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

A progress report submitted for continuation towards a PhD

Supervisor: Prof. David De Roure
Examiner: TBC

How Semantic Web Knowledge Technologies Might Improve Natural Language Querying Systems

by David R. Newman

July 19, 2006

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

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

A progress report submitted for continuation towards a PhD

by David R. Newman

Natural Language (NL) querying has an extensive history, which this report reviews to determine what constitutes a NL querying system. It then examines Semantic Web (SW) knowledge technologies and considers how these may be used to improve an NL querying system. This report looks at the Comb*e*Chem project for an example of how these knowledge technologies can be used. Then it describes the partial implementation of a NL querying system using SW knowledge technologies. The report concludes by considering some applications that may benefit from this type of of NL querying system and how, through the use of ontologies, some of the inherent flaws in existing NL querying systems can be resolved. The report finishes my laying out a plan for future work.

Contents

List of Figures

Chapter 1

Introduction

The main purpose of this report is to consider whether Semantic Web (SW) knowledge technologies would be suitable in helping construct a back-end for a Natural Language (NL) querying system. Further, whether these technologies would improve on existing NL querying systems and be able to solve some of the inherent problems with them.

In Chapter 2 this report reviews some of NL querying's history, before considering a scheme for how to implement such a system, based on how previous systems have been designed. Creating any new system is only worthwhile if it has distinguishing features or uses new technologies.

The semantic web uses knowledge technologies such as Resource Description Framework[1] (RDF) (Manola 04), the Web Ontology Language[2] (OWL) (Bechhofer 04a) and triplestores that are databases designed to store RDF data. This report examines these in Chapter 3 and analyses how these could be used as a knowledge base, which is a crucial part of a NL querying system. Chapter 4 looks at the Comb*e*Chem project[3], which uses these SW knowledge technologies, for its digital lab book's database (Hughes 04).

From the understanding gained in the literature review of Chapters 2-4, a test implementation has been proposed and partially completed for an NL querying system using an ontology similar to the MusicBrainz[4] ontology.

This report concludes by distinguishing how a NL querying system that uses SW knowledge technologies is different from older NL querying systems. It also proposes potential new applications for the system, that take advantage of its web orientation and the potential for integration within a pervasive system. The conclusion also considers how the implementation proposed in Chapter 5 would have to be modified for a new domain. It then goes on to conclude how SW knowledge technologies can solve some of the problems

[1]http://www.w3.org/TR/rdf-primer/
[2]http://www.w3.org/TR/owl-ref
[3]http://www.combechem.org
[4]http://www.musicbrainz.org

with NL querying systems. Finally the report lays out a plan of future work up to the completion of the mini-thesis report. It also briefly considers some future work that may be undertaken after the mini-thesis report.

Chapter 2

Natural Language (NL) Querying

The main goal of knowledge representation in computer science is to take human concepts and define them in a format that can be read by a machine. Achieving this is only half the battle, there must be a way for the human user to interact with this represented knowledge. These interactions come in two forms:

1. Inputs: Human-Readable (H-R) statements in the form of questions or commands.[1]
2. Outputs: H-R statements in the form of answers.

NL querying must address both of these interactions; however good a system is at representing a H-R query in a Machine-Readable (M-R) format is useless if there is not a mirrored system that can represent a M-R output in a H-R format.

2.1 The Origins of NL Querying

The principles of NL querying are not new. The concept of NL 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 Structured Query Language (SQL) as the M-R format to query their database. Some more novel alternatives compared to relational databases with SQL have also been implemented (Androutsopoulos 95).

CHAT-80 uses a database of Prolog statements to store logical relationships between entities such as the fact, "France exports wine to Britain", as exports(france,wine,britain)

[1]A question is a sentence that requires a question mark at the end, e.g. "How many people are there in London?" An equivalent command might be "Count all the people in London." For simplicity, when the term query is used, this means either a question or a command

(Warren 82). The ability to store relationships with this logical format is important in increasing the potential for representing human concepts in a M-R format, which is an essential element required for any NL querying system to be successful.

2.2 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 could be a form-based querying system and although it makes the generation of M-R queries trivial, it is liable to limit the number of queries that can be performed and therefore make the data in the database of a lot less useful than it could potentially be. There are also some queries 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:

1. The query submitted has either grammatical or spelling defects, so it meaning could not be understood.
2. The grammar and spelling of the query is correct but the NL Parser's (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 M-R query.
4. No appropriate data has been recorded in the database for the query to find[2].

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

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

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 another NLIDB 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 query so it can try and work out what the current query 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/query to be “Who recorded the album London Calling”

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). 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, if not most cases, be the easiest way to express a 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.

2.3 The Data Flow of a NL Querying System

There are many steps between the input of a NL query and the output of a H-R answer, (see Figure 2.1). Careful consideration is needed at each step to ensure seamless data flow through the system.

2.3.1 Natural Language Parsers (NLPs)

There is a reasonably large number of NLPs to be found on the web. Many are designed by specialist university research groups such as the Stanford NLP group[3] and the

