & R. R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification.* Rapport technique, W3C, February 1999. W3C Recommendation.

[Lee 04] R. Lee. *Scalability Report on Triple Store Applications.* http://simile.mit.edu/reports/stores/stores.pdf, July 2004. Simile Project.

[Lin 94] D. Lin. *PRINCIPAR-An Efficient, Broad-coverage, Principle-based Parser.* In COLING-94, pages 42–48, Kyoto, Japan, 1994.

[Lin 98] D. Lin. *Dependency-based Evaluation of MINIPAR.* In Workshop on the Evaluation of Parsing Systems, May 1998.

[Manola 04] F. Manola, E. Miller & B. McBride. *RDF Primer.* http://www.w3.org/TR/rdf-primer/, February 2004. W3C Recommendation.

[Martin 04] D. Martin, A. Ankolekar, M. Burstein, G. Denker, D. Elenius, J. Hobbs, L Kagal, O. Lassila, A. McDermott, D. McGuinness, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, T. Sabou, C. Schlenoff, E. Sirin, M. Solanki, N. Srinivasan, K. Sycara & R. Washington. *OWL-S: Semantic Markup for Web Services.* http://www.daml.org/services/owl-s/1.1/overview/, November 2004. DAML Technical Overview.

[McConnel 95] S. McConnel. *PC-PATR Reference Manual.* Sil International, 7500 W. Camp Wisdom Road Dallas TX 75236-5629, USA, October 1995.

[Minksy 74] M. Minksy. *A Framework for Representing Knowledge.* Rapport technique, Massachusetts Institute of Technology, Cambridge, MA, USA 1974.

[MyS 07] MySQL AB. *MySQL 5.1 Reference Manual*, 4959 edition, February 2007.

[Naber 04] D. Naber. *OpenThesaurus: Building a Thesaurus with a Web Communitye.* http://www.openthesaurus.de/download/openthesaurus.pdf, June 2004.

[Newman 06] D. Newman. How semantic web knowledge technologies might improve natural language querying systems. 9 Month Mphil/PhD Progress Report, July 2006.

[Prud'hommeaux 06] E. Prud'hommeaux & A Seaborne. *SPARQL Query Language for RDF.* http://www.w3.org/TR/rdf-sparql-query/, April 2006. W3C Candidate Recommendation.

[schraefel 04a] m. c. schraefel, G. Hughes, H. Mills, G. Smith & J. Frey. *Making Tea: Iterative Design through Analogy. In Proceedings of Designing Interactive Systems.* In 2004 conference on Designing interactive systems: processes, practices, methods, and techniques, pages 49–58, Cambridge Mass, USA, 2004. ACM Press.

[schraefel 04b] m. c. schraefel, G. Hughes, H. Mills, G. Smith, T. Payne & J. Frey. *Breaking the Book: Translating the Chemistry Lab Book into a Pervasive Computing Lab Environment.* In Conference on Human Factors in Computing Systems (CHI), pages 25–32, Vienna, Austria, 2004. ACM Press.

[Seaborne 04] A Seaborne. *RDQL - A Query Language for RDF.* http://www.w3.org/Submission/RDQL/, January 2004. W3C Member Submission.

[Shneiderman 80] B. Shneiderman. *Natural vs. precise language for human operation of computers.* In ACL Proceedings, 18th Annual Meeting, pages 139–141, 1980.

[Sleator 03] D. D. Sleator & D. Temperley. *Parsing English with a link grammar.* In Third International Workshop on Parsing Technologies, 2003.

[Spenceley 92] S. E. Spenceley. *The theory of principal-based engineering.* IEE Colloquium on Principle Based Engineering, no. 1, pages 1–3, 1992.

[Thompson 83] B. H. Thompson & F. B. Thompson. *Introducing ASK A Simple Knowledgeable System.* In Proceedings of the 1st Conference on Applied Natural Language Processing, pages 17–24, Santa Monica, California, 1983. California Institute of Technology.

[Torisawa 00] K. Torisawa, K. Nishida, Y. Miyao & J. Tsujii. *An HPSG parser with CFG filtering.* Natural Language Engineering, vol. 6, no. 1, pages 63–80, 2000.

[tri 05] *Tripletest Report.* http://tripletest.sourceforge.net/2005-06-08/index.html, June 2005.

[van Assem 06] M. van Assem, A. Gangemi & G. Schreiber. *Developing Language Processing Components with GATE Version 4 (a User Guide).* Working draft, W3C, June 2006.

[Vogel 90] C. M. Vogel. Inheritance reasoning and head-driven phrase structure grammard. Master's thesis, Computing Science at Simon Fraser University, December 1990.

[Waltz 78] D. L. Waltz. *An English Language System for a Large Relational Database.* Communications of teh ACM, vol. 21, no. 7, pages 526–539, July 1978.

[Warren 82] D. H. D. Warren & F. C. N. Pereira. *An Efficient Easily Adaptable System for Interpreting Natural Language Queries.* American Journal of Computational Linguistics, vol. 8, no. 3-4, pages 110–122, 1982.

[Wood 05] D. Wood, P. Gearon & T. Adams. *Kowari: A Platform for Semantic Web Storage and Analysist.* In WWW2005, 2005.

[Woods 72] W. A. Woods, R. M. Kaplan & B. N. Webber. The lunar sciences natural language information system: Final report. Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.