UNIVERSITY OF PRETORIA
Faculty of Engineering, Build and
Information Technology

Department of Computer Science

Semantos: A semantically smart information query language

by

Theodorus Crous

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Master of Science (Computer Science)

In the Faculty of Engineering, Built

Environment and Information Technology

University of Pretoria

2008

Acknowledgement

I would like to thank the following people for all their assistance during my studies:

- Debbie-Lee for understanding and supporting the weekends and late nights spent reading papers and running experiments.

- My family for always believing in me and encouraging me to pursue my studies further.

- Professor Judith Bishop who has patiently guided and assisted me during my studies.

- Anne Archer for nursing the language in my transcripts.

Finally, many thanks to the National Research Foundation for the bursary that funded my studies.

Abstract

Semantos: A semantically smart information query language

By

Theodorus Crous

Supervisor: University-Professor Dr. Judith Bishop

Enterprise Information Integration (EII) is rapidly becoming one of the pillars of modern corporate information systems. Given the spread and diversity of information sources in an enterprise, it has become increasingly difficult for decision makers to have access to relevant and accurate information at the opportune time. It has therefore become critical to seamlessly integrate the diverse information stores found in an organization into a single coherent data source. This is the job of EII and one of the key components to making it work is harnessing the implied meaning or semantics hidden within data sources. Modern EII systems are capable of harnessing semantic information and ontologies to make integration across data stores possible. These systems do not, however, allow a consumer of the integration service to build queries with semantic meaning. This is due to the fact that most EII systems make use of XQuery, SQL, or both, as query languages, neither of which has the capability to build semantically rich queries. In this thesis Semantos (from the Greek word sema for "sign or token") is proposed as a viable alternative: an information query language based in XML, which is capable of exploiting ontologies, enabling consumers to build semantically enriched queries. An exploration is made into the characteristics or requirements that Semantos needs to satisfy as a semantically smart information query language. From these requirements we design and develop a software implementation. The benefit of Semantos is that it possesses a query structure that allows automated processes to decompose and restructure the queries without human intervention. We demonstrate the applicability of Semantos using two realistic examples: a query enhancement- and a query translation service. Both expound the ability of a Semantos query to be manipulated by automated services to achieve Information Integration goals.

Keywords: Enterprise Information Integration, Semantics, Ontologies, Query Language, XML.

Table of Contents – Chapters

List of Tables

List of Figures

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

List of Works

- Danzfuss, T, T Crous, A Liebenberg and A Moolman. “Adaptive Object Modelling using the .NET Framework.” *3rd Conf .NET Technologies*. Plzen, May 2005.

- Crous, T and J Bishop. “Semantos: XML-based Query Enhancement of RDF for Agents in the Semantic Web.” *Eighth International Conference on Web Engineering ICWE 2008*. New York, July 2008. Note: Accepted, but unable to attend due to logistical problems.

CHAPTER 1: INTRODUCTION

"Give me a lever long enough and a fulcrum on which to place it, and I shall move the world." -- Archimedes

1.1. BACKGROUND

This thesis explores how to leverage the semantics and ontologies captured in Enterprise Information Integration systems. In order to write meaningful queries in very large, decentralized data spaces (like the web) semantic information is required to provide the context mediation (Madnick 1995). This is where the family of technologies surrounding the Semantic Web comes into play. The Semantic Web is built around technologies that tag information, such as the resource description framework (RDF), and technologies that link meaning to these tags, like the resource description framework schema (RDFS) (Brickley and Guha 2000). When combined, RDF and RDFS allow for semantics or "meaning" to be linked to vast quantities of information. This information can then be interpreted and queried, in an intelligent way, by machines or agents (Shadbolt, Berners-Lee and Hall 2006).

Semantos is a technology which is primarily focused on how this information is queried; specifically the query language used. There are currently many query languages that can be selected for use in Information Integration systems. These languages range from the standard enforced XQuery (Boag and Chamberlin 2007) to the more recent and perhaps more applicable SPARQL (Prud'hommeaux and Seaborne 2007). It is the position of this researcher, that these standard languages are inadequate, as they do not facilitate the simple manipulation of these queries by people, software or, more importantly, software agents. This can be rectified by the use of a query language built from XML structures, instead of a language built on custom built syntax constructs. Both man and machine can process XML with ease. (Bray, et al. 2006).

	Data Model	Query allows ontologies	Query constructs	Queries at abstraction level
SQL	Relational	No	Standard Syntax	Syntactic
XQuery	Hierarchical	Sometimes	Custom Syntax	Syntactic
SPARQL	Graph	Yes	Custom Syntax	Semantic
Semantos	Graph	Yes	Pure XML	Semantic

Table 1 How Semantos weights up against other query languages

Table 1 provides a summary of how Semantos compares to other information query languages. From this summary we can see that Semantos has a graph data model, the concept of which is further explained a little further in this chapter. Semantos also allows ontology information to be used alongside normal query syntax to further enhance the expressive capabilities of the language. The queries are expressed as XML snippets, taken to mean that a Semantos query need not confirm to all the specifications of the XML DOM document specification, it does still have to be valid XML though. Semantos queries are executed at the semantic level of abstraction, of which a further discussion is made in chapter 3. Taking into account all these facets, it can be concluded that Semantos is better suited as a semantically smart information query language than SQL or XQuery, and although SPARQL weights up equally to Semantos in all other respects, it will be shown in this thesis that the addition of a pure XML syntax, permits several new and useful possibilities.

Before getting to grips with Semantos and the processes of querying large scale enterprise information sources using semantic technologies, we present a brief background regarding the two "technology spaces" which Semantos will have to successfully straddle. These two technologies are Enterprise Information Integration and the Semantic Web.

1.1.1. Enterprise Information Integration

Corporate data is hoarded in all manner of places and formats. Even the most modest of companies store a vast amount of data in the strangest of places. With tools like e-mail, word processing, spreadsheets, presentation documents, data bases, company portals and websites being part of every corporate work day, it is no wonder that companies spend a very large amount of time and effort trying to find information that they already posses. Not being able to find necessary information leads to duplication of effort and in many cases discrepancies in the information itself. This may be fine when ordering the stationary for the week, but consider when a company needs to decide whether or not they can afford to pay employee benefits based on the financial performance of the last year. In a corporate environment every piece of information is vital and to this point it is crucial that decision makers have access to the correct information at the opportune time. It is for this purpose that the concept of Information Integration was created (Giachetti 2004).

The two motivating factors behind Information Integration are access to real time information, and the need to integrate various forms of data. A good working definition of Information Integration, or, as it is better known, Enterprise Information Integration (EII), is provided by John (JT) Taylor from Software AG: “EII is the integration of data from multiple systems into a unified, consistent and accurate representation geared toward the viewing and manipulation of the data. Data is aggregated, restructured and relabeled (if necessary) and presented to the user.” In other words, Information Integration is rooted in Virtual Database technology and its accompanying disciplines, such as distributed query processing and data modeling (Delen, Nikunj and Perakath 2005). Information integration focuses on the integration of data from multiple systems into a unified, consistent and accurate representation without first loading the data into warehouse (Halevy, et al. 2005), which is then used for the viewing and manipulation of that data.

The process of integration is outlined in Figure 1, and demonstrates the different problems that need to be overcome in order to successfully integrate information. The first issue that needs to be resolved is to overcome the fact that data might not be stored in the same location and/or that data might not be stored in the same format or storage medium. This is usually achieved by aggregating data into an intermediate format and merging the data formats into a data source at a common location. The second issue is to resolve the structural heterogeneity that may exist between different data sources, meaning that a name field in one data source may be 20 characters long and 30 characters long in another data source. This can be accomplished by mapping attributes to meta data and then assigning structural relationships between matching attributes. The third issue requires different information representations to be matched up when they carry a similar meaning. This can be explained by the example that an employee table would also store information about people and would then technically also count as a people table. This matching can again be accomplished with meta data, although in this instance the meta data should also be inferable: if an employee table is relative to a person table and a person table is relative to a voter table, it should be reasonable that an employee table can also contain voter data. The final step is to merge the results from all the different data sources and representations into a coherent view of the data. Information Integration provides immediate benefit to end users who are required to deal with multiple systems on a daily basis. In a nutshell, the main purpose of an Information Integration platform is to make a collection of miscellaneous data sources look like a single database.

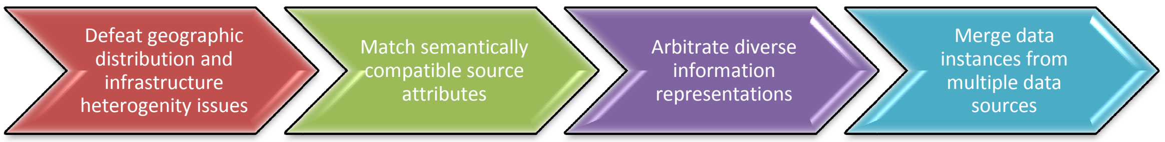

Defeat geographic distribution and infrastructure heterogenity issues
Match semantically compatible source attributes
Arbitrate diverse information representations
Merge data instances from multiple data sources

Figure 1 The process of Information Integration (Hauch, Miller and Cardwell 2005)

Data warehousing and its foundational ETL (Extract, Transform and Load) technologies have been around for many years and have also been providing corporate consumers with the ability to cross-examine and analyze data from several sources in a cohesive view. The two main drawbacks these systems suffer from though, is the inherent latency involved with the gathering and stockpiling of the data in central data stores, -warehouses and –marts and the focus on primarily catering for structured data which resides in databases. Information Integration addresses both these disadvantages by means of its virtualized database approach. First, Information Integration focuses on both structured and unstructured data, i.e. documents, images, and other media files. Second, it provides real-time access to information by means of federated queries over integrated views of disparate data sources. In short, EII holds the promise of providing decision makers with the tools to timely access relevant information in aid of underpinning important corporate choices.

1.1.2. Semantic Web

The World Wide Web is a very large, information rich, data store, which can be described as the "library of humanity". The key problem with all very large libraries concerns the extraction of relevant information at the desired time. Considerable effort has gone into building very complex information retrieval systems for the Web, most notably the modern search engine, which is the incarnation of the librarian in the digital age. The aid of this librarian is indispensable as far as finding pertinent information at the required time is concerned. So far, these "librarians" have done an adequate job of keeping track of the individual "books" or documents on the web, but, as is becoming evident, merely keeping tabs on books, is not sufficient for contemporary information retrieval needs. What is required, is access to the "paragraphs" of information contained inside the "books". Furthermore, it is necessary that the details or content of these Web documents can, in turn be searched, aggregated, categorized and analyzed.

This fundamental problem is the main reason for the invention of the Semantic Web. The Semantic Web may realize the full potential of the enormous digital library that is the Web. By tagging individual elements of data, the Semantic Web is able to open the door for very refined searches on the information contained inside web documents. These tags are different from the traditional HTML mark-up tags used on the Web today. In the first place, they have nothing to do with document layout and visual styles. Most importantly, they describe the semantic relationships between data elements, and so provide a reasonable conceptual framework for automated software agents to "think" about the information contained within the document.

Tim Berners-Lee's initial vision for the Semantic Web in 2001 (Berners-Lee, Hendler and Lassila 2001) entailed information avatars or software agents crawling the web, looking for information that would be relevant to our personal needs. This is one of the foundations of the Semantic Web: making information available to automated software systems. Such a system would allow us to manage the vast quantity of information on the web by allowing machines to do the searching, categorization and analysis for us. Now, several years later, this agent assisted framework has not realized its full global potential and is still relegated to academic implementations - with a few exceptions (Shadbolt, Berners-Lee and Hall 2006).

The primary driving force behind the semantic web is to address the large volume of data in the World Wide Web. When he originally envisioned the semantic web Tim Berners-Lee attempted to untangle the mess of large data spaces by exposing the implied semantic associations between data items in an explicit way. This would enable machines to reason about the information contained in data and not just fetch the data. Because it takes inhuman reasoning abilities to search the internet for the right information, the process needs to become automated to the extent where a person can make a simple request in a natural human language. The request should then be carried out by a digital agent on the World Wide Web, which is tasked with searching the internet (Ding, et al. 2004), tirelessly looking for matches to the originator's request, by using its reasoning and logic capabilities. The true benefit of this idea can be sampled by using a normal web browser today. However, when searching for anything in the internet, the best search engines may occasionally fail to comply with even the most modest request. This is not due in any matter to the simplicity of search engines as, in fact, they have become complex examples of software engineering in themselves. The problem posed to search engines, rather, is that due to the sheer volume of data on the internet, the information has become un-indexable. Figure 2 provides the architecture of the semantic web.

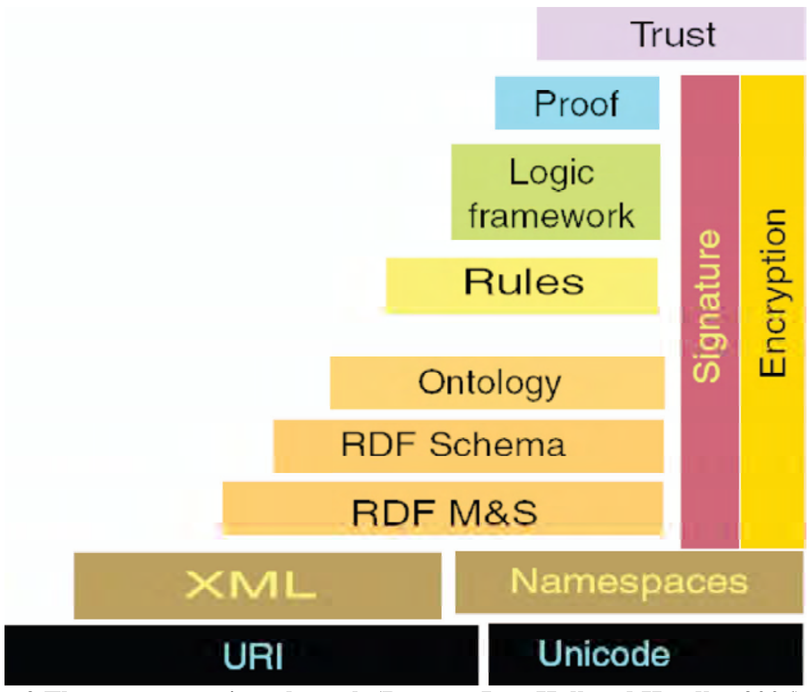


Figure 2 The new semantic web stack (Berners-Lee, Hall and Hendler 2006)

1.2. SEMANTOS OVERVIEW

An information query language does not exist in a vacuum and is subject to many external motivating factors. Various technologies and decisions influence the way in which a language is constructed and ultimately the way in which it is used. In this section some of the elements that influence the construction of Semantos are identified and discussed.

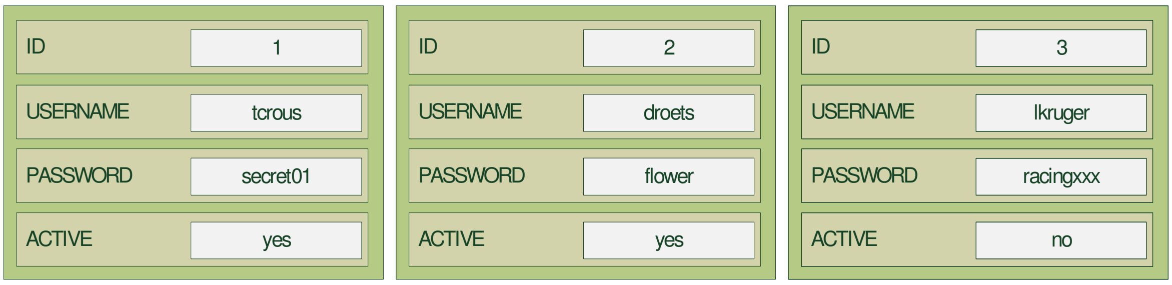
1.2.1. DATA MODELS

When deciding which data model to use for Information Integration purposes, there are three distinct models to choose from. The choice of model is important, as it influences what type of data can be stored and, specifically, how the stored data may be queried. The first model is the relational data model which has become quite a commonplace and standard model for storing and retrieving data. The relational data model is queried using SQL (International Organization for Standardization 2003). Another popular data model is the hierarchical data model, which is normally represented by XML and may be queried using XQuery (Boag and Chamberlin 2007). The final data model which may be selected, is the graph model, which is a newcomer to the scene and can be queried using languages like SPARQL (Prud'hommeaux and Seaborne 2007). Each of these three data models has different strengths and weaknesses, so that some are more suitable than others for Information Integration (Melton 2006).

ID	USERNAME	PASSWORD	ACTIVE
1	tcrous	secret01	yes
2	droets	flower	yes
3	lkruger	racingxxx	no

Figure 3 a) The relational data model

The relational data model is more suited to structured data and is the model most employed in traditional Data Base Management Systems (DBMS). In the relational model data is represented as rows and columns (see Figure 3a). Each row represents a fundamental piece of information and each column signifies a projected attribute of the information. Another aspect of the relational model is that all the rows of information must possess the same attribute set. It is not possible to store biscuit recipes in the same space as the specifications for a helicopter. It is possible however, to group different data “things” together in containers called tables. Each table has a unique set of columns and can contain data with similar attributes. Given its broad appeal in Relational DBMS (RDBMS) systems, a lot of research and development has gone into the efficiency and capacity of the relational data model. It is therefore suited to manipulate very large amounts of data with great ease.