[3]http://www-nlp.stanford.edu/

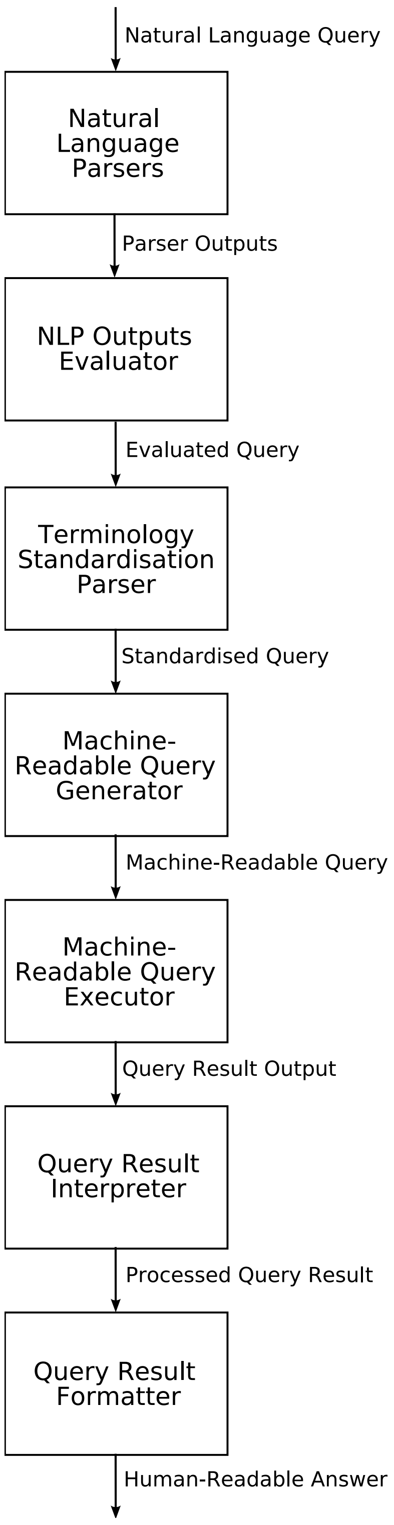


FIGURE 2.1: The Flow from NL Query to H-R Answer

Sheffield NLP group[4]. One of the main attributes of an NLP is the type of grammar it uses. There are quite a number of different types of grammar used in NLPs, it is difficult to be certain of the exact number, as some grammars are sometimes considered to be specialisations of other grammars. The following four types of grammars are quite common and have been used by NLP implementations:

1. Rule-based
2. Principal-based
3. Unification-based
4. Head-driven Phrase Structure (HPS)

The main difference between rule-based and principal-based grammars is that rules "specify how to precisely generate sentences" (Spenceley 92) whereas principals use constraints, "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 principal-based grammars have the ability to generate parse trees proving a sentence is valid. Principal-based grammars have the advantage of also being able to generate proofs for why a sentences is not valid.