ID	1
USERNAME	tcrous
PASSWORD	secret01
ACTIVE	yes

ID	2
USERNAME	droets
PASSWORD	flower
ACTIVE	yes

ID	3
USERNAME	lkruger
PASSWORD	racingxxx
ACTIVE	no

Figure 4 b) The hierarchical data model

The hierarchical data model is more suited for semi-structured- , “messy”- or incomplete data (omitted values, duplicates values etc.). Markup languages are the technologies that practically underwrite the hierarchical model. These markup languages notably include HTML for creating web pages and XML for storing and transporting more general data. The hierarchical data model relies on a tree like structure to store information. Each piece of information can be seen as a node and each node may have zero or more children. Nodes may not however have ancestral nodes as child nodes, prohibiting loops from taking place (see Figure 3b). What the hierarchical model offers, that the relational model can’t, is the ability to store dissimilar or unrelated data together. It is therefore not necessary to work with artificial data containers or tables, nor is it necessary to have predetermined columns or projections.

The graph model is suited to unstructured data and tends to be more complex in its processing requirements than the previous models. The largest data store on the planet, the World Wide Web, is arguably the best example of a graph data model around. If we were to take each web page as a node and the hyper links between pages as edges, then we would have a graph: a tree like structure that allows ancestral nodes to be child nodes. It is interesting to note that these models imply each other, in that the hierarchical model can be used to represent relational models, and the graph model can in turn be used to represent hierarchical models. This is illustrated in Figure 3a, Figure 3b and Figure 3c. The columns of a relational model can be represented by nodes in the hierarchical model. Each node then represents a column and the row relationship between data artifacts can be represented by grouping the nodes together. It is a simple task for the graph model to represent a hierarchical model, as a hierarchical model is already a graph model, except with the constraint of no loops. Notably however this process cannot be easily reversed without losing some data fidelity. As the crux of Information Integration is to draw information from as many different data sources (structured-, semi-structured and unstructured), the only viable data model for Semantos appears to be the graph model.

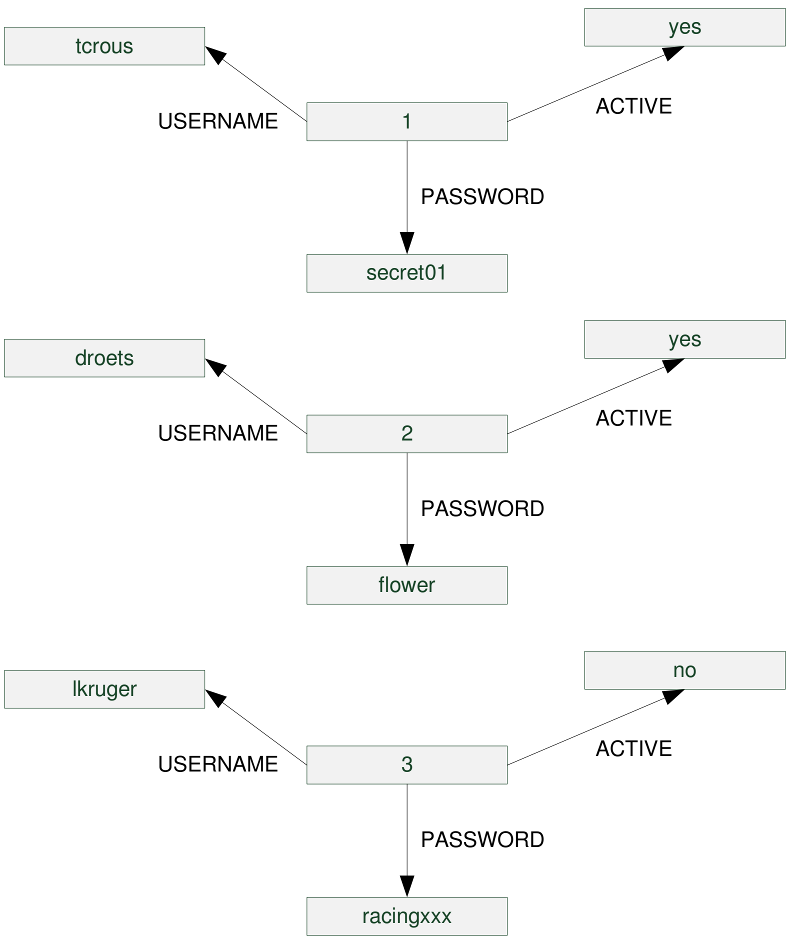


Figure 5 c) The graph data model

1.2.2. Semantic Web for Integration

At the core of the semantic web recommendations is the RDF graph (Klyne and Carrol 2004), which is proposed as a universal data structure. In essence, an RDF graph is a set of triples (S, P, O), where P names a binary predicate over (S, O); S is the subject and O the object. Using mapping ontologies (Heflin, et al. 2006) in combination with the RDF universal data structure allows Information Integration across a wide variety of data sources, including structured (i.e. RDB); semi-structured (i.e. XML) and unstructured (i.e. HTML) (Gutierrez, Hurtado and Mendelzon 2004). The use of mapping ontologies for integration purposes is shown in Figure 6. This powerful flexibility of the semantic web makes it the perfect choice for an integration technology. It is therefore pivotal for Semantos to be able to query RDF data sources and interact with the semantic web in order to be an information query language.

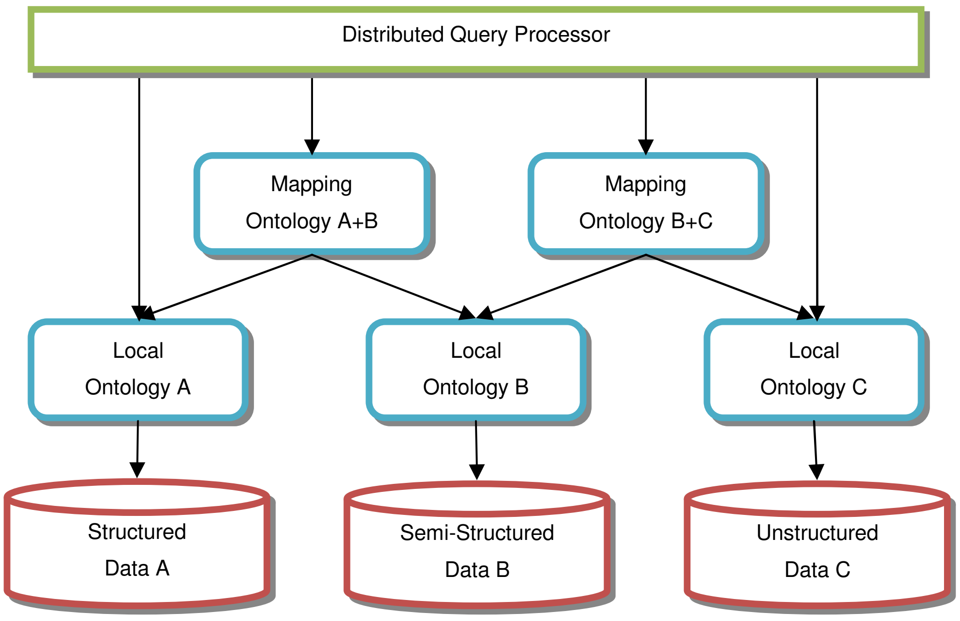


Figure 6 Mapping ontologies for Information Integration (Heflin, Dimitrov and Qasem 2006)

1.3. IMPLEMENTATION ENVIRONMENT AND LIMITATIONS

The current implementation of Semantos uses RDF/RDFS data sources, this is however merely an implementation choice and can be changed, as the Semantos syntax is not bound to any RDF/RDFS constructs. The Semantos query is parsed, inferred and executed by a .NET RDF/RDFS engine. For the purposes of this thesis the details of query optimization, are alluded to, but it is noteworthy that Semantos queries need to be optimized in order to implement a successful information query language (Greco, Greco and Trubitsyna 2005).

Semantos is implemented in C# using the Microsoft .Net 3.5 framework. The choice of implementation technology was made based on the flexibility of the language and its ease of use. Another deciding factor was the availability of ADO.NET, which provides enhanced data access features for Information Integration purposes (Baldassarre, Caivano and Visaggio 2005). The most important benefit gained from using the Microsoft .Net 3.5 framework, is the enhanced features for working with XML data sources. These features are part of the LINQ technology; which is examined in a later section. The core of Semantos is implemented as a web service; which leverages the service oriented architecture (SOA) (Sikka 2005), enabling greater flexibility and a wider range of integration options for the technology.

1.4. EVALUATION CRITERIA

Before designing the query language, it is necessary to identify the characteristics the language must possess and the requirements it should satisfy. The design criteria for Semantos share similarities with the characteristics of an XML query language (Bonifati and Ceri 2000). The criteria for Semantos are outlined below.

1.4.1. XML OUTPUT

The resultant output from a Semantos query must be in XML format. This has many benefits, including aiding the composition of queries and the ability to have derived entities (data views) defined via a single query. Probably the greatest benefit is the ability to manipulate the resultant data via XSLT or any other XML handling technology. It is also possible to embed the results from a Semantos query directly into HTML for simple and easy result representation on the Web.

1.4.2. XML REPRESENTATION

A Semantos query must be represented by XML. This property ensures the simple storage and transportation of queries. It also satisfies the programmatic manipulation criteria. A multitude of doors are opened when the language is actually presented in XML format. Not only is modification of the language syntax a simple matter of updating the schema, but many additional benefits are also gained by the inherent serializability of XML and its popularity on the web.

1.4.3. MUTUALLY EMBEDDABLE WITH XML

It must be possible to embed a Semantos query within an XML document. The converse is also possible: having arbitrary XML markup embedded within a Semantos query. Semantos elements are identified by the namespace in arbitrary XML documents. The great benefit of this is that it is possible to embed a Semantos query directly into an XHTML markup page. The query could be processed before the results are passed from the server to the client and the embedded XML inside the Semantos tags could be used to format the results. This is very similar to what XQuery does (Miller, Seaborne and Reggiori 2002).

1.4.4. SERVER-SIDE PROCESSING

Semantos must be suitable for server-side processing. Queries are self-contained and remotely executable. They are not dependent on resources available at the time of creation, when they are being evaluated. This is probably the most important language attribute that a query language for the Semantic Web should possess.

1.4.5. NO SCHEMA REQUIRED

Semantos must be capable of querying a data source without prior knowledge of its structure or schema. This capability means that it is possible for the language to be used against an XML source with limited knowledge of the structure of the data source. This property is important when considering the heterogeneity of data structures available on the Web.

1.4.6. PROGRAMMATIC MANIPULATION

It must be possible to create and manipulate Semantos queries using programs. This capability is necessary, as most queries will not be written by users, but rather by tools in application development environments. Programmatic manipulation goes to the heart of the usefulness of the language in agent assisted searching.

1.4.7. SUPPORT NEW DATA TYPES

Semantos must be fully extended with regards to data types and the language itself holds no relevance to the type of data being processed. This is especially important when considering that the language should be able to query data sources where the data types are not known upfront.

1.5. DIFFERENCES FROM OTHER WORKS AND OUR CONTRIBUTIONS

The contribution this thesis strives to make is the development of a new information query language which enables Information Integration on a semantic level. Neither the idea of Information Integration nor the idea of a semantic query language is new. Information Integration has been studied in depth and although the common consensus suggests that it is more a goal than a product, research in this domain is carried out every day. Even more active is the research field of the Semantic Web and specifically in our case query languages: for example (Karvounarakis and Magkanaraki 2003) and (Vdovjak, et al. 2003). What makes Semantos different and unique is that it is a query language that is as easy to manipulate for machines as it is for people. This key differentiator will in the opinion of the author result in better take up of semantically smart search technologies and therefore also improve the global appeal of the Semantic Web.

In the introductory text we have presented key characteristics or requirements to which Semantos must adhere. Given these requirements we designed and developed a software implementation of the Semantos language framework. Armed with the implementation we carried out rigorous experimentation and testing against each of the criteria. We succesfully proved that it is indeed possible to develop and program a Semantos software framework: it is not just a theoretical model. We demonstrate the applicability of Semantos using two realistic examples: a query enhancement service and a query translation service. Both cases clearly illustrate the ability of a Semantos query to be manipulated by automated services to achieve Information Integration goals.

1.6. OUTLINE OF THE DISSERTATION

The following outlines the remaining chapters that forms part of this research and serves as a roadmap of the work done:

Chapter 2 – Semantic Language Fundamentals: This chapter presents a literature study on RDF query languages. The fundamentals learned from current semantic languages are applied to Semantos at the design level.

Chapter 3 – Semantos Overview: In this chapter we present the language specification for Semantos. This determines the syntax and capabilities of the language. Each element of the language structure is also further explored

Chapter 4 – Design and Implementation: A Semantos implementation is illustrated in this section. Attention is paid to the processing loop and a step by step account is provided of the query processing itself.

Chapter 5 – Evaluation: Based on the evaluation criteria set out in the first chapter, this chapter continues to evaluate the implementation of Semantos from chapter 4 and provides feedback on the success and failure of Semantos to meet the initial requirements.

Chapter 6 – Use Cases: This chapter provides 2 distinct use cases that exemplify the unique implementation possibilities that Semantos offers. The first looks at the possibility of using Semantos as an intermediary language between existing RDF languages and the second investigates the use of Semantos in a query enhancement service.

Chapter 7 – Conclusions and Future work: This chapter identifies future work and concludes by listing the contributions made by this work.

Appendix A – XML Schema: A full XML schema for the Semantos query language constructs.

Appendix B – Examples: This chapter presents several examples of Semantos queries and the processing logic behind them.

CHAPTER 2: SEMANTIC LANGUAGE FUNDAMENTALS

"A scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die and a new generation grows up that is familiar with it." -- Max Planck

The fundamental purpose of a semantic language is to be able to construct questions about the connections between resources in a manner that allows software to interpret and act upon these questions and connections. In its most basic form the semantic web boils down to the description of these connections between resources and is designed to allow reasoning and inference capabilities to be added to the descriptions. This includes stating facts such as "a hex-head bolt is a type of machine bolt" (Berners-Lee, Hendler and Lassila 2001), but also stretches to the deduction of complicated inter-relationships. This important characteristic is what allows intelligent software agents to not only collect descriptions but to interpret and act on them as well. The semantic web works on top of the existing web, by adding machine-readable information without modifying the currently existing web.

At the core of the semantic web recommendations is the RDF graph (Klyne and Carrol 2004), which is proposed as a universal data structure. In essence, an RDF graph is a set of triples (S, P, O), where P names a binary predicate over (S, O); S is the subject and O the object. These triples allow for the creation of expressive statements regarding resources or in other words to create connections between them. Figure 7 presents a simple RDF graph describing an article written by "Cody Burlson". The most fundamental benefit of RDF compared to other meta-data approaches is that using RDF, you can say anything about anything. Anyone can make RDF statements about any identifiable resource. Using RDF, the problems of extending meta-data and combining meta-data of different formats, from different schemas disappear, as RDF does not use closed documents (Nilsson 2001).

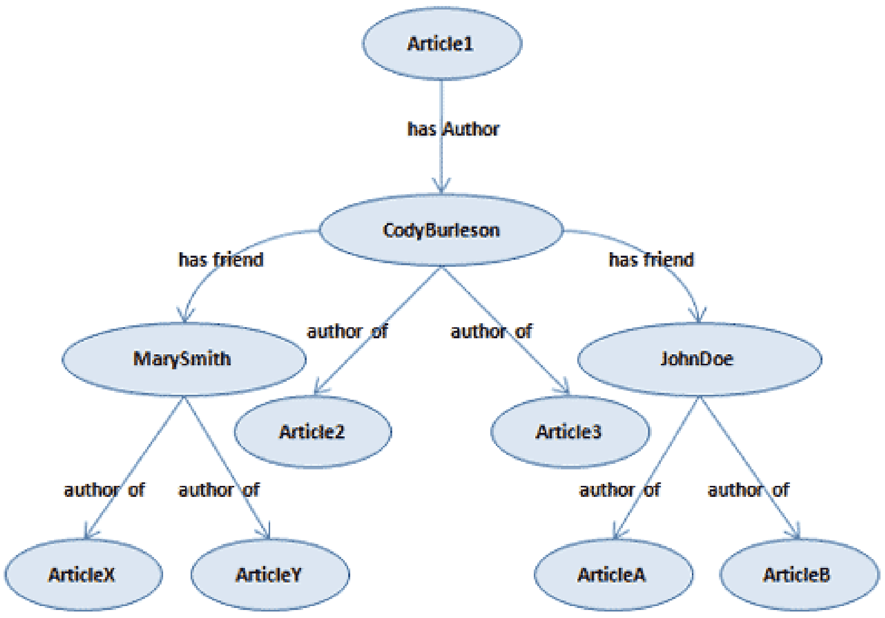


Figure 7 An example RDF graph.