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 must 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})
```

[4]http://nlp.shef.ac.uk/

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 really rule-based grammars with greater formalism, therefore comparison with principle-based grammars is very similar to comparing rule-based grammars with principal-based grammars.

HPS grammars are based on Generalised Phrase Structure (GPS) grammars but provide a much simpler Context Free Grammar (CFG) at the expense of a much more complex lexicon, i.e. a structure that stores information about words in the grammar. This is achieved by using greater subcategorization on the heads of phrases. What makes HPS grammars quite interesting is they basically use frame-based representation for knowledge (Vogel 90), which has similarities to SW knowledge technologies (see Chapter 3).

The Stanford parser[5] (Klein 03) and link grammar parser[6] (Sleator 03) are good example of NLPs with rule-based grammars. Principal-based grammar NLPs are slightly less common, principar(Lin 94) and minipar[7] (Lin 98), which is a smaller version of principar are quite well known. PC-PATR[8] (McConnel 95) is one of the most well known unification-based grammar NLPs. The TNT parser (Torisawa 00) is an example of a NLP that uses an HPS grammar.

Any NLP, whatever type of grammar it uses will have to handle the problem of a sentence having two or more potential structures. The simplest way of dealing with this is to return all possible structures but in general a user will want an NLP package that at least returns the most probable structure, if not returns a weighted list of all possible structures. Various statistical and stochastic methods have been applied to this problem, some NLP packages even use evolving machine-learning techniques to better tackle this problem.

2.3.2 NLP Output Evaluation

Output from a NLP will give syntactic and possibly even semantic structure to the sentences parsed to it but this data is still a long way off being M-R query, particular if output is taken from two or more NLPs. Taking output from more than one NLP may

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

at first appear illogical, as it adds an extra layer of complexity, particularly if two NLPs contradict one another. Contradictions are however a good thing, as something missed by one NLP may be picked up by the other. To make the extra complexity viable the use of multiple NLPs must be justified by demonstating that the NLPs approaches are different. (E.g. One uses a rule-based grammar and the other uses a principal-based grammar).

One of the main purposes of the NLP output evaluation" phase is to determine which NLP has found the correct structure for a particular sentence, this task may not be straightforward, particularly if there is a high variation in sentence structure. However if the sentences are of a regular structure, e.g. of a simple query-like structure, this task becomes easier.

The other purpose of this phase is to process the NLP output and put it into an appropriate data structure that will allow the next phase, i.e. the terminology standardisation, to be performed.

2.3.3 Terminology Standardisation

The greatest inherent problem with NL querying is that one person will phrase a query differently to another. Phrasing a query differently means that either the query's syntactic structure is different or the actual words used within the query are different, more often than not both of these are true. The former should have been resolved by the NLP(s) but they may not be able to resolve the latter.

It is not uncommon for NLPs to have the additional feature of a thesaurus and this can be used to determine whether two statements are the same e.g.

1. There was a great conflagration.
2. There was a great fire.

An NLP with a built-in thesaurus would be able to resolve that the two sentences were stating the same thing. This is however a very simple case, which uses two synonyms that are not overloaded terms. In general built-in thesauruses or even stand-alone thesauruses in post-processing are likely to struggle with more difficult cases, even constraining the domain in which the thesaurus has to operate will not necessarily alleviate the problem. E.g. Consider a domain specifically for the expression of musical albums and associated entities. Two NL queries in that domain might be:

1. Who recorded the album Help?
2. Who was the creator of the album Help?

To a human these two queries will most likely mean the same thing, if an NLP can resolve that the queries have the same semantic meaning, i.e. some unknown entity has performed an action on the album Help!, a general-purpose thesaurus will not resolve that the verb roots ‘record’ and ‘create’ are synonymous. This is because in specific domains, terms can come to mean the same thing because the more generalised term cannot mean anything other than the more specific. I.e. the creator of an album is the same as the recording artist of that album.

To make a it possible to generate a M-R query the terminology of the NL query must be standardised. The output from this step should be identical or near identical for two NL queries that require the same answer. To achieve this for a specific domain requires a thesaurus to be created explicitly for the domain.

How to generate a domain specific thesaurus is a difficult question. To make it comprehensive, synonymous terms must be defined by the users but this could lead to potential conflicts. Any initial thesaurus would have to be defined by the designer(s) of the NL querying system for that domain, probably with the input of end users. This initial thesaurus would be far from complete and therefore it would need to be able to evolve after the system goes live. Evolution would need to be managed to prevent conflicts arising. The main reason that conflicts may arise is due to different interpretations by different users. A hierarchical approach, (e.g. senior users can edit the root thesaurus, less senior users can edit sub thesauruses and some users can only edit their personal thesauruses), may help reduce conflicts but would also slow the evolution of the thesaurus.

2.3.4 Machine-Readable (M-R) Query Generation

Most well-developed databases have a highly-structured language(s) for querying the data they store. The most common example of this is SQL for relational databases. The nature of these querying languages means that automated generation of statements should be reasonably straightforward, as long as the data used for generating these statements is well-structured[9]. Assuming the phases defined in sections 2.3.1 to 2.3.3 have found no ambiguity this well-structured data should be available.

2.3.5 Machine-Readable (M-R) Query Execution

Executing a M-R query should be simple, as the database and query language being used should have been decided at the system’s inception. All that is required is to pass the M-R query generated in section 2.3.4 to the chosen database’s query engine.

[9]Data with interpretation, which can be provided by structure, is generally considered to be Information and sometimes even Knowledge, (see section 3.3).

2.3.6 Query Result Interpretation

A M-R query engine, will return raw data with limited structure, e.g. a table. There is a high probability that data will not explicitly answer the original query posed. E.g. the query may have been "How many albums did the Rolling Stones record?", some query languages allow you to perform count commands, such as SQL but some such as SPARQL Protocol And RDF Query Language[10] (SPARQL) (Prud'hommeaux 06) do not, therefore to interpret the result the number of records returned needs to be counted. Counting is the most basic of arithmetic operations but there are many more complex arithmetic and logical operations that may need to be performed on a query result to properly interpret the data so that an appropriate answer can be returned.

2.3.7 Query Result Formatting

Even if all the appropriate data can be disseminated from a database to answer a NL query, this is a pointless exercise if the data cannot be displayed in a useful H-R way. H-R does not necessary mean a perfectly structured NL sentence or even natural language at all, it is a question of whether data is displayed in the way a user would expect, based on their original query. E.g. A user asks how did the Premiership table finish at the end of the 2005/06 season. Returning a NL answer saying "Chelsea finished first, eight points clear of Manchester United in second..." would be far less useful than just returning the complete Premiership table.

More often than not good query result formatting is just a question of style and layout, setting data out logically and highlighting where appropriate. Query result formatting can sometimes be a subjective task, if user trials were carried out to see how many users get appropriate answers to their NL queries, the time taken to dissemenate this information from the displayed result would be a test of how effective the query result formatter had been.

2.4 The Future for NL Querying

As determined in section 2.2, there are many disadvantages with using a NL querying system over other form-based or GUI systems. SW knowledge technologies that are starting to be used on the web may provide solutions to some of these disadvantages. The need for simple yet flexible systems has become more important since the advent of the web with all the information it contains and this may herald a golden age for NL querying systems.

[10]http://www.w3.org/TR/rdf-sparql-query

Chapter 3

Semantic Web Knowledge 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 Markup Language[1] (XML) (Bray 04). In combination with Minsky (Minksy 74) and others frame-based representation systems, MCF inspired the development of Resource Description Framework[2] (RDF) (Manola 04) (Lassila 98).

The new generation of the Web aims increase the amount of M-R 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.

3.1 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[3]. 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 M-R 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.

[1]http://www.w3.org/TR/REC-xml/
[2]http://www.w3.org/TR/rdf-primer/
[3]RDF can also be represented in N3, NTriple and TURTLE.

RDF Schema[4] (RDFS)(Brickley 04) allows RDF data to be grouped into a class/property structure, somewhat like an Object-Oriented (OO) programming language[5], 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 hierarchy 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[6] (DAML) in the US and Ontology Interchange Language[7] (OIL) in Europe, these two projects eventually merge to form the Web Ontology Language[8] (OWL) (Bechhofer 04a) (Horrocks 02)

3.2 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 are required, only these are used. For this reason there are three main species of OWL[9], 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 04b)

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.

[4]http://www.w3.org/TR/rdf-schema/

[5]RDFS is still quite different to an object-oriented programming language. For one, RDFS has a more property-oriented view whereas OO programming languages have a more class-oriented view.

[6]http://www.daml.org/

[7]http://www.ontoknowledge.org/oil/

[8]http://www.w3.org/TR/owl-ref/

[9]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/)

3.3 Data, Information and Knowledge

As already stated one of the main purposes of knowledge technologies is to take human concepts and convert it to M-R 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 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 3.1 by (Bellinger 04) shows

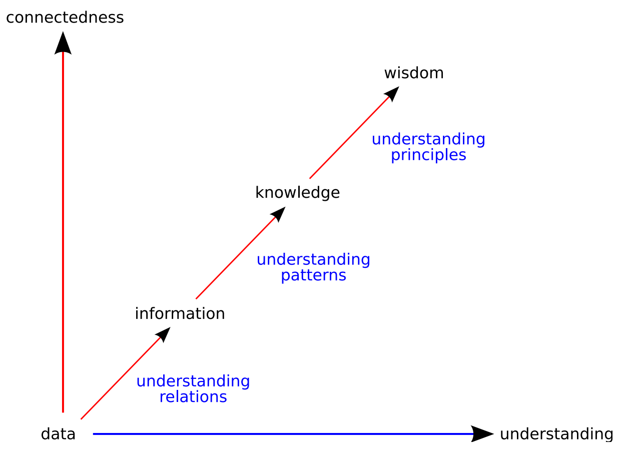


FIGURE 3.1: Data, Information, Knowledge and Wisdom

how both relations and patterns must be understood before data can be transformed into knowledge. What is meant when someone says, "The computer understands..." is subjective. If the person is making reference to understanding like a human understands that a tomato is a fruit, then a computer does not even understand that $1 + 1 = 2$. The only reason that a computer can calculate $1 + 1 = 2$ is because it is built into its hardware that it can only return that result, it has no understanding of the concept of 1, 2 or why $1 + 1 = 2$. 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.

3.3.1 How SW Knowledge Technologies turn 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 4. (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.

3.3.2 How SW Knowledge Technologies turn 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 3.1, 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 a ontology designer will enforce some sort of hierarchy, as without this structure the data within the triplestore would become almost impossible to manage.

RDF 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[10] 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 3.2). 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.

3.4 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

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

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 an ontology can often make it less flexible. It also may mean that less people are willing use your ontology because of these contentious 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, which is not always the case in relational databases as the data may not be compatible with the enforced structure of the tables.

3.4.1 Querying a Triplestore

The whole purpose of having a triplestore is to be able to query it. SPARQL Protocol And RDF Query Language[11] (SPARQL) (Prud'hommeaux 06) is designed specifically for querying RDF triplestores. SPARQL uses a syntax similar to SQL, which is intuitive considering many 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:

```
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

[11]http://www.w3.org/TR/rdf-sparql-query/

(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.

Chapter 4

Comb*e*Chem: An Example Triplestore Implementation

4.1 Background

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. 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[1].

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.
- Minimising 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.

4.1.1 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 SmartTea project was developed.

SmartTea[2] 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 chemical 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

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

[2]http://www.smarttea.org

(Dix 03), this advocates deconstructing a process to a point where its exact purposes can be understood, (i.e. from a complex chemical 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.

4.2 The Comb*e*Chem Triplestore

The Comb*e*Chem triplestore is the database described in section 4.1. The Comb*e*Chem Triplestore uses JenaRDF[3] implementation to store its data. The Comb*e*Chem triplestore currently uses an specifically designed RDFS ontology but there are plans to use an OWL ontology in the future (Hughes 04).

The current ontology 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 ontology 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 ontology, as the experiments that the system would be used for were very unlikely to be of this nature. Learning lessons from Business Process Execution Language (BPEL) (Alves 06) and OWL-S (Martin 04) may make it possible to design an ontology that is capable of generating graphs like previously described (Hughes 04).

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

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.

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 ontology was designed this inference is a somewhat limited, mainly to RDFS's 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.

Chapter 5

Implementation

As discussed in Chapter 2, producing an effective NL querying system is far from trivial, if possible at all. The only real way to determine the feasibility of designing a new NL querying system using SW knowledge technologies, is through an example implementation. Any implementation would require a strict set of constraints, to make the design of any system achievable in reasonable time. One of the most important constraints would be to focus on only one domain.

A further consideration, is the choice of the database back-end. As the purpose of an implementation is to take advantage of the SW knowledge technologies, this database back-end would therefore inevitably be an RDF triplestore. Consideration must then be given to ensure a suitable triplestore application is chosen.

5.1 Choosing the Domain

Choosing one domain from the almost unlimited list is not straightforward therefore some criteria most be followed to determine whether an appropriate domain has been chosen.

- A high ratio of facts to opinions within the domain. A domain that models opinions rather than facts make evaluating a NL querying system harder, as it more difficult to determine whether the systems answers are correct.
- A low degree of human ambiguity for the terms used within the domain. (I.e. terms are not overloaded so that there is a high probability of misinterpretation between two or more users).
- A reasonably small domain. A large domain would take a long time to model and domain modelling is not the purpose of this implementation.

- A low complexity of domain relationships (I.e. low difficulty in writing a SPARQL query to return a particular domain relationship).
- A sufficient amount of data is available in the domain. A lack of data might make it difficult to evaluate the full coverage of domain.

A domain that represents musical albums and all their associated entities, somewhat like that demonstrated by MusicBrainz[1], meets all the above criteria sufficiently.

5.2 Choosing a Triplestore Application

There are many triplestore applications available for download, such as Jena[2] (Carroll 04), Sesame[3] (Broekstra 02), Kowari[4] (Wood 05), Redland[5] (Beckett 01) and 3Store[6] (Harris 03). Several factors must be considered to determine which system is the best to use:

- Does the system provide interfaces with a familiar query language?
- 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 should help determine why some queries may not work and aid design of faster queries
- How easy is it to find and communicate with the developers / other users of the system?
- How fast is the system, both in importing and querying?

Jena does have a familiar SPARQL query interface. Jena can also be configured to store RDF data using the familiar MySQL database system. However there is no contact with the developers and only contact with a few users. In comparison to 3Store the general speeds of Jena seems to be quite a bit slower (Lee 04).

Sesame did not originally have a SPARQL query engine, it has it's own Sesame RDF Query Language (SeRQL) but a plug-in[7] has been built for it. Sesame can use MySQL as a repository (Broekstra 02). Like Jena, there is no contact with the developers or many users. Using MySQL as a repository, reading in roughly the equivalent number of

[1]http://musicbrainz.org
[2]http://jena.sourceforge.net/
[3]http://www.openrdf.org/
[4]http://www.kowari.org/
[5]http://librdf.org/
[6]http://sourceforge.net/projects/threestore/
[7]http://sparql.sourceforge.net/design-spec/design.html

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 appears to be a good choice especially as there is contact with the developers and a good number of users. 3Store also supports the familiar SPARQL query language and it is based on top of the familiar MySQL database system. 3Store was purposefully designed to store large amounts of RDF data so it should scale well.

Whilst using 3Store, as part of this test implementation, a certain number of bugs have become apparent that may make it difficult if not impossible to build an NL querying system using it. Redland, Kowari and several other triplestore applications have yet to be evaluated, these may prove to better suited to being a component of an NL querying system.

5.3 Designing an Ontology

MusicBrainz has its own ontology[8]. This ontology is not very well defined, running it through the University of Manchester's OWL Validator[9], it does validate as OWL Full but with a lot of additional messages. These messages mostly make reference to the wrong vocabulary being used. At the head of the file the owl:Ontology tag is used but throughout the ontology tags such as rdfs:Class are used rather than owl:Class. A further problem with this ontology is that it does not sufficiently express the domain, apparently an extended ontology exists but is not very easy to find. Due to these two factors it was only sensible to design a new ontology based on the original MusicBrainz Ontology. It should be possible to make this ontology OWL DL.