2.1. RDF/RDFS

The semantic web backbone is the W3C's RDF/RDFS (Brickley and Guha 2000). At its core, the RDF model defines subject-predicate-object triples that are used to tag pieces of data and align them to a bigger picture or RDF graph (Carroll and Stickler 2004). This graph then represents all the information in the data source. RDFS defines additional relationships between these triples and provides for the ability to create rich ontologies or namespaces that define the objects, terminologies and semantics that are used in an RDF graph. This allows the user to query two related data sources, even if they use different triple assignments. That is, provided there is a unifying ontology that maps triples from the one graph to the other. Semantos is a RDF/RDFS query language, based in XML. The expression of RDF triples can be constructed in many forms, of which the most popular is to use XML, an example of which is provided in Figure 8.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
```

```
  <rdf:Description
   rdf:about="http://www.retrorabbit.co.za/aboutus.aspx">
     <dc:title>About Us</dc:title>
   </rdf:Description>
</rdf:RDF>
```

Figure 8 RDF/XML structure example

The same statement in Figure 8 can also be expressed in Notation 3, also known as simply "N3" (Berners-Lee 2006). N3 is a less verbose way of constructing triples, as can be seen by the same statement as before in N3 notation, Figure 9.

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
<http://www.retrorabbit.co.za/aboutus.aspx> dc:title "About Us";
```

Figure 9 Example statement in N3

Although RDF/RDFS has been commonly acknowledged as the language to describe metadata information on the semantic web, the question of which query language to use for the RDF/RDFS metadata, has been hotly contested for the last couple of years. These languages include RQL (Karvounarakis, et al. 2002), RDQL (Miller, Seaborne and Reggiori 2002) and even purely mathematical languages (Frasincar, et al. 2004). Formal definitions for RDF query languages have been compiled (Gutierrez, Hurtado and Mendelzon 2003) and various comparisons between the languages have also been done (Haase, et al. 2004), these results are not reproduced in this report. However, for the purposes of Semantos it is pertinent to note that it is the first RDF query language to be fully represented in XML. Unlike XQueryX (Melton and Muralidhar 2005) which is simply a mapping of XQuery syntax onto an XML representation, Semantos is an altogether XML native construct.

2.2. RDF QUERY LANGUAGES

There are several languages available with which RDF data sources may be queried. Although they all support different subsets of features, these languages have a lot in common. All the query languages provide some syntax with which to project the attribute result set, as well as providing some syntax for defining the data graph that needs to be queried. All the languages also provide features to limit the result set as well as include external namespaces. A short list of common RDF query languages are provided here.

2.2.1. RQL

RQL is a query language for RDF and RDF Schema, which is loosely based on OQL (Karvounarakis, et al. 2002). RQL has a powerful feature in its ability to address RDF Schema semantics in the language itself. Specific language constructs cater for class instance relationships, class property subsumption, domains and ranges (Broekstra 2004). An example of a typical RQL query is provided below, in Figure 10:

```
SELECT
 name, spousename

FROM
 {person}, human:name, {name},
 {person}, human:hasSpouse, {spouse},
 {spouse}, human:name, {spousename},
 {person}, rdf:type, {X : human:Woman}

WHERE
 name = “Theo”

USING NAMESPACE
       rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#,
    human = http://www.inria.fr/2006/12/05/humans.rdfs#
```

Figure 10 Example of an RQL query

RQL is a typed language following a functional approach and supports generalized path expressions, featuring variables on both nodes and edges, or classes and properties. In contrast to other triple-based RDF query languages RQL relies on a formal graph model that captures the RDF modeling primitives. The formal graph permits the interpretation of resource descriptions by means of one or more schemas. RQL supports (Karvounarakis and Christophides 2003):

- XML Schema data types (for filtering literal values)
- grouping primitives (for constructing nested values)
- arithmetic operations (for converting literal values)
- aggregate functions (for extracting statistics)
- namespace facilities (for handling different schemas)
- metaschemas querying (for browsing schemas)
- duplicate elimination (select distinct)
- quantification iterators (EXISTS, FORALL)
- recursive traversal of class and property hierarchies (for pattern-matching)

The RQL software consists of four modules: the parser, which analyzes the syntax of queries; the graph constructor, which collects the semantics of queries, especially concerning typing and interdependencies; the SQL translator, which rewrites RQL to efficient SQL queries; and the evaluation engine, which executes the SQL queries against the underlying PostgreSQL database. Below, Figure 11 provides the full BNF grammar of RQL.

```
ns_query       ::= (query | strict_query) [ "using namespace" nsdeflist ]
query          ::= "(" query ")"
                 | "subClassOf" [ "^"] "(" query ")"
                 | "superClassOf" [ "^"] "(" query ")"
                 | "subPropertyOf" ["^"] "(" query ")"
                 | "superPropertyOf" ["^"] "(" query ")"
                 | "topclass"
                 | "topproperty"
                 | "leafclass" [ "(" query ")" ]
                 | "leafproperty" [ "(" query ")" ]
                 | "nca" "(" query "," query ")"
                 | "domain" "(" query ")"
                 | "range" "(" query ")"
```

```
                | "typeOf" "(" query ")"
                | "namespace" "(" query ")"
                | "count" "(" ( query | strict_query ) ")"
                | "avg" "(" (query | strict_query) ")"
                | "min" "(" (query | strict_query) ")"
                | "max" "(" (query | strict_query) ")"
                | "sum" "(" (query | strict_query) ")"
                | "bag(" query, [ query ]")"
                | "seq(" query, [ query ]")"
                | query "[" query "]"
                | query "in" (query | strict_query)
                | (query | strict_query) set_op (query | strict_query)
                | query comp_op query
                | query bool_op query
                | "not" query
                | constant
                | identifier
                | var
                | sfw_query
                | "exists" var "in" (query | strict_query) "such that"
               query
                | "forall" var "in" (query | strict_query) "such that"
               query
strict_query   ::= "^" identifier
                | "^" var
                | "(" strict_query ")"
sfw_query      ::= "select" [ "distinct" ] projslist "from" rangeslist [
               "where" query ]
comp_op        ::= "<" | "<=" | ">" | ">=" | "=" | "!=" | "like"
set_op         ::= "union" | "intersect" | "minus"
bool_op        ::= "and" | "or"
constant       ::= integer_literal
                | real_literal
                | quoted_string_literal
                | quoted_char_literal
                | date
                | "true"
                | "false"
                | "&" identifier
var            ::= data_var | class_var | type_var | property_var
data_var       ::= identifier
class_var      ::= "$" identifier
type_var       ::= "$" "$" identifier
property_var   ::= "@" identifier
projslist      ::= "*" | query { "," query }
rangeslist     ::= pathexpr { "," pathexpr }
pathexpr       ::= pathelem { "." pathelem }
pathelem       ::= [ "{" from_to "}" ] (query | strict_query) [ "{"
               from_to "}" ]
from_to        ::= [ data_var ] [ ";" [ "^" ] (class_var | type_var |
               identifier) ] | class_var | type_var
nsdeflist      ::= nsdef { "," nsdef }
nsdef          ::= identifier "= " "&" identifier
```

Figure 11 BNF grammar for RQL

2.2.2.RDQL

The RDF Data Query Language or RDQL is a query language for RDF based on SquishQL (Miller, Seaborne and Reggiori 2002). The syntax for RDQL follows a select pattern comparable to SQL, where the “from” clause is similarly omitted (Haase, et al. 2004). RDQL does not support the incorporation of RDF Schema information. A typical RDQL query is provided below, in Figure 12:

```
SELECT
    ?name, ?spousename

WHERE
    (?person, human:name, ?name),
    (?person, human:hasSpouse, ?spouse),
    (?spouse, human:name, ?spousename),
    (?person, rdf:type, human:Woman)

AND
    ?name = “Theo”

USING
    rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    human FOR <http://www.inria.fr/2006/12/05/humans.rdfs#>
```

Figure 12 Example of an RDQL query

An RDQL query consists of a graph pattern, expressed as a list of triple patterns. Each triple pattern is comprised of named variables and RDF values which can be URI’s or literals. The query can also have a set of constraints on the values of the variables, and a list of variables required for the result set. The RDF graph is treated as data by an RDQL query, if inference is provided by the graph, it will appear as "virtual triples" and RDQL will include these triples as possible matches for triple patterns. There is therefore no distinction between inferred- and ground triples. The BNF grammar for RDQL is provided in Figure 13.

```
Lexical Tokens:
QuotedURI      ::= '<' URI characters (from RFC 2396) '>'
NSPrefix       ::= NCName As defined in XML Namespace v1.1 and XML 1.1
LocalPart      ::= NCName As defined in XML Namespace v1.1 and XML 1.1
SELECT         ::= 'SELECT'    Case Insensitive match
FROM           ::= 'FROM'      Case Insensitive match
SOURCE         ::= 'SOURCE'    Case Insensitive match
WHERE          ::= 'WHERE'     Case Insensitive match
AND            ::= 'AND'       Case Insensitive match
USING          ::= 'USING'     Case Insensitive match
Identifier     ::= ([a-z][A-Z][0-9][-_.])+
EOF            ::= End of file
COMMA          ::= ','
INTEGER_LITERAL ::= ([0-9])+
FLOATING_POINT_LITERAL ::= ([0-9])*'.'([0-9])+('e'('+'|'-')?([0-9])+)?
STRING_LITERAL1 ::= '"'UTF-8 characters'"' (with escaped \")
STRING_LITERAL2 ::= "'"UTF-8 characters"'" (with escaped \')
LPAREN         ::= '('
RPAREN         ::= ')'
COMMA          ::= ','
DOT            ::= '.'
GT             ::= '>'
LT             ::= '<'
BANG           ::= '!'
TILDE          ::= '~'
HOOK           ::= '?'
COLON          ::= ':'
EQ             ::= '=='
NEQ            ::= '!='
LE             ::= '<='
GE             ::= '>='
SC_OR          ::= '||'
SC_AND         ::= '&&'
STR_EQ         ::= 'EQ'  Case Insensitive match
STR_NE         ::= 'NE'  Case Insensitive match
PLUS           ::= '+'
MINUS          ::= '-'
STAR           ::= '*'
SLASH          ::= '/'
REM            ::= '%'
STR_MATCH      ::= '=~' | '~~'
STR_NMATCH     ::= '!~'
DATATYPE       ::= '^^'
AT             ::= '@'

Grammar:
CompilationUnit ::= Query <EOF>
CommaOpt       ::= ( <COMMA> )?
Query          ::= SelectClause ( SourceClause )? TriplePatternClause
               ( ConstraintClause )? ( PrefixesClause )?
SelectClause   ::= ( <SELECT> Var ( CommaOpt Var )* | <SELECT> <STAR> )
SourceClause   ::= ( <SOURCE> | <FROM> ) SourceSelector
               ( CommaOpt SourceSelector )*
SourceSelector ::=  QName
TriplePatternClause ::=  <WHERE> TriplePattern
               ( CommaOpt TriplePattern )*
ConstraintClause ::= <SUCHTHAT> Expression
               ( ( <COMMA> | <SUCHTHAT> ) Expression )*
TriplePattern  ::= <LPAREN> VarOrURI CommaOpt VarOrURI CommaOpt VarOrConst
               <RPAREN>
VarOrURI       ::= Var
```

```
              | URI
VarOrConst    ::= Var
              | Const
Var           ::= "?" Identifier
PrefixesClause::= <PREFIXES> PrefixDecl ( CommaOpt PrefixDecl )*
PrefixDecl    ::= Identifier <FOR> <QuotedURI>
Expression    ::= ConditionalOrExpression
ConditionalOrExpression ::= ConditionalAndExpression
              ( <SC_OR> ConditionalAndExpression )*
ConditionalAndExpression ::= StringEqualityExpression
              ( <SC_AND> StringEqualityExpression )*
              StringEqualityExpression ::= ArithmeticCondition ( <STR_EQ>
              ArithmeticCondition | <STR_NE> ArithmeticCondition |
              <STR_MATCH> PatternLiteral | <STR_NMATCH> PatternLiteral )*
ArithmeticCondition ::= EqualityExpression
EqualityExpression ::= RelationalExpression
              ( <EQ> RelationalExpression | <NEQ> RelationalExpression )?
RelationalExpression ::= AdditiveExpression ( <LT> AdditiveExpression |
              <GT> AdditiveExpression | <LE> AdditiveExpression |
              <GE> AdditiveExpression )?
AdditiveExpression ::= MultiplicativeExpression (
              <PLUS> MultiplicativeExpression |
              <MINUS> MultiplicativeExpression )*
MultiplicativeExpression ::=  UnaryExpression ( <STAR> UnaryExpression |
              <SLASH> UnaryExpression | <REM> UnaryExpression )*
UnaryExpression ::= UnaryExpressionNotPlusMinus
               | ( <PLUS> UnaryExpression | <MINUS> UnaryExpression )
UnaryExpressionNotPlusMinus ::= ( <TILDE> | <BANG> ) UnaryExpression
               | PrimaryExpression
PrimaryExpression ::= Var
               | Const
               | <LPAREN> Expression <RPAREN>
Const         ::= URI
               | NumericLiteral
               | TextLiteral
               | BooleanLiteral
               | NullLiteral
NumericLiteral ::= ( <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> )
TextLiteral   ::= ( <STRING_LITERAL1> | <STRING_LITERAL2> ) (
              <AT> Identifier )? ( <DATATYPE> URI )?
PatternLiteral ::=
BooleanLiteral ::= <BOOLEAN_LITERAL>
NullLiteral   ::= <NULL_LITERAL>
URI           ::= <QuotedURI>
               | QName
QName         ::= <NSPrefix> ':' (<LocalPart>)?
Unlilke XML Namespaces, the local part is optional
Identifier    ::= ( <IDENTIFIER> | <SELECT> | <SOURCE> | <FROM> | <WHERE>
              | <PREFIXES> | <FOR> | <STR_EQ> | <STR_NE> )
```

Figure 13 BNF grammar for RDQL

2.2.3.SPARQL

SPARQL (pronounced "sparkle") is an RDF query language. The name is a recursive acronym, which stands for SPARQL Protocol and RDF Query Language. SPARQL is being designed and standardized by the RDF Data Access Working

Group (DAWG) of the World Wide Web Consortium (Prud'hommeaux and Seaborne 2007). A SPARQL query example is provided below, in Figure 14:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX human: <http://www.inria.fr/2006/12/05/humans.rdfs#>

SELECT
    ?name ?spousename

WHERE {
    ?person human:name ?name.
    ?person human:hasSpouse ?spouse.
    ?spouse human:name ?spousename.
    ?person rdf:type human:Woman.

FILTER
    (?name = “Theo”)
}
```

Figure 14 An example of a SPARQL query

A SPARQL query contains a set of triple patterns forming a basic graph pattern. These triple patterns are similar to RDF triples, except that the subject, predicate, object or any combination thereof may be a variable. This basic graph pattern matches to a sub graph of the RDF data. Any of the RDF terms from that sub graph may be substituted for the variables and the result is an RDF graph equivalent to the sub graph. The main features of SPARQL are:

- ability to express queries across diverse data sources
- capable of querying required and optional graph patterns
- can query graph pattern conjunctions and disjunctions
- supports extensible value testing and constraining queries by source RDF graph
- SPARQL query results can be results sets or RDF graphs

The BNF grammar for SPARQL is provided below in Figure 15.

```
Query          ::= Prologue
               ( SelectQuery | ConstructQuery | DescribeQuery | AskQuery )
Prologue       ::= BaseDecl? PrefixDecl*
BaseDecl       ::= 'BASE' IRI_REF
PrefixDecl     ::= 'PREFIX' PNAME_NS IRI_REF
SelectQuery    ::= 'SELECT' ( 'DISTINCT' | 'REDUCED' )? ( Var+ | '*' )
               DatasetClause* WhereClause SolutionModifier
ConstructQuery::= 'CONSTRUCT' ConstructTemplate DatasetClause*
               WhereClause SolutionModifier
DescribeQuery ::= 'DESCRIBE' ( VarOrIRIref+ | '*' ) DatasetClause*
               WhereClause? SolutionModifier
AskQuery       ::= 'ASK' DatasetClause* WhereClause
DatasetClause ::= 'FROM' ( DefaultGraphClause | NamedGraphClause )
DefaultGraphClause ::= SourceSelector
NamedGraphClause ::= 'NAMED' SourceSelector
SourceSelector ::= IRIref
WhereClause    ::= 'WHERE'? GroupGraphPattern
SolutionModifier ::= OrderClause? LimitOffsetClauses?
LimitOffsetClauses ::= ( LimitClause OffsetClause? | OffsetClause
               LimitClause? )
OrderClause    ::= 'ORDER' 'BY' OrderCondition+
OrderCondition ::= ( ( 'ASC' | 'DESC' ) BrackettedExpression )
               | ( Constraint | Var )
LimitClause    ::= 'LIMIT' INTEGER
OffsetClause   ::= 'OFFSET' INTEGER
GroupGraphPattern ::= '{' TriplesBlock? ( ( GraphPatternNotTriples
               | Filter ) '.'? TriplesBlock? )* '}'
TriplesBlock   ::= TriplesSameSubject ( '.' TriplesBlock? )?
GraphPatternNotTriples ::= OptionalGraphPattern |
               GroupOrUnionGraphPattern | GraphGraphPattern
OptionalGraphPattern ::= 'OPTIONAL' GroupGraphPattern
GraphGraphPattern ::= 'GRAPH' VarOrIRIref GroupGraphPattern
GroupOrUnionGraphPattern ::= GroupGraphPattern ( 'UNION'
               GroupGraphPattern )*
Filter         ::= 'FILTER' Constraint
Constraint     ::= BrackettedExpression | BuiltInCall | FunctionCall
FunctionCall   ::= IRIref ArgList
ArgList        ::= ( NIL | '(' Expression ( ',' Expression )* ')' )
ConstructTemplate ::= '{' ConstructTriples? '}'
ConstructTriples ::= TriplesSameSubject ( '.' ConstructTriples? )?
TriplesSameSubject ::= VarOrTerm PropertyListNotEmpty |
               TriplesNode PropertyList
PropertyListNotEmpty ::= Verb ObjectList ( ';' ( Verb ObjectList )? )*
PropertyList   ::= PropertyListNotEmpty?
ObjectList     ::= Object ( ',' Object )*
Object         ::= GraphNode
Verb           ::= VarOrIRIref | 'a'
TriplesNode    ::= Collection | BlankNodePropertyList
BlankNodePropertyList ::= '[' PropertyListNotEmpty ']'
Collection     ::= '(' GraphNode+ ')'
GraphNode      ::= VarOrTerm | TriplesNode
VarOrTerm      ::= Var | GraphTerm
VarOrIRIref    ::= Var | IRIref
Var            ::= VAR1 | VAR2
GraphTerm      ::= IRIref | RDFLiteral | NumericLiteral | BooleanLiteral |
               BlankNode | NIL
Expression     ::= ConditionalOrExpression
ConditionalOrExpression ::= ConditionalAndExpression ( '||'
               ConditionalAndExpression )*
ConditionalAndExpression ::= ValueLogical ( '&&' ValueLogical )*
```