MyMusicOntology v1.1 (MMO1.1) [10], is a reworked version of the MusicBrainz ontology which has increased domain coverage and also validates as OWL DL using the University of Manchester's OWL validator. Figure 5.1 shows the domain coverage of MMO1.1. In designing the MMO1.1 Ontology the emphasise has been on ensuring that it only represents the domain's structure and not any instantiation data. The three classes Type, Status and MusicalActionType, may have any number of instantiations. It is almost inconceivable that Type and Status would ever need any 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

[8]http://www.ldodds.com/projects/musicbrainz/schema/index.rdf
[9]http://phoebus.cs.man.ac.uk:9999/OWL/Validator
[10]http://www.ecs.soton.ac.uk/˜drn05r/musicbrainz/mmo1.1

to represent these would be by re-defining the ontology each time. MusicalActionType could have an almost and 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 the users to define instantiations sensibly and to annotate and label the instantiations appropriately. As data input is not the major concern of the implementation, this problem can be overlooked for the time being.

5.4 Interface Design

As the NL querying system is using the knowledge technologies it is only sensible to have a web-based interface. At the present time this is a HTML form with a text box and submit button. When the NL query is submitted the output is printed on a new page. This interface is unlikely to develop greatly until the core of the implementation is complete, when evaluating the interface should be more straightforward.

By knowing the start point, the NL query and the end-point, the H-R answer, this provides an added criteria when choosing or building components for the implementation.

5.5 Choosing NLP(s)

The choice of NLP(s) should be carefully considered. There are a number of criteria to consider when choosing one or more NLPs:

- How easy is it for the system to call the NLP and get a timely output response?
- How easy is the output to process?
- How accurate is the NLP?
- How great is the coverage of the NLP?
- If more than one NLP is being used does this significantly improve the accuracy or coverage? (see section 2.3.1).

Both Stanford parser (Klein 03) and minipar (Lin 98) have executable applications that can be executed using PHP. Stanford Parser uses quite a large dictionary and therefore a server and client was built, so that the dictionary does not have to be read into memory each time, dramatically improving performance. Based on informal testing, both these NLPs seem to be quite accurate and have more than sufficient coverage. As Stanford parser is a rule-based NLP and minipar is a principal-based NLP, these two different approaches should hopefully complement each other to improve accuracy and coverage but this is yet to be proven.

5.6 Importing Data

Importing data for the implementation has the potential to be one of the more time-consuming tasks. It is important to be aware what data has been imported so that it is possible to be certain that the results returned are correct. As the MusicBrainz Ontology has been heavily adapted it is not possible to port data in from there. This left two real options, screen-scrapping music websites or providing some manual, most likely form-based submission system. Neither option is ideal but as screen-scrapping websites gives very little guarantee of gathering reliable data. Manual form-based submission despite being time-consuming, gives a much more reliable dataset. The size of the dataset need not be too big, as long as it has good domain coverage.

To both reduce the time taken for entry and to improve consistency submission forms have been designed to allow users to select options rather than forcing them to type in data. Figure 5.2 is the submission form for album entries, through using SPARQL queries the majority of the fields can filled through choosing an option from a list.

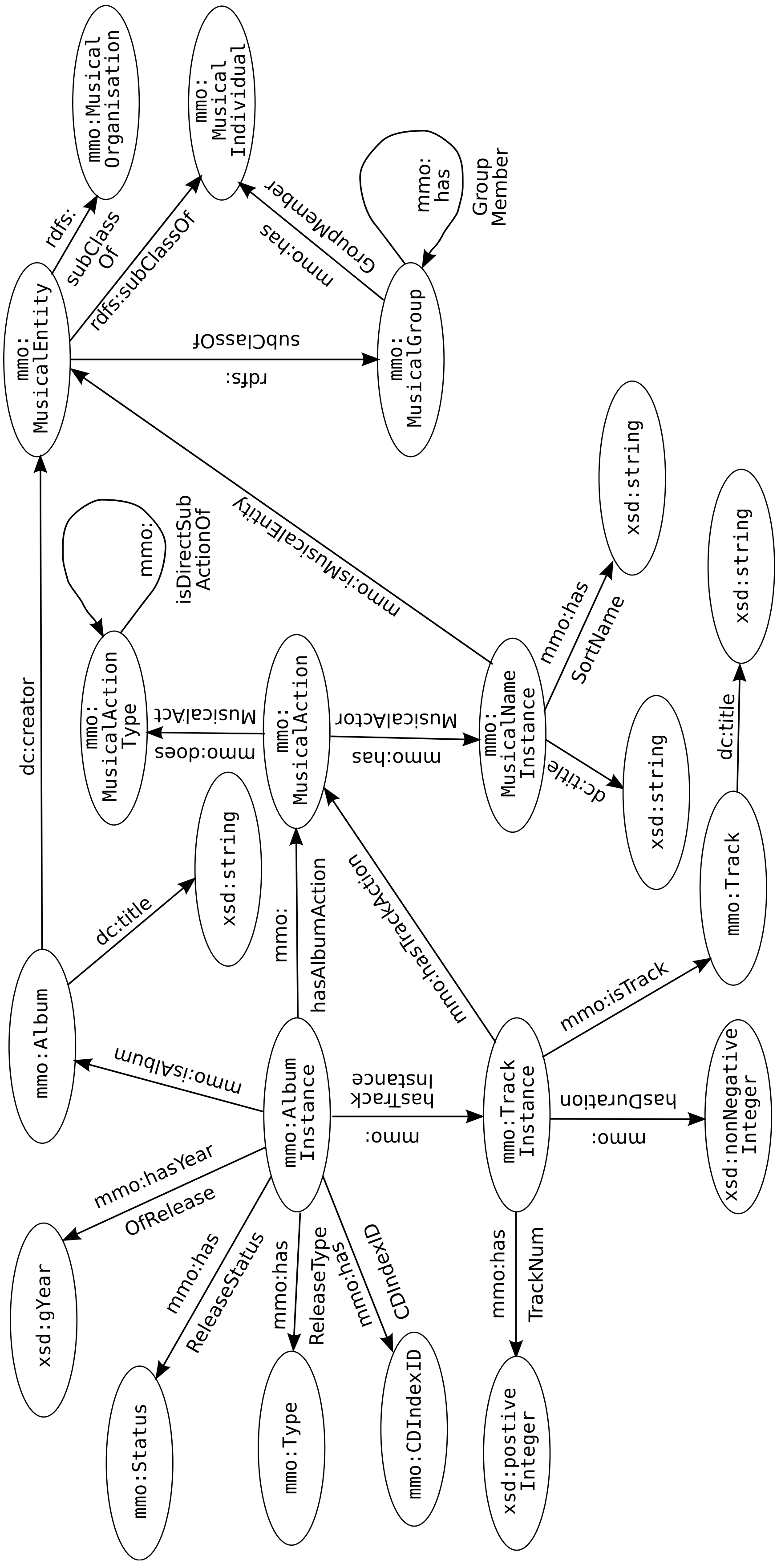
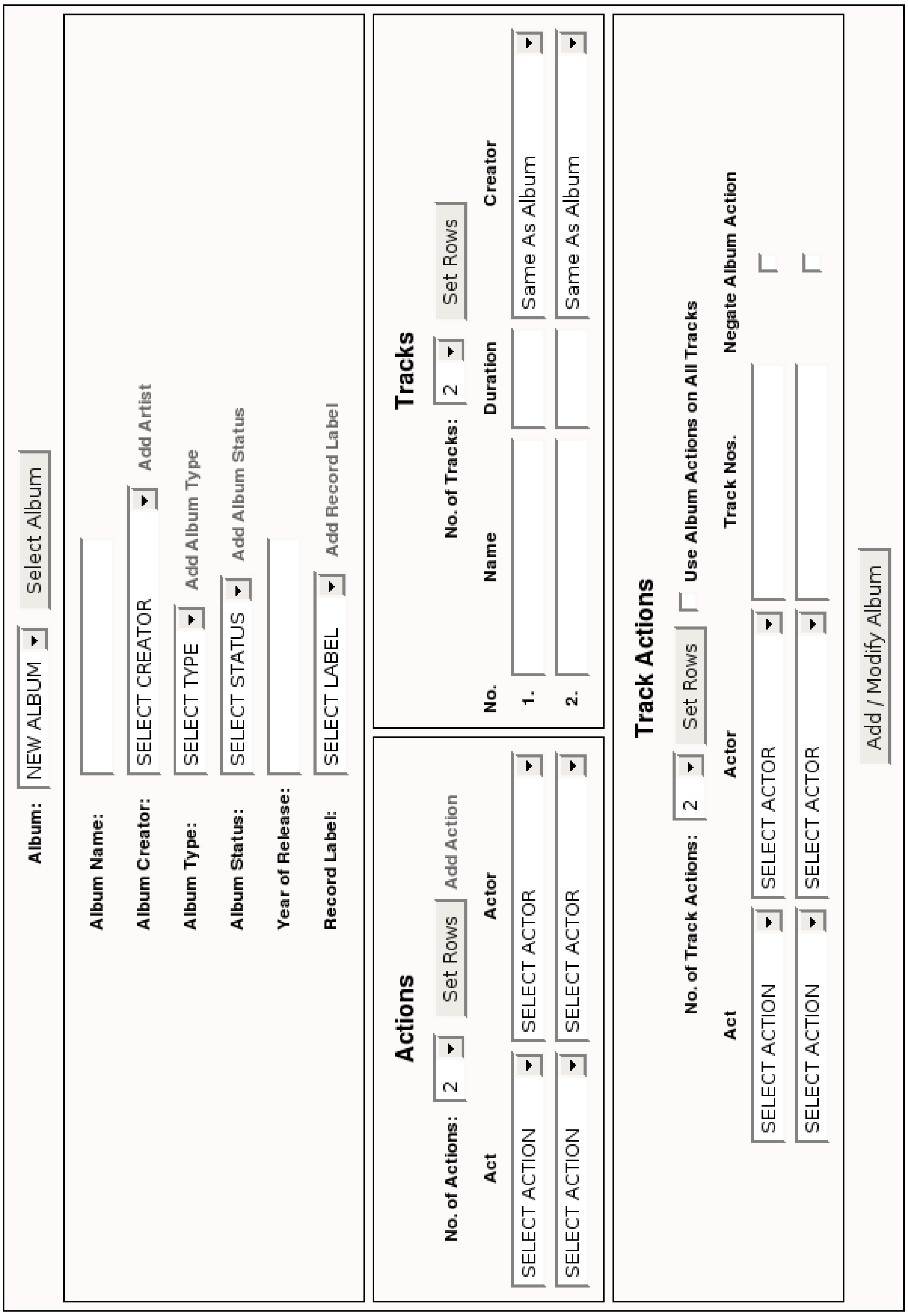


Figure 5.1: Domain Map based on MMO1.1 Ontology



Album: NEW ALBUM Select Album

Album Name:		
Album Creator:	SELECT CREATOR	Add Artist
Album Type:	SELECT TYPE	Add Album Type
Album Status:	SELECT STATUS	Add Album Status
Year of Release:		
Record Label:	SELECT LABEL	Add Record Label

Actions

No. of Actions: 2 Set Rows Add Action

Act	Actor
SELECT ACTION	SELECT ACTOR
SELECT ACTION	SELECT ACTOR

Tracks

No. of Tracks: 2 Set Rows

No.	Name	Duration	Creator
1.			Same As Album
2.			Same As Album

Track Actions

No. of Track Actions: 2 Set Rows ☐ Use Album Actions on All Tracks

Act	Actor	Track Nos.	Negate Album Action
SELECT ACTION	SELECT ACTOR		☐
SELECT ACTION	SELECT ACTOR		☐

Add / Modify Album

FIGURE 5.2: Album Submission Form

Chapter 6

Conclusions and Future Work

6.1 Conclusions

What is noticeable across a large number of existing NL querying systems / NLIDBs is that they concentrate on only one domain, e.g. LUNAR (Woods 72), PLANES (Waltz 78), ASK (Thompson 83), although ASK was designed generically so it could be used for more than one domain but not concurrently. This is mainly 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, any new system would be best to concentrate on one domain, although to make a system widely applicable it must be possible to swap between domains with ease.