```
ValueLogical  ::= RelationalExpression
RelationalExpression ::= NumericExpression ( '=' NumericExpression |
              '!=' NumericExpression | '<' NumericExpression |
              '>' NumericExpression | '<=' NumericExpression |
              '>=' NumericExpression )?
NumericExpression ::= AdditiveExpression
AdditiveExpression ::= MultiplicativeExpression (
              '+' MultiplicativeExpression | '-' MultiplicativeExpression
              | NumericLiteralPositive | NumericLiteralNegative )*
MultiplicativeExpression ::= UnaryExpression ( '*' UnaryExpression
              | '/' UnaryExpression )*
UnaryExpression ::= '!' PrimaryExpression | '+' PrimaryExpression
              | '-' PrimaryExpression | PrimaryExpression
PrimaryExpression ::= BrackettedExpression | BuiltInCall
              | IRIrefOrFunction | RDFLiteral | NumericLiteral
              | BooleanLiteral | Var
BrackettedExpression ::= '(' Expression ')'
BuiltInCall   ::= 'STR' '(' Expression ')'
              | 'LANG' '(' Expression ')'
              | 'LANGMATCHES' '(' Expression ',' Expression ')'
              | 'DATATYPE' '(' Expression ')' | 'BOUND' '(' Var ')'
              | 'sameTerm' '(' Expression ',' Expression ')'
              | 'isIRI' '(' Expression ')' | 'isURI' '(' Expression ')'
              | 'isBLANK' '(' Expression ')'
              | 'isLITERAL' '(' Expression ')'
              | RegexExpression
RegexExpression ::= 'REGEX' '(' Expression ','
              Expression ( ',' Expression )? ')'
IRIrefOrFunction ::= IRIref ArgList?
RDFLiteral    ::= String ( LANGTAG | ( '^^' IRIref ) )?
NumericLiteral ::= NumericLiteralUnsigned |
              NumericLiteralPositive | NumericLiteralNegative
NumericLiteralUnsigned ::= INTEGER | DECIMAL | DOUBLE
NumericLiteralPositive ::= INTEGER_POSITIVE
              | DECIMAL_POSITIVE | DOUBLE_POSITIVE
NumericLiteralNegative ::= INTEGER_NEGATIVE
              | DECIMAL_NEGATIVE | DOUBLE_NEGATIVE
BooleanLiteral ::= 'true' | 'false'
String        ::= STRING_LITERAL1 | STRING_LITERAL2
              | STRING_LITERAL_LONG1 | STRING_LITERAL_LONG2
IRIref        ::= IRI_REF | PrefixedName
PrefixedName  ::= PNAME_LN | PNAME_NS
BlankNode     ::= BLANK_NODE_LABEL | ANON

@terminals
IRI_REF       ::= '<' ([^<>\"{}|^`\\]-[#x00-#x20])* '>'
PNAME_NS      ::= PN_PREFIX? ':'
PNAME_LN      ::= PNAME_NS PN_LOCAL
BLANK_NODE_LABEL ::= '_:' PN_LOCAL
VAR1          ::= '?' VARNAME
VAR2          ::= '$' VARNAME
LANGTAG       ::= '@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
INTEGER       ::= [0-9]+
DECIMAL       ::= [0-9]+ '.' [0-9]* | '.' [0-9]+
DOUBLE        ::= [0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT
              | ([0-9])+ EXPONENT
INTEGER_POSITIVE ::= '+' INTEGER
DECIMAL_POSITIVE ::= '+' DECIMAL
DOUBLE_POSITIVE ::= '+' DOUBLE
INTEGER_NEGATIVE ::= '-' INTEGER
DECIMAL_NEGATIVE ::= '-' DECIMAL
```

```
DOUBLE_NEGATIVE ::= '-' DOUBLE
EXPONENT       ::= [eE] [+-]? [0-9]+
STRING_LITERAL1 ::= "'" ( ([^#x27#x5C#xA#xD]) | ECHAR )* "'"
STRING_LITERAL2 ::= '"' ( ([^#x22#x5C#xA#xD]) | ECHAR )* '"'
STRING_LITERAL_LONG1 ::= "'''" ( ( "'" | "''" )?
               ( [^'\\] | ECHAR ) )* "'''"
STRING_LITERAL_LONG2 ::= '"""' ( ( '"' | '""' )?
               ( [^"\\] | ECHAR ) )* '"""'
ECHAR          ::= '\\' [tbnrf\\"']
NIL            ::= '(' WS* ')'
WS             ::= #x20 | #x9 | #xD | #xA
ANON           ::= '[' WS* ']'
PN_CHARS_BASE  ::= [A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6]
               | [#x00F8-#x02FF] | [#x0370-#x037D] | [#x037F-#x1FFF]
               | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF]
               | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD]
               | [#x10000-#xEFFFF]
PN_CHARS_U     ::= PN_CHARS_BASE | '_'
VARNAME        ::= ( PN_CHARS_U | [0-9] ) ( PN_CHARS_U | [0-9] | #x00B7
               | [#x0300-#x036F] | [#x203F-#x2040] )*
PN_CHARS       ::= PN_CHARS_U | '-' | [0-9] | #x00B7 | [#x0300-#x036F]
               | [#x203F-#x2040]
PN_PREFIX      ::= PN_CHARS_BASE ((PN_CHARS|'.')* PN_CHARS)?
PN_LOCAL       ::= ( PN_CHARS_U | [0-9] ) ((PN_CHARS|'.')* PN_CHARS)?

@pass: [ \t\r\n]+ | '#' [^\r\n]*
```

Figure 15 Full BNF grammar for SPARQL

2.3. SUMMARY

In this chapter, we have introduced the languages used for the purposes of querying RDF data sources. We have made explicit the syntax to which these languages adhere and have provided some history behind these languages.

CHAPTER 3: SEMANTOS OVERVIEW

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom." -- Isaac Asimov

For any query or access language to be an effective and efficient semantic language it must be equally capable of interrogating raw data, meta-data or even meta-meta-data. The language syntax itself must therefore support the possibility of querying multiple levels of data encapsulation. Foremost, Semantos must be a semantic language and as such be capable of interacting with data at various meta levels and viewing data as a graph of connected nodes.

3.1. A SEMANTIC LANGUAGE

In the process of querying data, it is possible to view these data source at three levels of abstraction. This is especially true of RDF data sources. The three levels are (Broekstra, Kampman and van Harmelen 2001):

- The syntactic level (raw data level)
- The structure level (subject-predicate-object triplets)
- The semantic level (1 or more graphs with predefined semantics)

3.1.1. THE SYNTACTIC LEVEL

An RDF data source/model can be represented as a simple XML document (Brickley and Guha 2000). It is then possible to query this XML document with any of the available XML query language, e.g. XQuery. This is, however, not an adequate solution, as much of the inherent information in an RDF model is not apparent from its hierarchical structure, but is rather derived from the relationships established in the ontology. Any language querying at this level would be dependent on the structure of the XML. In the case of RDF this structure is flexible, making it nearly impossible to query the syntactic level of disparate data sources.

3.1.2. THE STRUCTURE LEVEL

Any RDF data source represents a set of triples, with each triple representing a statement of the form Subject-Predicate-Object. Querying at the structure level has a clear advantage over that of the syntactic level, as it is independent of the underlying XML structure which has been chosen to represent the RDF data source. At this level, the query directly interprets the RDF model. The problem with any structure level query language, however, is that it only interprets explicitly defined triplets. It does not take inferred triplets into account.

3.1.3. THE SEMANTIC LEVEL

Query languages operating at the semantic level are superior to other query languages, in that they are capable of interpreting inferred triplets in data sources, if provided with appropriate ontologies. It is at this level that Semantos queries data sources.

3.2. SYNTAX

The syntax of Semantos can be formalized as a XML schema. However it would be useful, for the sake of brevity, to explain the syntax and structure of the language, by means of an example first. This example is provided below in Figure 16.

```
<semantos:fetch>

 <semantos:source
  name="rdf source"
  uri="http://www.retrorabbit.co.za/data/example1.rdf"/>
```

```
<semantos:ontology
 name="rdfs source"
 uri="http://www.retrorabbit.co.za/data/ontology1.rdfs"/>

<semantos:namespace
 name="rdf"
 value="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />

<semantos:namespace
 name="people"
 value="http://www.retrorabbit.co.za/data/humans.rdfs#" />

<semantos:entity
 name="firstnames">

 <semantos:attribute
  name="name"/>

 <semantos:attribute
  name="spousename"/>

 <semantos:graph>

  <semantos:triple
   subject="person"
   predicate="human:name"
   object="name"/>

  <semantos:triple
   subject="person"
   predicate="rdf:type"
   object="human:Woman"/>

  <semantos:triple
   subject="person"
   predicate="human:hasSpouse"
   object="spouse"/>

  <semantos:triple
   subject="spouse"
   predicate="human:name"
```

```
      object="spousename"/>

    </semantos:graph>

    <semantos:filter>

      <semantos:condition
        attribute="name"
        operator=="eq"
        value=="Theo"

    </semantos:filter>

  </semantos:entity>

</semantos:fetch>
```

Figure 16 An example of a Semantos query

The first thing to note is that Semantos uses the "semantos" namespace. This is to identify Semantos queries when embedded in other XML documents. The syntax or structure of a query itself is quite simple; each element type found in the query above, is addressed below.

3.2.1. FETCH <SEMANTOS:FETCH>

The outer most element of any Semantos query is the fetch element, as seen below in Figure 17. It contains all the necessary ingredients to process a query and format the results. A fetch element may be embedded inside another XML document (i.e. an XHTML page) in order to stream the results directly into the document's format. This supports the required characteristic identified in (Bonifati and Ceri 2000). It should be noted that XQuery also supports the idea of embedding a query (World Wide Web Consortium 2007).

```
<semantos:fetch>…</semantos:fetch>
```

Figure 17 The Semantos fetch element

3.2.2. SOURCE <SEMANTOS:SOURCE>

Each fetch element requires at least one source element as illustrated below in Figure 18. The source element identifies the source of the data, and, for integration purposes, may declare multiple data sources. Currently, Semantos requires that these data sources be RDF compatible, meaning that either the data source must be an actual RDF document or the data source should be exposed as an RDF data source. This is simply a limitation of the implementation at this stage, as any graph representative markup can potentially be used.

```
<semantos:source
   name="rdf source"
   uri="http://www.retrorabbit.co.za/data/example1.rdf" />
```

Figure 18 The Semantos source element

3.2.3. ONTOLOGY <SEMANTOS:ONTOLOGY>

The ontology tag is probably one of the more important elements in a Semantos query as shown in Figure 19; it indicates the existence of associated ontologies. Each fetch element may contain zero or more ontology elements, which makes it possible to do ontology mappings, using multiple ontologies or to query the data source as is, without any semantic representation/inference. Semantos currently requires these ontologies to be RDFS documents, but this is only a limitation of this specific implementation, and any ontology format should be workable.

```
<semantos:ontology
     name="onto"
     uri="http://www.retrorabbit.co.za/semantos/people.rdfs" />
```

Figure 19 The Semantos ontology element

3.2.4. NAMESPACE <SEMANTOS:NAMESPACE>

In order to circumvent the necessity of providing long fully qualified names for the identification of node elements, Semantos supports namespaces, as declared with the namespace tag shown in Figure 20. Each fetch element may have zero or more namespaces. These namespace declarations are later used in the element tags, to simplify node identification.

```
    <semantos:namespace
         name="rdf"
         value="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
```

Figure 20 The Semantos namespace element

3.2.5. ENTITY <SEMANTOS:ENTITY>

The result of a Semantos query is an entity list. An entity is a logical processing unit, similar to a class structure in object oriented programming. An entity possesses attributes. Each fetch element has exactly one entity element, shown in Figure 21, which helps to define the structure of the results. The entity element also allows the query results to be shaped by adding non Semantos XML tags in the body of the entity element. These tags will then be repeated for each returned result.

```
<semantos:entity
        name="a">
        …
</semantos:entity>
```

Figure 21 The Semantos entity element

3.2.6. ATTRIBUTE <SEMANTOS:ATTRIBUTE>

In order to fully structure the data response, the result entity requires attributes. Each entity element may have one or more attribute elements, shown in Figure 22, each of which maps to a variable node in the query graph. The attribute elements give structure to the data projection.

```
<semantos:attribute
    name="name"/>
```

Figure 22 The Semantos attribute element

3.2.7. GRAPH <SEMANTOS:GRAPH>

Creating a semantic query requires the creation of a context graph. The graph identifies the structure of the data being queried. Each entity node must have exactly one graph element as shown in Figure 23. A graph is a set of triples (Gutierrez, Hurtado and Mendelzon 2004), which is responsible for mapping the tree/graph structure of information into a structured tuple data set, which can be used in this specific result set.

```
<semantos:graph>
    …
</semantos:graph>
```

Figure 23 The Semantos graph element

3.2.8. TRIPLE <SEMANTOS:TRIPLE>

The triple tag is used to create the triples that build the context graph. A graph element may contain one or more triple elements. The triple element has three attributes matching either the subject, predicate or object section as shown in Figure 24. The values of these properties are fully qualified XML element names. If a namespace element is declared in the query, it is possible to write out the full node name via a shortcut format, i.e. "human:name" would translate to "http://www.retrorabbit.co.za/human#name". It is also possible to assign any of these three properties as a placeholder or variable node, meaning that any possible node in any possible graph may match to the node. A variable node is indicated by prefixing the name of the attribute with the "?" symbol.

```
<semantos:triple
     subject="person"
     predicate="human:name"
     object="?name"/>
```

Figure 24 The Semantos triple element

3.2.9. FILTER <SEMANTOS:FILTER>

The entity element may contain zero or one filter elements as shown in Figure 25. A filter is used to further sieve out the tuples that are returned from the graph processing. Nested filters may be encountered. A filter may either be an "and" filter or an "or" filter. The type is applied on the nested filters and conditions contained within a filter element.

```
<semantos:filter
     type="and">
     …
</ semantos:filter>
```

Figure 25 The Semantos filter element

3.2.10. CONDITION <SEMANTOS:CONDITION>

The condition element has "attribute", "operator" and "value" properties as shown in Figure 26 below. These are used to build conditions for the filter. The "attribute" property represents one of the query attributes, as specified by the attribute elements.

```
<semantos:condition
     attribute="human:name"
     operator="eq"
     value="theo"/>
```

Figure 26 The Semantos condition element

The operator indicates what type of condition is to be imposed. Possible values include:

- eq: Equals
- lt: Less than
- gt: Greater than
- ge: Greater than, or equals
- le: Less than, or equals
- ne: Not equal
- null: Null or empty
- not-null: Not null or not empty
- like: Like
- not-like: Not like
- in: In
- not-in: Not in

For the operators “in” and “not-in” it is possible to specify a list of values, this syntax is shown in Figure 27.

```
<semantos:condition
     attribute="human:name"
     operator="in">
            <value>Debbie-Lee</value>
            <value>Christine</value>
            <value>Connie</value>
     </semantos:condition>
```

Figure 27 The Semantos condition element, with a list

3.3. SUMMARY

In this chapter we have established the level at which a Semantos query interrogates a data source. We have also provided the syntax of the Semantos language by means of an example, illustrating each aspect of the language structure. The syntax of Semantos as established in this chapter is used as the base building block for the engineering of the software in later chapters. The software must adhere strictly to the requirements set out in chapter 1 as well as the syntactic constructs laid out in chapter 3.

CHAPTER 4: DESIGN AND IMPLEMENTATION

"Each problem that I solved became a rule, which served afterwards to solve other problems." -- Rene Descartes

Armed with the requirements and language specification from previous chapters, this chapter embarks on the design and implementation of a Semantos query processing engine. The framework presented here provides a working test case for the Semantos query language structure and will be tested against the language specification as set out in chapter 1. The major goal of this chapter is to establish a working version of the Semantos framework and to further illustrate the workings of Semantos specification. With regards to the code snippets provided in this chapter, the new C# language features, as they appear in the code, will also be investigated. This is to aid the reader's digestion of the code.

4.1. LANGUAGE INTEGRATED QUERY (LINQ)

Semantos makes use of the Language Integrated Query or LINQ technology, which arrives on the scene along with .C# 3.0 and VB.NET 9.0 (Meijer, Torgersen and Bierman 2007). The main philosophy around LINQ is the integration it provides between object, relational and semi structured data models (Meijer, Schulte and Bierman 2003). It achieves this by way of generalization, rather than by ad-hoc specializations. In particular, extensive use is made of Xlinq, provided by the System.XML.Xlinq namespace. Xlinq strives to make XML documents or document fragments first class citizens, meaning XML values can be constructed, loaded, passed, transformed and updated in a type-safe manner (Meijer, Schulte and Bierman 2003). As the example below illustrates, the heavy use of the DOM object has been removed (Meijer, Beckman and Bierman 2006), as can be seen in Figure 28, and has freed up XML construction, navigation and querying drastically, with regards to code efficiency, execution speed and memory requirements (Microsoft 2006).