One of the distinguishing features of the implementation proposed in Chapter 5 is that it uses SW knowledge technologies such as RDF, OWL and triplestores. These provide a new type of database that is specifically designed to be able to store knowledge. Looking back at Figure 6.1, the use of these technologies sets out a structure for defining data within a domain. This structure could be used across many domains making switching between one domain and another trivial.

The main purpose underlying the "Semantic Web" paradigm, is to change the current web from one that stores H-R knowledge as M-R data to one that can still produce H-R knowledge but stores it as M-R knowledge. By design, a NL querying system meets these criteria but it also goes one step further by providing a human-readable/writeable interface to find this knowledge.

There are many web-based applications that would find a NL querying system useful. As it is not uncommon to use a search engine such as Google to find singular bits of information, i.e. direct questions, the ultimate goal would be to design a NL querying system that had the appearance of a search engine but was capable of understanding questions and returning specific answers. For the reasons discussed earlier in this conclusion, there is clearly no current way to achieve this across the almost limitless number of domains defined on the web.

Ginseng (Guided Input Natural language Search ENGine) (Bernstein 06) is one of the first system that attempts to provide an NL search engine that uses SW knowledge technologies. It guides the user when they enter words for their question to try and ensure getting an NL question that uses known words and structure. This helps overcome one of the greatest problems, i.e. domain opacity (see section section 2.2). The user may be annoyed by this guidance, therefore it is only acceptable if the system does not prevent the user from asking any NL question that can be answered. It is also desirable that formality is minimised, after all these are suppose to be 'Natural' language questions.

Another reason why traditional search engines, such as Google have no genuine NL search engine competitors is because no RDF knowledge base has been built that is comparative in size to even a miniscule fraction of the web's knowledge base. The time taken to generate all the formatted data to put into a RDF triplestore, is one of reason for this but mainly it is due the poor scalability of triplestores at the present time. Just because a global coverage NL search engine is not currently feasible it does not mean there are not certain niches for this new type of NL querying system.

The Comb*e*Chem project has already designed and uses a digital lab book with a triplestore back-end (schraefel 04b). This digital lab book may benefit from having a NL querying system. It could be used to quickly find old plans or results or to perform fast lookups on chemical data books for information such as atomic weight, molecular structure, etc. A NL querying system may provide an advantage over having to find a table at a particular web address or having to navigate to results using a form or GUI-based system.

A remote interface for the digital lab book might also be useful when performing long timescale experiments. E.g. an experimenter might want to check that the temperature within the lab has remained below a certain temperature or even that an experiment has concluded. A NL querying system particularly comes into its own when the interfacing device is small such as a cellphone. 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.

Some Multi-User Dungeons (MUDs) use a pseudo NL formal query language, (that has a vastly reduced vocabulary and grammar compared to natural language), to find out information about the surrounding environment. In the original MUDs this interface was used to find out things about a virtual environment but with the invention of context-aware ubiquitous systems, a MUD's NL interface can be used to find out about a real environment (Cruickshank 04). There are distinct similarities between the Comb*e*Chem project example described previously and a real-world MUD system, such as the one for medical devices described in (Cruickshank 04), in particular the ideas of a ubiquitous monitoring system and using cellphones to interact with the system. Therefore its worth considering the other features considered within this system, such as the use of RSS feeds, as these may also be applicable to the Comb*e*Chem project example.

In theory, once the implementation described in Chapter 5 is completed, adapting it for the Comb*e*Chem project should just be a matter of designing the required ontologies as illustrated in Figure 6.1. Clearly it is unlikely to be that straightforward, most likely because having to handle a more complex domain will highlight flaws in the system that the MusicBrainz-based domain did not. In general though, the fact that SW Knowledge Technologies allow a domain's structure to be encapsulated into one or more ontologies and then allowing these ontologies to be put together is a bespoke manner is it's greatest advantage. There are several reasons for this:

- Various components of an NL querying system (as shown in Figure 6.1 can be defined in the same format to simplify interoperability.
- Simple swapping between domains, just by swapping the ontologies used.
- Generic logical reasoners can be applied upon the domain ontologies, data and even thesauruses

It also sensible to use these SW knowledge technolgies as they become a standard on the web because in the future rather than having to format the data for your NL querying system's knowledge base, it will be possible to use somebody else's knowledge base that they have made available over the web using these technologies. This concept is not a reality in existing relational databases despite some databases having web interfaces. Relational databases were envisioned before the web and therefore were not designed to be integrated within it. The next generation of the web wants to make finding specific information easier. The purpose of NL querying systems is to reduce the formality that an interface needs to be able to find that specific information.

6.2 Future Work

After reviewing a number of papers that use innovative techniques to improve the performance of there NL interfaces, such as Ellipsis (Hendrix 78) (Waltz 78) or storing logical concepts as a database of Prolog statements (Warren 82). Further study on NLIDBs papers may be useful to discover any other novel techniques that may help improve the performance of this NL querying system.

Only a small portion of being an NL querying system has been completed, there are still many tasks to be achieved. Looking back at Figure 2.1, one of the first tasks to complete is the evaluation of the outputs from the NLPs. To complete this task using only one NLP initially is best. Therefore all that is required is to put the NLP output into a suitable data structure, most probably a parse tree. Evaluation of output from more than NLP is probably better as an adaptation to the complete system and this would make it easier to test the difference in performance compared to a one NLP system.

Terminology standardisation also needs to be considered. One way of achieving this is to write an ontology that is able to define related terms and phrases and then design some interface to express these. Figure 6.1 shows where data and structure for a domain should be laid out. This diagram may well change as the NL querying system develops. Figure 6.1 also represents how there is a layer of class type instances that comes between the domain structure and the specific data being collected for that domain, as described in section 5.3. Using the thesaurus of related terms should take the data structure (i.e.

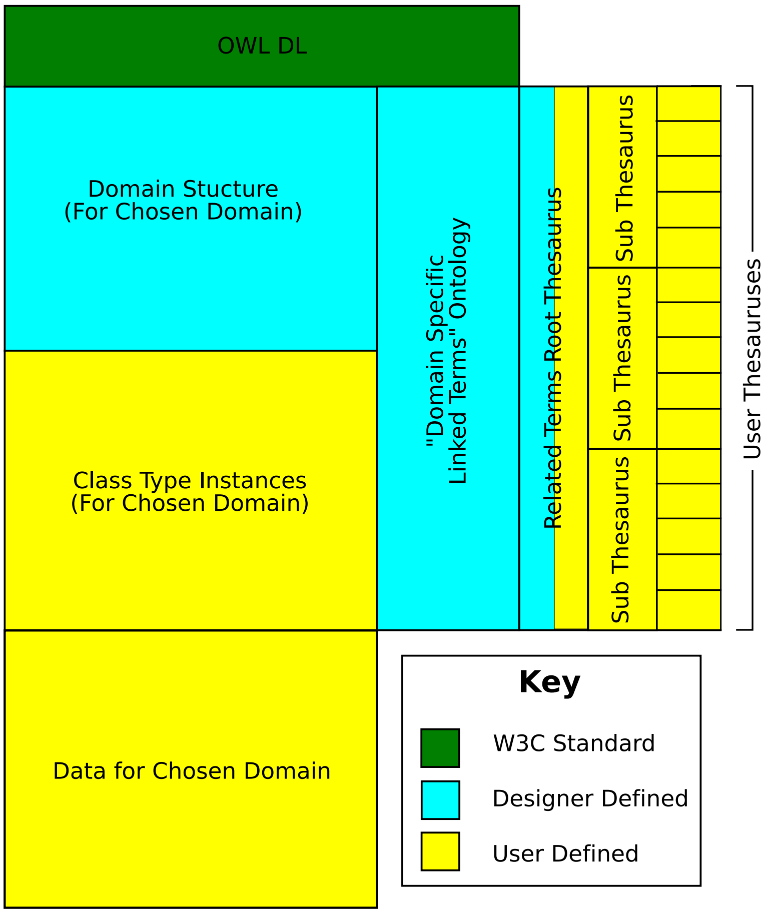


FIGURE 6.1: Architecture for Defining a Domain for a NL Querying System

a parse tree) that has already been generated and be able to replace elements with properties or classes defined by the ontology.

After terminology standardisation, the data structure should already contain elements of a SPARQL query, all that is required is extract them and express them in a legal SPARQL query. There will potentially be elements of the data structure that will not have been converted but are still required for query interpretation, e.g. "How many" which would be converted to a tag to tell the Query Result Interpreter (QRI) to count the number of results returned.

The final component of the NL querying system is the query result formatter. Designing this will require analysis of the QRI's output to work out the best format to display it. E.g. if there is a lot of data but it can be broken down sensibly into fields and records, then a table would probably be the best format, whereas only a small amount amount of data may be better expressed as one or several statements.

Once the system is complete, extensive user trials will be needed. Hopefully these user tests will give an insight into adaptations to the system to improve both its accuracy but also the speed that users can disseminate the information they require.

6.2.1 Future Work Plan

It is essential to plan the work discussed in 6.2 to ensure its completion in time for it to be reviewed as part of the mini-thesis report.

6.2.1.1 July - September 2006

1. Continue to review NL querying system papers to try to discover any novel approaches that may help improve the performance of the NL querying system implementation.
2. Generate a reasonable size dataset for the MusicBrainz-based domain and upload it to the triplestore.
3. Build a test set of possible NL queries that could be made to the MusicBrainz-based domain.
4. Use this test set to see how many of these queries each NLP parses correctly to determine which NLP to use.
5. Ensure the chosen NLPs output is read into a data structure that can be used by the terminology standardisation phase.

6.2.1.2 October - December 2006

1. Design the ontology for defining linked terms and start to built up the three varieties (root, sub and user) of thesaurus for the MusicBrainz-based domain.
2. Implement the code that replaces elements of the NLP-generated data structure with classes and properties from the MMO1.1 ontology.

3. Implement the code that converts the modified data structure to a legal SPARQL query.

4. Implement the query result interpreter.

6.2.1.3 January - March 2007

1. Implement the query result formatter.

2. Integrate any novel techniques discovered or additional features proposed, (e.g. using multiple NLPs), to try to improve performance of the system.

3. Conduct user tests to determine how effective the system is.

4. Write up mini-thesis report.

6.2.2 Future Work after Mini-Thesis Report

Assuming the results from the user tests described in section 6.2.1 have been reasonably successful, it may be possible to undertake some of the tasks suggested in the conclusion (see section 6.1). The conclusion describes how it should be possible to apply the test implementation to a different domain, in particular the Comb*e*Chem project domain. If this is successful a further task would then be to integrate the NL querying system into the existing digital lab book's pervasive system and to provide new interfaces to this pervasive system, e.g. cellphones using SMS or an instant messenger client over GPRS.

Bibliography

[Allen 87] J. Allen. Natural lanaguage understanding. The Benjamin/Cummings Publishing Company, Inc., 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 04a] S. Bechhofer, F. van Harmalen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider & L. A. Stein. *OWL Web Ontology Language Reference.* http://www.w3.org/TR/owl-ref/, February 2004. W3C Recommmendation.

[Bechhofer 04b] 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.

[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.

[Cruickshank 04] D. Cruickshank & D. De Roure. *A Portal for Interacting with Context-Aware Ubiquitous Systems.* In J. Indulska & D De Roure, editeurs, Proceedings of First International Workshop on Advanced Context Modelling, Reasoning And Management, pages 96–100, Nottingham, UK, 2004. University of Southampton.

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

[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.

[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.

[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 if rge Evaluation of Parsing Systems, 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.

[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.

[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 the 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.