```
XElement contacts = new XElement("contacts",
    from c in customers
    where c.Country == "USA"
    select new XElement("contact",
        new XElement("name", c.CompanyName),
        new XElement("phone", c.Phone)
    )
);
```

Figure 28 XLinq without the DOM object

Another benefit which may be obtained by using LINQ, stems from LINQ's origins in the experimental programming language Cω (C Omega). One of the original design goals for Cω, was to evolve C# in such a way that it provides an integration of the object, relational and semi-structured data models (Bierman, Meijer and Schulte 2005). As one of the primary purposes of Semantos is to be an Information Integration query language, it would be appropriate to leverage the integration features provided by the LINQ extensions. This would allow access to all three of the mentioned data models: relational, hierarchical and graphical.

The use of XLinq may further be justified by the fact that the programmatic construction of Semantos queries is far more robust and concise (see Figure 29) when using the "functional construction" methodologies introduced with .NET 3.5.

```
XDocument query = new XDocument(
 new XDeclaration("1.0", null, null),
 new XElement(semantos + "fetch",
  new XElement(semantos + "source",
   new XAttribute(semantos + "name", "WhoRU Full User Profile"),
   new XAttribute(semantos + "uri",
    @"http://localhost/whoru/fullprofile.aspx")),
   new XElement(semantos + "entity",
    new XAttribute(semantos + "name", "knownpeople"),
    new XElement(semantos + "attribute",
     new XAttribute(semantos + "name", "@name"))),
   new XElement(semantos + "graph",
    new XElement(semantos + "triple",
     new XAttribute(semantos + "subject", whoru + "person"),
     new XAttribute(semantos + "predicate", whoru + "myname"),
     new XAttribute(semantos + "object", "@name"));
```

Figure 29 Programmatically creating a Semantos query in LINQ.

4.2. Processing Algorithm

The core Semantos query processing algorithm is provided below. This algorithm does not make provision for ontology processing, so no logical reasoning is required on a semantic level. It is convenient to think of the algorithm as executing in three different phases: a preparation, a processing and a cleanup phase. The preparation phase interrogates the Semantos query to retrieve all the information required for the execution of the query. The processing phase is tasked with looping through all the triple pairs discovered in the query by the preparatory phase and to query the information sources regarding the triple pairs. The cleanup phase collates all the retrieved information into a single coherent result, which may be a data table or XML document to be returned to the requesting process. Figure 30 below provides a graphical representation of the algorithm in full. All the code provided in this chapter compiles in C#, using the Microsoft .Net 3.5 framework. Short descriptions are also provided to aid the reader in understanding the code snippets.

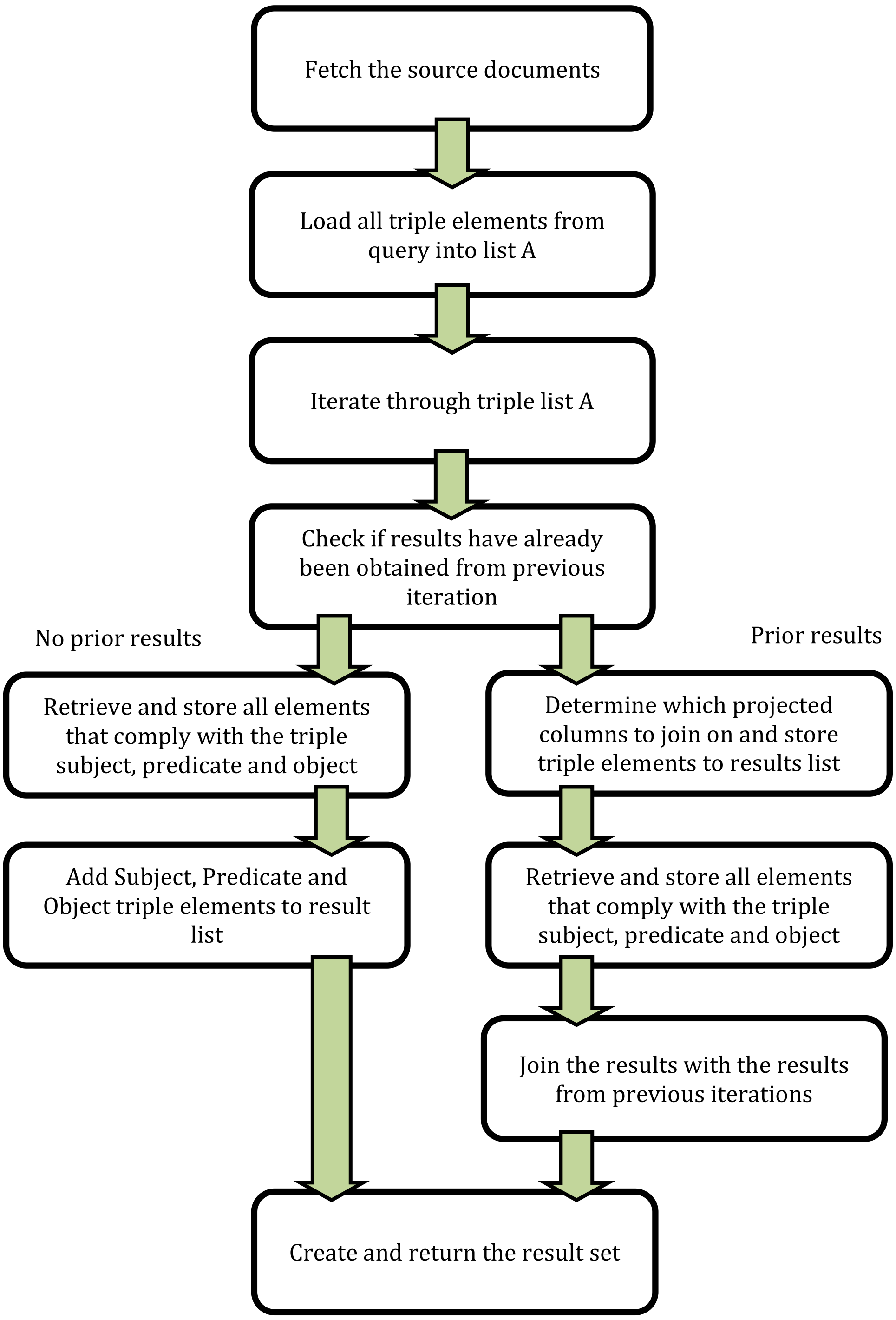

Fetch the source documents
Load all triple elements from query into list A
Iterate through triple list A
Check if results have already been obtained from previous iteration
No prior results
Prior results
Retrieve and store all elements that comply with the triple subject, predicate and object
Determine which projected columns to join on and store triple elements to results list
Add Subject, Predicate and Object triple elements to result list
Retrieve and store all elements that comply with the triple subject, predicate and object
Join the results with the results from previous iterations
Create and return the result set

Figure 30 Simple query processing algorithm

4.2.1. Preparation Phase

As stated previously, the preparation phase interrogates the Semantos query, in order to retrieve all the information required for the execution of the query. The first step is to fetch the RDF source documents and collate them into a single XML document to simplify processing. The code snippet below makes use of several new C# language features, including an implicitly typed local variable declaration, where the type of the variable is inferred. The second language addition which is made use of is the query expression. Query expressions are language integrated syntax for queries. This syntax is similar to other query languages like SQL (relational) and XQuery (hierarchical). The final new language feature that is apparent in this code snippet is the appearance of the XDocument class, which comes from the new System.Xml.Linq namespace, as can be found in Figure 31.

```
// Fetch the source documents
var rdf_sources =
  from s in query.Descendants(semantos+"source")
  select (string)s.Attribute(semantos+"uri");


// Collate source documents into single XML document
XDocument source = new XDocument(new XElement("rdfsources"));

foreach (string s in rdf_sources)
  source.Add(XDocument.Load(s));
```

Figure 31 Fetch and collate source documents

The next step is to retrieve all the triple elements from the Semantos query and store them in an enumerable list of triple objects. The code snippet below uses a new C# language feature, called object initializer; which reduces the instantiation of the Triple object to a single line of code, shown in Figure 32. Another XLinq feature that is shown in this snippet is the XNamespace object which is instantiated by via the syntax semantos + “triple”, where semantos is an XNamespace object that resolves to the URI: “http://www.retrorabbit.co.za/semantos#”. This allows us to fully qualify all the XML elements with namespaces, without a great deal of trouble.

```
// Load all triple elements from query into list A
IEnumerable<Triple> triple_list =
   (from s in query.Descendants(semantos+"triple")
   select new Triple {
   Subject = ((string)s.Attribute(semantos+"subject")).ToString(),
   Predicate = ((string)s.Attribute(semantos+"predicate")).ToString(),
   Object = (string)s.Attribute(semantos+"object")
   }).ToList<Triple>();
```

Figure 32 Load all triple elements

4.2.2. PROCESSING PHASE

The processing phase is tasked with looping through all the triple pairs discovered in the query by the preparatory phase and to query the information sources regarding the triple pairs. This is achieved by searching through the information source for any XML element matches to the triple. In other words, an XML element with the name specified in the subject part of the triple is searched for; which also has a direct descendant or child element with the name specified in the predicate part of the triple. Thereafter, it is confirmed that the value of the elements matching to the predicate part of the triple is equal to the value specified in the triple object part; or, if the object part is a value holder (in other words it starts with an “?” symbol), allowance is made for any XML value in the predicate element.

```
QueryResult queryResult = new QueryResult();

// Loop through triple list A
foreach (Triple triple in triple_list)
{
   // Have results already been obtained from previous iteration?
   if (queryResult.ProjectedColumnValues == null)
   {
      ProcessFirstTriple(queryResult, source, triple);
   }
   else
   {
      ProcessNonFirstTriple(queryResult, source, triple);
   }
}
```

Figure 33 Loop through triple lists

The processing of the triple varies, depending on whether or not it is the first triple, and if results have already been obtained, as can be seen from Figure 33. If results have already been obtained, then a join operation must also be executed. The method for executing the first triple is simpler, and is provided below in Figure 34:

```
private void ProcessFirstTriple(
   QueryResult queryResult,
   XDocument source,
   Triple triple)
{
   // Retrieve and store all elements that comply with the triple
   // subject, predicate and object restrictions
   queryResult.ProjectedColumnValues =
       (from s in source.Descendants(triple.Subject)
        from p in s.Elements(triple.Predicate)
        where ((string)p == triple.Object) ||
            triple.Object.StartsWith("?")
       select new ElementList(s, p, p.FirstNode)).ToList<ElementList>();

   // Add Subject, Predicate and Object triple elements to result list
   queryResult.ProjectedColumnNames = new List<string>();
   queryResult.ProjectedColumnNames.Add(triple.Subject);
   queryResult.ProjectedColumnNames.Add(triple.Predicate);
   queryResult.ProjectedColumnNames.Add(triple.Object);
}
```

Figure 34 Processing of the first triple

The more complex method for processing triple pairs that need to be joined (all but the first) is provided below, in Figure 35. This method differs from the previous, with regard to the joining discovery and processing. The first task this method executes is to discover the two projected columns that need to be joined. This is achieved by matching all the exiting projected column names that come from the Semantos triple expressions to the subject, predicate and object parts of the current triple being queried. Once a match is found, the method searches for all triple matches in the manner described above, but with the addition of also simultaneously joining the new result set with the previous result set.

```
private void ProcessNonFirstTriple(
   QueryResult queryResult,
   XDocument source,
   Triple triple)
{

   int leftsidepcolumn = 0;
   int rightsidepcolumn = 0;
   int rightsidepcolumn_include1 = 0;
   int rightsidepcolumn_include2 = 0;

   // Determine which projected columns to join on and store
   // triple elements to results list
   for (int i = 0; i < queryResult.ProjectedColumnNames.Count; i++)
   {
       if (queryResult.ProjectedColumnNames[i] == triple.Subject)
       {
           leftsidepcolumn = i;
           rightsidepcolumn = 0;
           rightsidepcolumn_include1 = 1;
           rightsidepcolumn_include2 = 2;
           queryResult.ProjectedColumnNames.Add(triple.Predicate);
           queryResult.ProjectedColumnNames.Add(triple.Object);
           break;
       }
       else if (queryResult.ProjectedColumnNames[i] == triple.Predicate)
       {
           leftsidepcolumn = i;
           rightsidepcolumn = 1;
```

```
                rightsidepcolumn_include1 = 0;
                rightsidepcolumn_include2 = 2;
                queryResult.ProjectedColumnNames.Add(triple.Subject);
                queryResult.ProjectedColumnNames.Add(triple.Object);
                break;
            }
            else if (queryResult.ProjectedColumnNames[i] == triple.Object)
            {
                leftsidepcolumn = i;
                rightsidepcolumn = 2;
                rightsidepcolumn_include1 = 0;
                rightsidepcolumn_include2 = 1;
                queryResult.ProjectedColumnNames.Add(triple.Subject);
                queryResult.ProjectedColumnNames.Add(triple.Predicate);
                break;
            }
        }

        IEnumerable<XElement> source_descendants =
            (triple.Subject.StartsWith("@")) ?
            source.Descendants() : source.Descendants(triple.Subject);

        // Retrieve and store all elements that comply with the
        // triple subject, predicate and object restrictions and
        // join the results with the results from previous iterations
        queryResult.ProjectedColumnValues =
            (from r in
                (from r in queryResult.ProjectedColumnValues select r)
            join q in
                (from s in source_descendants
                from t in s.Elements(triple.Predicate)
                select new ElementList(s, t, t.FirstNode))
                on r.List[leftsidepcolumn] equals q.List[rightsidepcolumn]
                select
                r.Append(q.List[rightsidepcolumn_include1])
                .Append(q.List[rightsidepcolumn_include2]))
                .ToList<ElementList>();
}
```

Figure 35 Processing of subsequent triples

4.2.3. CLEANUP PHASE

The cleanup phase collates all the retrieved information into a single coherent result, which may be a data table or XML document which is to be returned to the requesting process. The method that constructs a DataTable object from the query results, is shown below in Figure 36. This is achieved by first adding a new data column to an empty data table for each projected column from the query results. After the data table structure is defined, a new data row is added to the data table for each of the value tuples retrieved from the query.

```
private DataTable BuildResultTable(
  XDocument query,
  QueryResult
  queryResult)
{
  DataTable tblTemp = new DataTable();

  foreach (string attribute in
    from s in query.Descendants(semantos + "attribute")
    select (string)s.Attribute(semantos + "name"))
    {
      tblTemp.Columns.Add(attribute, typeof(string));
    }

  for (int i = 0; i < queryResult.ProjectedColumnValues.Count; i++)
  {
    DataRow newrow = tblTemp.NewRow();
    foreach (DataColumn column in tblTemp.Columns)
    {
      newrow[column] = queryResult[i, column.ColumnName];
    }
    tblTemp.Rows.Add(newrow);
  }
  return tblTemp;
}
```

Figure 36 Constructing the results table

The last bit of code to be shown, is the supportive data structures that have been used in the code snippets above These structures are shown below in Figure 37. The first class is a simple storage container used to programmatically store the triple set retrieved from the Semantos query in memory. The second class stores the results of a single tuple or result set. For example if one were to query an employee database for people, each person retrieved would be stored as a single ElementList object. The final class contains the query results. The projected column names list stores the names of the columns as they are added after each iteration of the main processing loop. The projected column value list stores the values of the returned tuples or value sets from each iteration of the loop. In terms of the employee database example above, this would translate into the list of all retrieved employees.

```
public class Triple
{
   public string Subject { get; set; }
   public string Predicate { get; set; }
   public string Object { get; set; }
}

public class ElementList
{
   public List<XNode> List { get; set; }

   public ElementList(XNode columnNode1, XNode columnNode2,
      XNode columnNode3)
   {
      List = new List<XNode>();
      List.Add(columnNode1);
      List.Add(columnNode2);
      List.Add(columnNode3);
   }

   public ElementList Append(XNode columnNode)
   {
      List.Add(columnNode);
      return this;
```

```
    }
}

public class QueryResult
{
    public List<string> ProjectedColumnNames { get; set; }
    public List<ElementList> ProjectedColumnValues { get; set; }

    public QueryResult()
    {
        ProjectedColumnNames = null;
        ProjectedColumnValues = null;
    }

    public XNode this[int index, string name]
    {
        get
        {
            for (int i = 0; i < ProjectedColumnNames.Count; i++)
            {
                if (ProjectedColumnNames[i] == name)
                {
                    return ProjectedColumnValues[index].List[i];
                }
            }
            return null;
        }
    }
}
```

Figure 37 Supporting data structures

4.3. SUMMARY

This chapter established a working software version of the Semantos query language. The chapter also introduced a working design for the query processing pipeline, divided into a multi stage or phase process that supports the logical design of the software. Code examples were provided in C#, illustrating an implementation of the designed processing pipeline. These code examples also introduce LINQ technology features and how they can be used in Semantos processing engines, as made available with the latest version of Microsoft's .NET framework version 3.5. The software constructed in this chapter will be used as a benchmark in establishing whether or not the requirements from the evaluation criteria in chapter 1 can indeed be satisfied.

CHAPTER 5: EVALUATION

"The most important thing is not to stop questioning." -- Albert Einstein

Empowered with a working Semantos query processing engine from the previous chapter, this chapter evaluates the software and langauge syntax against the evaluation criteria established in chapter 1. Through testing and inspection this chapter will show that Semantos does indeed meet all the established criteria and expectations.

5.1. EVALUATION DATA SET

For the purpose of evaluation we will be making use of Open Directory RDF dumps (Open Directory RDF Dump 2004) to test Semantos queries against, unless stated otherwise. Open Directory is the largest internet index database maintained by real people. Open Directory makes two dumps available to the public, the first is the structure dump file, shown in Figure 38, which provides hierarchy information regarding the categories used in the second file. The second file contains all the indexed and categorized web links as shown in Figure 39.

```
<Topic r:id="Top/Arts/Movies">
  <catid>38</catid>
  <aolsearch>movies</aolsearch>
  <dispname>Movies</dispname>
  <d:Title>Movies</d:Title>
  <d:Description>This category is for information on anything about
  movies, the motion-picture medium, or the film industry, including
  actors, actresses, filmmakers, and individual films.  </d:Description>
  <altlang r:resource="Arabic:Top/World/Arabic/ترفيه/افلام"/>
  <altlang r:resource="Romanian:Top/World/Română/Artă/Cinema"/>
  <altlang r:resource="Slovak:Top/World/Slovensky/Umenie/Film"/>
  <altlang r:resource="Serbian:Top/World/Srpski/Umetnost/Film"/>
  <symbolic r:resource="DVD:Top/Arts/Movies/Home_Video/DVD"/>
  <symbolic r:resource="Music:Top/Arts/Music/Movies"/>
  <symbolic r:resource="People:Top/Arts/Movies/Filmmaking/People"/>
  <lastUpdate>2004-04-23 03:47:16</lastUpdate>
  <narrow r:resource="Top/Arts/Movies/Characters"/>
  <narrow r:resource="Top/Arts/Movies/News_and_Media"/>
  <narrow r:resource="Top/Arts/Movies/Filmmaking"/>
  <narrow r:resource="Top/Arts/Movies/Awards"/>
  <narrow r:resource="Top/Arts/Movies/Release_Schedules"/>
  <editor r:resource="jeffconn"/>
  <editor r:resource="rob13"/>
  <editor r:resource="jennyhorm"/>
  <newsGroup r:resource="news:rec.arts.movies.current-films"/>
  <newsGroup r:resource="news:rec.arts.movies.movie-going"/>
  <newsGroup r:resource="news:rec.arts.movies.past-films"/>
</Topic>

<Alias r:id="DVD:Top/Arts/Movies/Home_Video/DVD">
  <d:Title>DVD</d:Title>
  <Target r:resource="Top/Arts/Movies/Home_Video/DVD"/>
</Alias>
```

Figure 38 Excerpt from Open Directory structure dump file

```
<Topic r:id="Top/Arts/Movies/Titles/1/13th_Warrior,_The">
  <catid>54809</catid>
  <link r:resource="http://www.geocities.com/darkdaze18/"/>
  <link r:resource="http://www.boxofficemojo.com/13thwarrior.html"/>
  <link r:resource="http://ter.air0day.com/13thwarrior.shtml"/>
  <link r:resource="http://www.metacritic.com/video/titles/13thwarrior"/>
  <link r:resource="http://us.imdb.com/title/tt0120657/"/>
</Topic>

<ExternalPage about="http://www.geocities.com/darkdaze18/">
  <d:Title>The 13th Warrior Domain</d:Title>
  <d:Description>Fan site, includes a review, video and sound clips, and
  photographs.</d:Description>
  <topic>Top/Arts/Movies/Titles/1/13th_Warrior,_The</topic>
</ExternalPage>

<ExternalPage
about="http://apolloguide.com/mov_revtemp.asp?Title=13th+Warrior,+The">
  <d:Title>Apollo Leisure Guide</d:Title>
  <d:Description>Includes a review, cast list, and links.</d:Description>
  <topic>Top/Arts/Movies/Titles/1/13th_Warrior,_The</topic>
</ExternalPage>
```

Figure 39 Excerpt from Open Directory content dump file

5.2. XML REPRESENTATION

This evaluation criteria establishes that a Semantos query must be represented by valid XML. In other words, a Semantos query should be seen as a valid XML document conforming to a provided schema. Evaluation of this criteria will be accomplished by means of schema validation.

5.2.1. METHOD

In order to satisfy this requirement Semantos queries must adhere to a valid XML schema. To illustrate this assertion about Semantos we will construct several valid Semantos queries and XML schema validators will then be used to validate these queries against the Semantos schema provided in Appendix A. These queries will be validated by means of the publicly available schema validator which may be found at the URL "http://tools.decisionsoft.com/schemaValidate/". The schema validator allows a user the validate XML documents against appropriate schemas. Secondly the Semantos XML schema itself will be validated for correctness. This will be accomplished by using the online W3C schema validator at the URL "http://www.w3.org/2001/03/webdata/xsv". Showing that the schema is a valid and correct XML schema allows us to make the statement that any Semantos query that is validated by the Semantos schema is indeed valid and well formed XML.

5.2.2. RESULTS

For the evaluation of this criteria several Semantos queries were constructed and tested against the schema validator. All of the queries pass the validation without any warnings. The results from the schema validation is provided below, for two of the tested queries.

```
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema"
 <semantos:source name="Open Directory Content Dump"
  uri="http://www.retrorabbit.co.za/data/od_structure.rdf"/>
 <semantos:namespace name="rdf"
  uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
 <semantos:namespace name="purl"
  uri="http://purl.org/dc/elements/1.0/#" />
 <semantos:namespace name="dmoz"
  value=" http://dmoz.org/rdf#" />
 <semantos:entity name="topicresources">
  <semantos:attribute name="?topicname"/>
  <semantos:attribute name="rdf:resource"/>
  <semantos:graph>
   <semantos:triple subject="dmoz:Topic" predicate="rdf:id"
    object="?topicname"/>
   <semantos:triple subject="dmoz:Topic" predicate="dmoz:link"
```

```
    object="rdf:resource"/>
  </semantos:graph>
  <semantos:filter>
   <semantos:condition attribute="rdf:id" operator="eq"
    value="Top/Arts/Movies/Titles/1/10_Rillington_Place" />
  </semantos:filter>
 </semantos:entity>
</semantos:fetch>
```

Figure 40 Semantos query one

Using the online validator, query one as shown in Figure 40 is approved as both a well formed XML document and successfully passes schema validation.

```
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:source name="Open Directory Content Dump"
  uri="http://www.retrorabbit.co.za/data/od_structure.rdf"/>

 <semantos:namespace name="rdf"
  uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
 <semantos:namespace name="purl"
  uri="http://purl.org/dc/elements/1.0/#" />
 <semantos:namespace name="dmoz"
  uri=" http://dmoz.org/rdf#" />
 <semantos:entity name="topicresources">
  <semantos:attribute name="?title"/>
  <semantos:attribute name="?description"/>
  <semantos:graph>
   <semantos:triple subject="dmoz:ExternalPage"
    predicate="purl:Title" object="?title"/>
   <semantos:triple subject="dmoz:ExternalPage"
    predicate="purl:Description" object="?description"/>
  </semantos:graph>
 </semantos:entity>
</semantos:fetch>
```

Figure 41 Semantos query two

Query two, as provided in Figure 41, also passes the online schema validation test successfully validating against both the well formed XML document criteria as well as the schema validation.

Validating the Semantos schema (from Appendix A) against the W3C schema validator yielded positive results. The report from the validator confirms that the

Semantos query schema is in fact a valid XML schema, as the report below indicates:

```
Schema validating with XSV 3.1-1 of 2007/12/11 16:20:05

    * Target: file:/usr/local/XSV/xsvlog/tmp5dC7couploaded
        (Real name: schema.xml)
    * docElt: {http://www.w3.org/2001/XMLSchema}schema
    * Validation was strict, starting with type [Anonymous]
    * The schema(s) used for schema-validation had
      no errors
    * No schema-validity problems were found in the target
```

5.3. XML OUTPUT

The second evaluation criteria establishes that a Semantos query must provide results in a valid XML format. In order to evaluate this criteria, several requests are sent to the Semantos service and each result set returned is then checked against an XML validator. Although this test does not conclusively establish that all Semantos queries will result in valid XML it does illustrate a method to test the XML validity of Semantos results.

5.3.1.METHOD

For the purposes of this experiment we have used two validators to test for valid XML. First the free and online W3C DOM XML evaluater is used to validate the XML result. This evaluator is available from the URL "http://www.w3schools.com/Dom/dom_validate.asp" and is capable of validating XML provided in a text box as well as a publicly available file. Secondly, use is made of the production software XMLSpy® 2008 from Altova software. The software can be downloaded for a free trial from the URL "http://www.altova.com/products/xmlspy/xml_editor.html". By proofing the results against both XML validators it is possible for us to experimentally verify that in all likelihood a Semantos query would yield well formed XML results.

5.3.2. Results

For the input to this experiment, the results from the queries used in the XML representation evaluation above are used. All result sets passed the XML validation test successfully. Provided below, are the query results and XML validation results for the two queries shown from the previous evaluation.

```
<semantos:dataset
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/1984_-_1984
  </semantos:topicname>
  <semantos:resource>
   http://www.geocities.com/aaronbcaldwel/1984.html
  </semantos:resource>
 </semantos:result>
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/1984_-_1984
  </semantos:topicname>
  <semantos:resource>
   http://orwell.ru/a_life/movies/m84_01.htm
  </semantos:resource>
 </semantos:result>
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/1984_-_1984
  </semantos:topicname>
  <semantos:resource>
   http://us.imdb.com/title/tt0087803/
  </semantos:resource>
 </semantos:result>
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/187
  </semantos:topicname>
  <semantos:resource>
   http://www.wbmovies.com/187/
  </semantos:resource>
 </semantos:result>
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/187
  </semantos:topicname>
  <semantos:resource>
   http://www.movieweb.com/movie/187/index.html
  </semantos:resource>
 </semantos:result>
 <semantos:result>
  <semantos:topicname>
   Top/Arts/Movies/Titles/1/187
  </semantos:topicname>
  <semantos:resource>
```

```
   http://us.imdb.com/title/tt0118531/
  </semantos:resource>
 </semantos:result>
</semantos:dataset>
```

Figure 42 Excerpt from data results for query one

The data result, excerpt in Figure 42, for query one is validated as both a well formed XML document and successfully passes schema validation, using both validating tools.

```
<semantos:dataset
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:result>
  <semantos:title>
   British Horror Films: 10 Rillington Place
  </semantos:title>
  <semantos:description>
   Review which looks at plot especially the shocking features of it.
  </semantos:description>
 </semantos:result>
 <semantos:result>
  <semantos:title>
   Top 100 Movie Lists: 1984
  </semantos:title>
  <semantos:description>
   Photos, sounds [Real Audio], and a review.
  </semantos:description>
 </semantos:result>
 <semantos:result>
  <semantos:title>
   George Orwell's Movies - 1984
  </semantos:title>
  <semantos:description>
   Review.
  </semantos:description>
 </semantos:result>
 <semantos:result>
  <semantos:title>
   187
  </semantos:title>
  <semantos:description>
   Official site. Includes synopsis, trailer, cast biographies,
   background information, reviews, production notes, and related links.
  </semantos:description>
 </semantos:result>
 <semantos:result>
  <semantos:title>
   FilmScouts: 187
  </semantos:title>
  <semantos:description>
   Production information, synopsis, filmmaker and cast biographies,
   and video clips.
  </semantos:description>
 </semantos:result>
 <semantos:result>
  <semantos:title>
   Cinebooks Database - When Bad Kids Happen to Good Teachers
  </semantos:title>
  <semantos:description>
   Review by Maitland McDonagh (predominantly negative), rating.
  </semantos:description>
 </semantos:result>
</semantos:dataset>
```

Figure 43 Excerpt from data results for query two

With the same success the data result for query two, excerpt in Figure 43, is also a well formed XML document and passes schema validation.

5.4. MUTUALLY EMBEDDABLE WITH XML

This evaluation criteria establishes that a Semantos query should be embeddable within another XML document and conversely that arbitrary XML markup should be embeddable within a Semantos query.

5.4.1. METHOD

In order to evaluate this criteria we will be testing three different scenarios. The first is a Semantos query that is nested within a random XML markup document. We will test for this scenario by adding a Semantos query inside an XHTML document, post Semantos processing the results should then yield an XHTML document with the Semantos query part replaced by the results of the Semantos query. The second scenario represents a Semantos query with random XML embedded in the result shaping part, similar to the constructs found in XSLT. To test against this scenario we will construct a Semantos query with some XHTML elements added to give the results the structure of an XHTML list. The third and final scenario looks at having Semantos embedded inside XML and some more XML embedded in the Semantos query itself, essentially combining scenarios one and two.

5.4.2. Results

The first scenario requires XHTML tags around the Semantos query itself. Shown below, in Figure 46, is the XHTML embedded Semantos query:

```
<html>
<head>
<title>Semantos Result Page</title>
</head>
<body>
<h1>Semantos Result Page</h1>
<div>
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:source name="Open Directory Content Dump"
  uri="http://www.retrorabbit.co.za/data/od_structure.rdf"/>

 <semantos:namespace name="rdf"
  uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
 <semantos:namespace name="purl"
  uri="http://purl.org/dc/elements/1.0/#" />
 <semantos:namespace name="dmoz"
  uri=" http://dmoz.org/rdf#" />
 <semantos:entity name="topicresources">
  <semantos:attribute name="?title"/>
  <semantos:attribute name="?description"/>
  <semantos:graph>
   <semantos:triple subject="dmoz:ExternalPage"
    predicate="purl:Title" object="?title"/>
   <semantos:triple subject=" dmoz:ExternalPage"
    predicate="purl:Description" object="?description"/>
  </semantos:graph>
 </semantos:entity>
</semantos:fetch>
</div>
</body>
</html>
```

Figure 44 Semantos query embedded in XML

The XHTML embedded query resulted in Figure 45 shown below:

```
<html>
<head>
<title>Semantos Result Page</title>
</head>
<body>
<h1>Semantos Result Page</h1>
<div>
<semantos:dataset
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
```

```
  <semantos:result>
   <semantos:title>
    British Horror Films: 10 Rillington Place
   </semantos:title>
   <semantos:description>
    Review which looks at plot especially the shocking features of it.
   </semantos:description>
  </semantos:result>
  <semantos:result>
   <semantos:title>
    Top 100 Movie Lists: 1984
   </semantos:title>
   <semantos:description>
    Photos, sounds [Real Audio], and a review.
   </semantos:description>
  </semantos:result>
  <semantos:result>
   <semantos:title>
    George Orwell's Movies - 1984
   </semantos:title>
   <semantos:description>
    Review.
   </semantos:description>
  </semantos:result>
  <semantos:result>
   <semantos:title>
    187
   </semantos:title>
   <semantos:description>
    Official site. Includes synopsis, trailer, cast biographies,
    background information, reviews, production notes, and related links.
   </semantos:description>
  </semantos:result>
  <semantos:result>
   <semantos:title>
    FilmScouts: 187
   </semantos:title>
   <semantos:description>
    Production information, synopsis, filmmaker and cast biographies,
    and video clips.
   </semantos:description>
  </semantos:result>
  <semantos:result>
   <semantos:title>
    Cinebooks Database - When Bad Kids Happen to Good Teachers
   </semantos:title>
   <semantos:description>
    Review by Maitland McDonagh (predominantly negative), rating.
   </semantos:description>
  </semantos:result>
 </semantos:dataset>
 </div>
 </body>
 </html>
```

Figure 45 Results for Semantos query embedded in XML

The second scenario requires XHTML tags embedded within the Semantos query. This specific query is given below in Figure 46:

```
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:source name="Open Directory Content Dump"
  uri="http://www.retrorabbit.co.za/data/od_structure.rdf"/>

 <semantos:namespace name="rdf"
  uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
 <semantos:namespace name="purl"
  uri="http://purl.org/dc/elements/1.0/#" />
 <semantos:namespace name="dmoz"
  uri=" http://dmoz.org/rdf#" />
 <semantos:entity name="topicresources">
  <li><semantos:attribute name="?title"/> -
  <semantos:attribute name="?description"/></li>
  <semantos:graph>
   <semantos:triple subject="dmoz:ExternalPage"
    predicate="purl:Title" object="?title"/>
   <semantos:triple subject=" dmoz:ExternalPage"
    predicate="purl:Description" object="?description"/>
  </semantos:graph>
 </semantos:entity>
</semantos:fetch>
```

Figure 46 XML embedded in Semantos query

The result for the query in Figure 46 is given below in Figure 47:

```
<semantos:dataset
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:result>
  <li><semantos:title>
   British Horror Films: 10 Rillington Place
  </semantos:title>&nbsp;-&nbsp;
  <semantos:description>
   Review which looks at plot especially the shocking features of it.
  </semantos:description></li>
 </semantos:result>
 <semantos:result>
  <li><semantos:title>
   Top 100 Movie Lists: 1984
  </semantos:title>&nbsp;-&nbsp;
  <semantos:description>
   Photos, sounds [Real Audio], and a review.
  </semantos:description></li>
 </semantos:result>
 <semantos:result>
  <li><semantos:title>
   George Orwell's Movies - 1984
  </semantos:title>&nbsp;-&nbsp;
```

```
    <semantos:description>
      Review.
    </semantos:description></li>
  </semantos:result>
  <semantos:result>
    <li><semantos:title>
      187
    </semantos:title>&nbsp;-&nbsp;
    <semantos:description>
      Official site. Includes synopsis, trailer, cast biographies,
      background information, reviews, production notes, and related links.
    </semantos:description></li>
  </semantos:result>
  <semantos:result>
    <li><semantos:title>
      FilmScouts: 187
    </semantos:title>&nbsp;-&nbsp;
    <semantos:description>
      Production information, synopsis, filmmaker and cast biographies,
      and video clips.
    </semantos:description></li>
  </semantos:result>
  <semantos:result>
    <li><semantos:title>
      Cinebooks Database - When Bad Kids Happen to Good Teachers
    </semantos:title>&nbsp;-&nbsp;
    <semantos:description>
      Review by Maitland McDonagh (predominantly negative), rating.
    </semantos:description></li>
  </semantos:result>
</semantos:dataset>
```

Figure 47 Results for Query with embedded XML

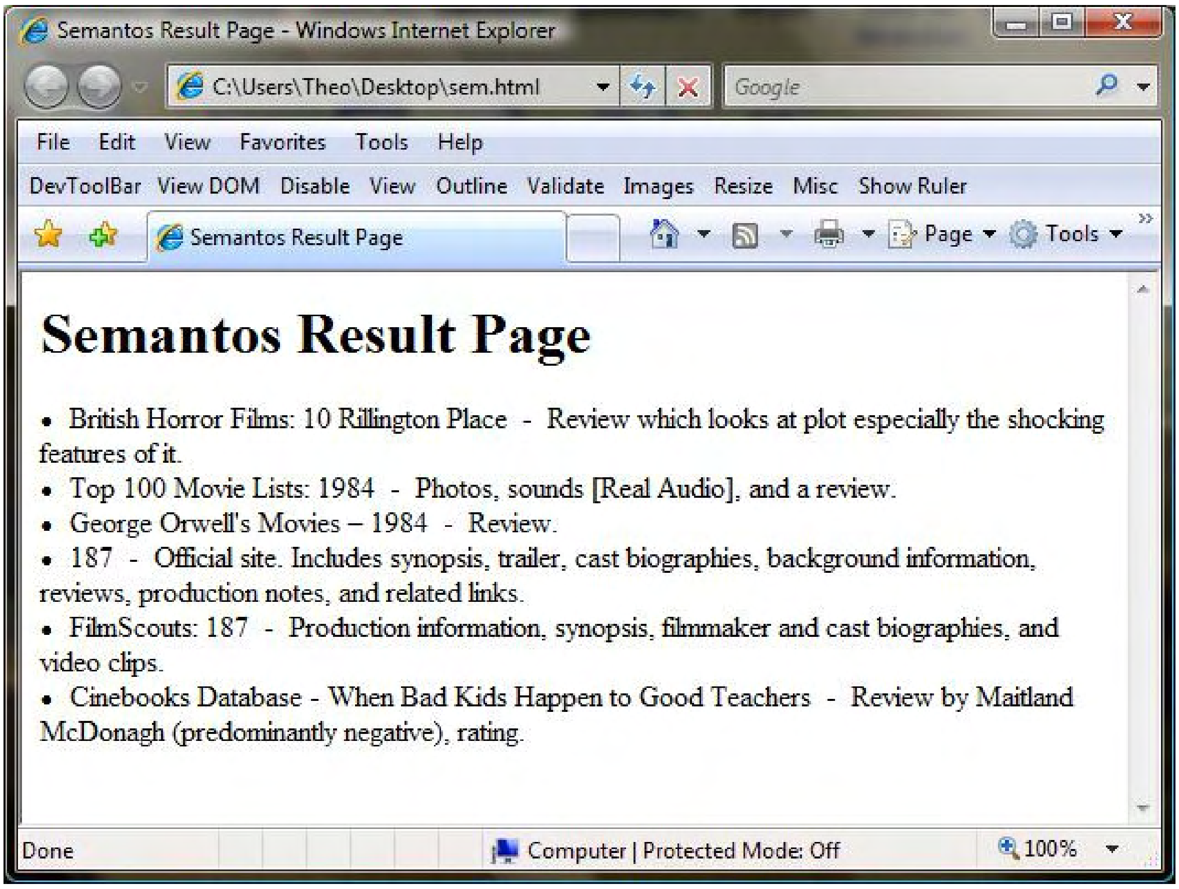
Combining both the above scenarios into a single query results in the following, as illustrated in Figure 48:

```
<html>
<head>
<title>Semantos Result Page</title>
</head>
<body>
<h1>Semantos Result Page</h1>
<div>
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
  <semantos:source name="Open Directory Content Dump"
    uri="http://www.retrorabbit.co.za/data/od_structure.rdf"/>

  <semantos:namespace name="rdf"
    uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
  <semantos:namespace name="purl"
    uri="http://purl.org/dc/elements/1.0/#" />
  <semantos:namespace name="dmoz"
    uri=" http://dmoz.org/rdf#" />
  <semantos:entity name="topicresources">
   <li><semantos:attribute name="?title"/> -
   <semantos:attribute name="?description"/></li>
   <semantos:graph>
    <semantos:triple subject="dmoz:ExternalPage"
     predicate="purl:Title" object="?title"/>
    <semantos:triple subject=" dmoz:ExternalPage"
     predicate="purl:Description" object="?description"/>
   </semantos:graph>
  </semantos:entity>
 </semantos:fetch>
 </div>
 </body>
 </html>
```

Figure 48 Combination Semantos query

As can be seen from the result in the experiments above, Semantos is indeed mutually embeddable with XML. Opening the results from the combination query in an internet browser window, yields the image as shown in Figure 49.


Semantos Result Page - Windows Internet Explorer
C:\Users\Theo\Desktop\sem.html
Google
File Edit View Favorites Tools Help
DevToolBar View DOM Disable View Outline Validate Images Resize Misc Show Ruler
Semantos Result Page
Page Tools
Done
Computer | Protected Mode: Off
100%

Semantos Result Page

- British Horror Films: 10 Rillington Place - Review which looks at plot especially the shocking features of it.
- Top 100 Movie Lists: 1984 - Photos, sounds [Real Audio], and a review.
- George Orwell's Movies – 1984 - Review.
- 187 - Official site. Includes synopsis, trailer, cast biographies, background information, reviews, production notes, and related links.
- FilmScouts: 187 - Production information, synopsis, filmmaker and cast biographies, and video clips.
- Cinebooks Database - When Bad Kids Happen to Good Teachers - Review by Maitland McDonagh (predominantly negative), rating.

Figure 49 XHTML result opened in Internet Explorer

5.5. SERVER-SIDE PROCESSING

This particular evaluation criteria ensures that Semantos can be used in a distributed web environment. The criteria establishes that a query should be self contained and that they are remotely executable.

5.5.1. METHOD

The test for this evaluation criteria is fairly simple. We will wrap the Semantos processing engine in a hosted web service. From a remote machine a query is then constructed and passed to the web service for processing. If the web service is capable of yielding results, then we satisfy both the self containment and remotely executable conditions.

5.5.2. RESULTS

Creating a small snippet of code (see Figure 50) for the web service is a relatively simple task. The only job that the web service has to complete is to forward a query request to the Semantos processing assembly and return the yielded results. This is possible because the Semantos processing engine works with URI resource locations, which means that it can only process a request, if the required resources are globally available.

```
[WebMethod (Description="Process a Semantos query provided as a string
and return the resultant XML response as a string.")]
public string ProcessQuery(string query)
{
    XDocument query = XDocument.Parse(query);
    XDocument result = Semantos.Instance.ProcessQuery(query);

    return result.ToString();
}
```

Figure 50 Web service code to handle Semantos query.

To call the web service from another application, a web service reference is added, pointing to the hosted Semantos web service. The exposed service methods are then capable of receiving queries in string format. Figure 51 below provides an example of using the web service.

```
Semantos.PublicService service = new Semantos.PublicService();
XNamespace semantos = "http://www.retrorabbit.co.za/semantos#";
string query =
  new XDocument(
     new XDeclaration("1.0", null, null),
     new XElement(semantos + "fetch",
       new XElement(semantos + "source",
          new XAttribute(semantos + "name", "source"),
          new XAttribute(semantos + "uri",
          @"http://www.retrorabbit.co.za/data/od_structure.rdf")),
     new XElement(semantos + "namespace",
          new XAttribute(semantos + "name", "rdf"),
          new XAttribute(semantos + "uri",
          @" http://www.w3.org/1999/02/22-rdf-syntax-ns#")),
     new XElement(semantos + "namespace",
          new XAttribute(semantos + "name", "purl"),
          new XAttribute(semantos + "uri",
          @" http://purl.org/dc/elements/1.0/#")),
     new XElement(semantos + "namespace",
          new XAttribute(semantos + "name", "dmoz"),
          new XAttribute(semantos + "uri",
          @" http://dmoz.org/rdf#")),


       new XElement(semantos + "entity",
          new XAttribute(semantos + "name", "topicresources"),
          new XElement(semantos + "attribute",
            new XAttribute(semantos + "name", "?topicname"))),
          new XElement(semantos + "attribute",
            new XAttribute(semantos + "name", "rdf:resource"))),
               new XElement(semantos + "graph",
               new XElement(semantos + "triple",
                 new XAttribute(semantos + "subject", "dmoz:Topic"),
                 new XAttribute(semantos + "predicate", "rdf:id"),
                 new XAttribute(semantos + "object", "?topicname"))
               new XElement(semantos + "triple",
                 new XAttribute(semantos + "subject", "dmoz:Topic"),
                 new XAttribute(semantos + "predicate", "dmoz:link"),
                 new XAttribute(semantos + "object", "rdf:resource"))

          ))).ToString();

string result = service.ProcessQuery(query);
```

Figure 51 Calling the Semantos web service

By comparing the results from using the Semantos assembly library on its own, with the results of using the Semantos public webservice it is found that the results are exactly alike. The conclusion can therefore be made that Semantos can be used in server side processing arrangements because it can be exposed as a web service (or any other remoting technology for that matter) and uses URI address locations to access data.

5.6. No Schema Required

This evaluation criteria establishes that Semantos should be able to query a data source without any prior knowledge (usually in the form of an XML schema) regarding the data source being queried.

5.6.1. Method

Validating this criteria requires a devious test. In order to test for a query against a data source for which Semantos would not possibly have a schema, we will modify the structure of the data source randomly, before the Semantos query is executed. This modification will be along the lines of exchanging some of the root elements with a newly created differently named element, ensuring that Semantos could not possibly have had prior knowledge of the structure of the data. The query should still yield results where possible, i.e. where the structure of the data has not been malformed from our randomization and where the data matches the filter provided in the query.

5.6.2. Results

Shown in Figure 52 is the process of taking an existing XML document and replacing 1000 of the child elements with a newly created and renamed XML element.

```
protected XElement RandomSelectLeafNode(Random rand, XElement current)
{
   XElement newcurrent = null;
   if (current.Descendants().Count() == 0)
      return newcurrent;
   else
      return RandomSelectLeafNode(rand,
      current.Descendants().
      ElementAtOrDefault(rand.Next(document.Descendants().Count())));
   return null;
}

protected XDocument RandomizeXml(XDocument document)
{
   // Randomly change a 1000 elements names
   Random rand = new Random();
   XElement current = null;
   for (int i = 0; i < 1000; i++)
   {
      // Randomly select 1000 leaf nodes and replace them with
      empty elements of different name.
      current = RandomSelectLeafNode(rand, current);
      current.ReplaceWith(new XElement("RandomElementNumber" + i));
   }
   return document;
}
```

Figure 52Randomly modify an XML string.

Experimentation with the schema randomization code above indicates that the results returned from Semantos with random nodes is similar to results obtained by not modifying the XML schema at all. This draws the conclusion that even though the schema drastically changed, the Semantos engine was still capable of returning the correct results.

5.7. PROGRAMMATIC MANIPULATION

This evaluation criteria establishes programmatic manipulation and creation of Semantos queries. Programmatic manipulation ensures that it should be possible to construct and change a Semantos query from code.

5.7.1.METHOD

In order to satisfy this requirement it must be possible to build a Semantos query using code. We will set up a small experiment where a Semantos query is programatically constructed from a few user provided inputs. Given these inputs the code will construct a query without any user intervention, demonstrating that it possible to build queries from code.

5.7.2. RESULTS

It is very simple to show the programmatic manipulation of Semantos queries. The code in Figure 53 shows a method that receives two string parameters. These parameters modify the Semantos query – it is also possible to modify the structure of the Semantos query using conditional and repeating constructs.

```
private string BuildQuery(string topicLikeParameter1,
                          string topicLikeParameter2)
{


XNamespace semantos = "http://www.retrorabbit.co.za/semantos/schema";

  string query = new XDocument(
    new XDeclaration("1.0", null, null),
    new XElement(semantos + "fetch",
      new XElement(semantos + "source",
      new XAttribute(semantos + "name", "source"),
      new XAttribute(semantos + "uri",
      @"http://www.retrorabbit.co.za/data/od_structure.rdf")),
    new XElement(semantos + "namespace",
    new XAttribute(semantos + "name", "rdf"),
    new XAttribute(semantos + "uri",
    @" http://www.w3.org/1999/02/22-rdf-syntax-ns#")),
    new XElement(semantos + "namespace",
      new XAttribute(semantos + "name", "purl"),
      new XAttribute(semantos + "uri",
      @" http://purl.org/dc/elements/1.0/#")),
    new XElement(semantos + "namespace",
      new XAttribute(semantos + "name", "dmoz"),
      new XAttribute(semantos + "uri", @" http://dmoz.org/rdf#")),

      new XElement(semantos + "entity",
        new XAttribute(semantos + "name", "topicresources"),
        new XElement(semantos + "attribute",
          new XAttribute(semantos + "name", "?topicname"))),
        new XElement(semantos + "attribute",
          new XAttribute(semantos + "name", "rdf:resource"))),

        new XElement(semantos + "graph",
          new XElement(semantos + "triple",
            new XAttribute(semantos + "subject", "dmoz:Topic"),
            new XAttribute(semantos + "predicate", "rdf:id"),
            new XAttribute(semantos + "object", "?topicname")),
          new XElement(semantos + "triple",
            new XAttribute(semantos + "subject", "dmoz:Topic"),
            new XAttribute(semantos + "predicate", "dmoz:link"),
            new XAttribute(semantos + "object", "rdf:resource"))),
          new XElement(semantos + "filter",
            new XAttribute(semantos + "type", "or"),
            new XElement(semantos + "condition",
              new XAttribute(semantos + "attribute", "topicname"),
              new XAttribute(semantos + "operator", "like"),
              new XAttribute(semantos + "value", topicLikeParameter1),
            new XElement(semantos + "condition",
              new XAttribute(semantos + "attribute", "topicname"),
              new XAttribute(semantos + "operator", "like"),
              new XAttribute(semantos + "value", topicLikeParameter2))
          )).ToString();

  return query;
}
```

Figure 53Programmatic manipulation of Semantos query.

5.8. SUPPORT NEW DATA TYPES

The data type evaluation criteria establishes that Semantos queries are not restricted by data types and it should be possible to build a Semantos query given any arbitrary data type.

5.8.1.METHOD

For this criteria we will devise a simple test. In order to satisfy the criteria of data type independence a new arbitrary data type is created, this data type represents floating point numbers and is stored in the format ##%## where # represents any number of integer numerals. The number is then a calculated value of dividing the two integer numbers seperated by the % sign. This is not a very useful data type, but it does allow us to see if Semantos is capable of working with arbitrary data types. When working with a dataset of unknown type, Semantos will default the behaviour of that type to a string representation, requiring post processing of the data if and when required. A dataset loaded with these data types is loaded and queried against, verifying whether or not Semantos handles the data type properly.

5.8.2. RESULTS

The data set used in the testing of this criteria is presented below in Figure 54. The source is a very simple data set giving each unique test case an identification number (id), a decimal value accurate to 4 decimal places (decimal_representation) and then the same value in the decimal_representation tag is repeated again in the strange data type (strange_representation).

```
<testcases>
 <testcase>
  <id>1</id>
  <decimal_representation>1.0000</decimal_representation>
  <strange_representation>2441%2441</strange_representation>
 </testcase>
 <testcase>
  <id>2</id>
  <decimal_representation>0.1509</decimal_representation>
  <strange_representation>234%1551</strange_representation>
 </testcase>
 <testcase>
  <id>3</id>
  <decimal_representation>13.1492</decimal_representation>
  <strange_representation>8021%610</strange_representation>
 </testcase>
 <testcase>
  <id>4</id>
  <decimal_representation>2.0000</decimal_representation>
  <strange_representation>6%3</strange_representation>
 </testcase>
 <testcase>
  <id>5</id>
  <decimal_representation>0.7952</decimal_representation>
  <strange_representation>66%83</strange_representation>
 </testcase>
...
<testcases>
```

Figure 54 Excerpt from data source file

Verifying that Semantos returns the values accurately is a simple matter of running the query provided (Figure 55) and then verifying that the strange representation does indeed match the decimal representation.

```
<semantos:fetch
xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema">
 <semantos:source name="Custom Data Type Verification"
```

```
   uri="http://www.retrorabbit.co.za/data/custom_type.rdf"/>

  <semantos:namespace name="rdf"
   uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
  <semantos:entity name="testcases">
   <semantos:attribute name="?id"/>
   <semantos:attribute name="?decimal_representation"/>
   <semantos:attribute name="?strange_representation"/>
   <semantos:graph>
    <semantos:triple subject="testcase"
     predicate="id" object="?id"/>
    <semantos:triple subject="testcase"
     predicate="decimal_representation"
     object="?decimal_representation"/>
    <semantos:triple subject="testcase"
     predicate="strange_representation"
     object="?strange_representation"/>
   </semantos:graph>
  </semantos:entity>
</semantos:fetch>
```

Figure 55 Semantos data type verification query

The results from the query are then compared line by line. The comparison compares the decimal representation with the calculated value of the strange representation. The code below, Figure 56, only returns results where the two representations did not match. Yielding no erroneous representations (all the representations matched), the conclusion can be made that Semantos is indeed capable of working with new data types, as long as these data types are serializable.

```
private decimal GetDecimalRepresentation(string strangeRepresentation)
{
    string[] parts = strangeRepresentation.Split(new char[] { '%' });
    return Math.Round(
        Convert.ToDecimal(parts[0]) /
        Convert.ToDecimal(parts[1]), 4, MidpointRounding.AwayFromZero);
}

private void RunTest(string query)
{
    XNamespace semantos = "http://www.retrorabbit.co.za/semantos/schema";
    XDocument query = XDocument.Parse(query);
    XDocument result = Semantos.Instance.ProcessQuery(query);

    var incorrect = from n in result.Root.Elements(semantos + "result")
        where Convert.ToDecimal(
             n.Element(semantos + "decimal_representation").Value) !=
             GetDecimalRepresentation(
             n.Element(semantos + "strange_representation").Value)
        select new {
             Id = n.Element(semantos + "id").Value,
                DecR = n.Element(semantos + "decimal_representation").Value,
                StrR = n.Element(semantos + "strange_representation").Value};

    Console.Out.WriteLine("Number of erroneous matches: " +
        incorrect.Count());
}
```

Figure 56 Code comparing decimal and strange data types.

5.9. Summary

Semantos successfully passed all the evaluation criteria tests posed to the language, as summarized below in Table 2 . This indicates that Semantos is indeed a language consisting of pure XML syntax and is fully embeddable with XML. It also shows that Semantos has query language constructs for projecting information, is machine processable and can work in a distributed fashion.

Criteria	**Evaluation**
XML representation	PASSED
XML output	PASSED
Mutually embeddable with XML	PASSED
Server-side processing	PASSED
No schema required	PASSED
Programmatic manipulation	PASSED
Support new data types	PASSED

Table 2 Semantos measured against evaluation criteria.

CHAPTER 6: USE CASES

"Tell me and I forget. Teach me and I remember. Involve me and I learn." -- Benjamin Franklin

The primary strength of the Semantos query language, is the fact that a Semantos query can be manipulated by either a software agent, or a human developer and with equal ease. This benefit may seem obvious, but the advantage of enabling software agents to manipulate the query, and not merely process it, is that a whole new range of applications is opened up for investigation.

As a semantically smart query language, the future applications for Semantos are as varied as they are interesting. Apart from the obvious application of using Semantos as a stock, standard RDF query language, there are two other unique applications. These applications will be provided here, in order to highlight the flexibility of the language. The first application is the use of Semantos as an intermediary language, operating between other RDF query languages. This would benefit the interoperability of different RDF data sources, as discussed below. The second, perhaps more interesting, application is the use of "query enhancers". The next section illustrates the possibility of enhancing a distributed Semantos query by injecting it with ontology knowledge from different query enhancement services.

6.1. SEMANTOS AS INTERMEDIARY LANGUAGE

It has already been established that XML has the strength to facilitate communication between disparate systems. It achieves this through the translation of information and instructions from system A into a serializable XML format, which is then transported to system B, as illustrated in Figure 57. System B then has the ability to extract the data and instructions from the XML into information native to system B.

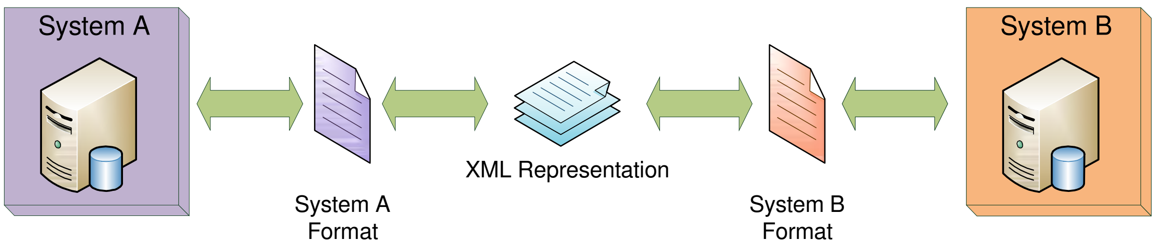


Figure 57 XML as a communication intermediary

Semantos, as an XML query language, inherits this positive trait from its technological ancestor, and may therefore be used to similarly facilitate communication between systems. In the case of Semantos the systems that we are translating between, are RDF data source systems. In order to be able to translate queries from one RDF language to another, a thorough mapping between language constructs from other languages and language constructs from Semantos takes place, as will be demonstrated. In particular, in this case, SPARQL and RQL are used as they are the most widely used. However, this exercise may be repeated for any other languages.

6.1.1. MAPPING QUERY CONSTRUCTS

In this section a close correlation between the language constructs of Semantos and the language constructs of SPARQL and RQL will be highlighted. This correlation facilitates the mapping of queries from SPARQL or RQL to Semantos and back. Once Semantos constructs have been mapped to SPARQL and RQL constructs individually, Semantos may be used as an intermediary, to map between SPARQL and RQL. This process may be applied to any arbitrary RDF query language, enabling any Semantos construct compatible query language to interact with any other RDF language matching the same criteria.

To show the correlation between language constructs, the basic components of an RDF query namely: the “include component”, the “attribute component” and the “graph component”, will be examined. Examples from each of the three language components will be provided, and the importance of these language components or constructs, will be discussed.

The "include component": The "include component" allows the query in question to incorporate ontology information from external sources. These sources may be from any ontology language; in the case of RDF this would be in the form of an RDF Schema (RDFS) document. The ontology information referenced by a query prefix is included in the query processing at the reasoning stage. The ontology information provides the query processing engine with knowledge about a certain domain, in order for it to reason effectively about the domain in question. By allowing queries to include any arbitrary ontology, the reasoning power of the language is improved drastically.

- SPARQL: possesses the ability to include or reference ontology information in schema documents through the PREFIX keyword, as illustrated in Figure 58.

```
PREFIX human: <http://ontology/humans.rdfs#>
```

Figure 58 The SPARQL include component

- RQL: via the USING NAMESPACE syntax, RQL is able to import any number of schema or ontology references, as shown in Figure 59.

```
using namespace human = http://ontology/humans.rdfs#
```

Figure 59 The RQL include component

- Semantos: uses the ontology tag to define and include external ontologies or schema documents, this can be seen in Figure 60.

```
<semantos:ontology name="human" uri="http://ontology/humans.rdfs" />
```

Figure 60 The Semantos include component

The "attribute component": The "attribute component" identifies the information column projections, which need to be retrieved by the query. In relational database terminology this would be the column set returned by the query. The same applies for RDF queries, although in this case the "attribute component" identifies attributes that map to variable nodes in the semantic query graph.

- SPARQL: utilizes a very simple solution which employs the SELECT keyword followed by a space delimited list of required attributes, as shown in Figure 61.

```
SELECT ?name ?spousename
```

Figure 61 The SPARQL attribute component

- RQL: implements the same syntax for defining the result set attributes as SPARQL. The only difference is that RQL has an extra @ character in front of attributes when they are bound to a predicate. An example of a RQL projection is provided in Figure 62.

```
SELECT name, spousename
```

Figure 62 The RQL attribute component

- Semantos: provides a more involved attribute selection component by allowing the definition of entities, which in turn contain attributes. This construct is shown in Figure 63.

```
<semantos:entity name="person">
  <semantos:attribute name="name"/>
  <semantos:attribute name="spousename"/>
  ...
</semantos:entity>
```

Figure 63 The Semantos attribute component

The "graph component": The "graph component" is responsible for constructing a context graph, which identifies the structure of the data being queried. Any nodes in the RDF data source that match the "shape" of the context graph, may be returned by the results of the query. In conjunction with the "attribute component", the "graph component" fully identifies and describes the shape and nature of the projected data set which will be returned by the query.

- SPARQL: employs a very straight forward approach to defining the graph, by way of the WHERE keyword. The keyword is followed by an unordered list of context graph triples, illustrated in Figure 64.

```
WHERE { ?person human:name ?name.
 ?person human:hasSpouse ?spouse.
 ?spouse human:name ?spousename.
 ?person rdf:type human:Woman. }
```

Figure 64 The SPARQL graph component

- RQL: The RQL graph declaration is defined by the FROM keyword. The triple sets are uncomplicated and provide the enhanced ability to bind types to the subject and object. An example of such a declaration can be found in Figure 65.

```
FROM
  {person}, human:name, {name},
  {person}, human:hasSpouse, {spouse},
  {spouse}, human:name, {spousename},
  {person}, rdf:type, {X : human:Woman}
```

Figure 65 The RQL graph component

- Semantos: The Semantos graph component is elementary; employing the graph and triple tags to define the shape of the context graph, shown in Figure 66.

```
<semantos:graph>
   <semantos:triple subject="person" predicate="human:name"
       object="name"/>
   <semantos:triple subject="person" predicate="rdf:type"
       object="human:Woman"/>
     <semantos:triple subject="person" predicate="human:hasSpouse"
       object="spouse"/>
   <semantos:triple subject="spouse" predicate="human:name"
       object="spousename"/>
  </semantos:graph>
```

Figure 66 The Semantos graph component

6.1.2. Implementation

By defining a one-to-many relationship between the language components of Semantos and the language components of RQL and SPARQL, it has been demonstrated that it is indeed possible so find a one-to-one mapping between Semantos and any other arbitrary RDF query language. This concept is diagrammatically shown in Figure 67.

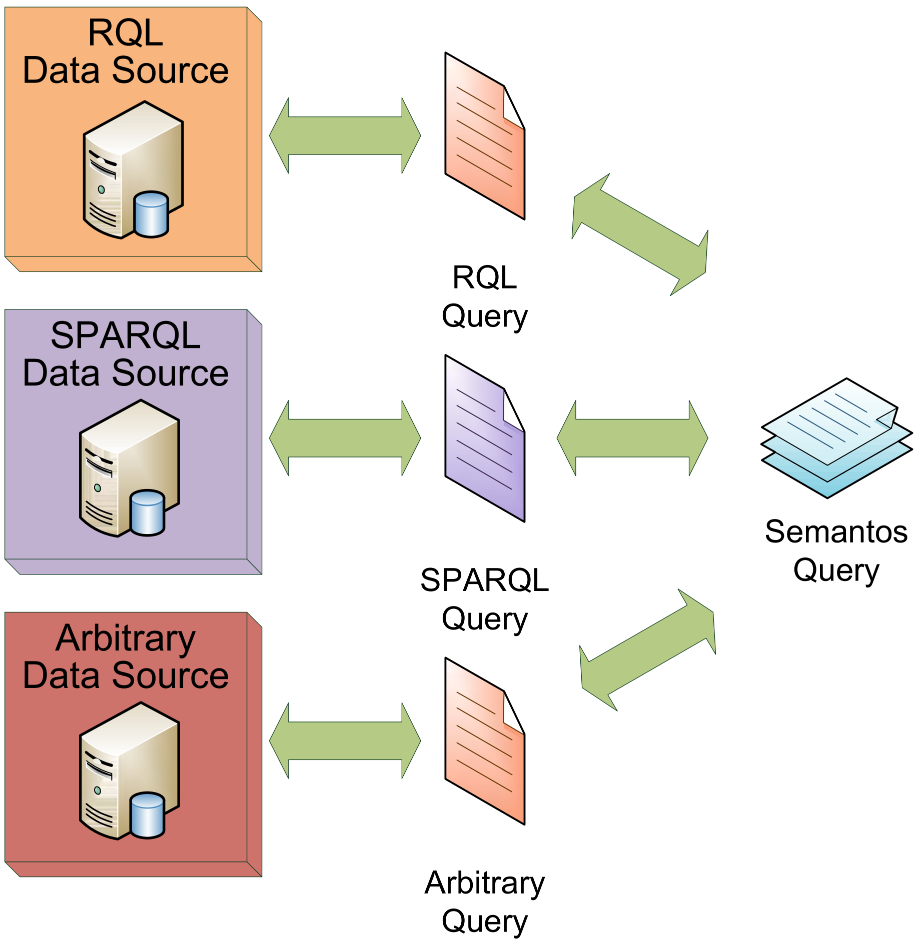


Figure 67 Semantos as RDF language intermediary

As an example of how simple such a translating application would be, an extract from a language converting web service written in C#, is included below. This extract is a method for transforming a Semantos query, provided as an XmlDocument, into a SPARQL query. The implementation is somewhat crude, but still very effective (Figure 68):

```
public static QueryPart[] ConvertToSparQL(XmlDocument query)
{
      // Get the prefixes
      XmlNamespaceManager namespaceManager =
            new XmlNamespaceManager(query.NameTable);
      namespaceManager.AddNamespace("semantos",
            @"http://semantos.retrorabbit.co.za/");
```

```
StringBuilder builder = new StringBuilder();
XmlNodeList namespaces =
     query.SelectNodes("//semantos:namespace", namespaceManager);

foreach (XmlNode n_namespace in namespaces)
{
     builder.AppendFormat(" PREFIX {0}: <{1}> ",
          n_namespace.Attributes["name"].Value,
          n_namespace.Attributes["value"].Value);
}

string prefixes = builder.ToString();
builder = new StringBuilder();
ArrayList tmpList = new ArrayList();
XmlNodeList entities = query.SelectNodes("//semantos:entity",
     namespaceManager);

foreach (XmlNode n_entity in entities)
{
     builder = new StringBuilder(" SELECT ");
     XmlNodeList attributes =
          n_entity.SelectNodes("//semantos:attribute",
               namespaceManager);
     foreach (XmlNode n_attribute in attributes)
     {
          builder.AppendFormat("?{0} ",
               n_attribute.Attributes["name"].Value);
     }

     builder.Append(" WHERE { ");
     XmlNodeList conditions =

n_entity.SelectNodes("//semantos:filter/semantos:condition",
               namespaceManager);
     foreach (XmlNode n_condition in conditions)
     {
          string _subject =
          (n_condition.Attributes["subject"].Value.IndexOf(":")==-1)
          ? ("?" + n_condition.Attributes["subject"].Value) :
          (n_condition.Attributes["subject"].Value);
```

```
                string _predicate =
                (n_condition.Attributes["predicate"].Value.
                IndexOf(":")==-1)
                ? ("?" + n_condition.Attributes["predicate"].Value) :
                (n_condition.Attributes["predicate"].Value);

                string _object =
                (n_condition.Attributes["object"].Value.IndexOf(":") == -1)
                ? ("?" + n_condition.Attributes["object"].Value) :
                (n_condition.Attributes["object"].Value);

                builder.AppendFormat("{0} {1} {2}.",
                     _subject, _predicate, _object);
            }
            builder.Append("}");

            XmlNode n_source = n_entity.SelectSingleNode("semantos:source",
                namespaceManager);

            XmlNode n_ontology =
                n_entity.SelectSingleNode("semantos:ontology",
                namespaceManager);

            tmpList.Add(new QueryPart(n_source.Attributes["uri"].Value,
                n_ontology.Attributes["uri"].Value,
                prefixes + builder.ToString()));
        }

        return (QueryPart[])tmpList.ToArray(typeof(QueryPart));
}
public struct QueryPart
{
    public string rdf;
    public string rdfs;
    public string query;

    public QueryPart(string rdf, string rdfs, string query)
    {
        this.rdf = rdf;
        this.rdfs = rdfs;
        this.query = query;
    }
}
```

Figure 68 Semantos to SPARQL code example

6.2. Query Enhancement Service

The Semantic Web is built up of Resource Description Framework (Brickley and Guha 2000) tags that associate data by building a semantic or concept graph. The RDF tags build this graph by specifying subject-predicate-object triples. Anyone publishing information in RDF format is free to use any subjects, predicates or objects that they wish. There is no "master list" of standard elements that may be used, as it is simply impossible to formulate such a standard list. Everyone is therefore free to create their own RDF vocabulary or ontology. However, while the publishing community may enjoy endless freedom, the consumers of this published information experience endless problems! Without a detailed knowledge of the ontology that the information is expressed in, it is generally difficult to construct a query against the information. When considering small established communities in the Web, the problem is manageable – arguably everyone writing about tropical fish could agree on a single ontology that describes their little corner of the Web very well, but what about the entire World Wide Web? It would most certainly not be possible to construct a single, all-encompassing ontology. The diverse set of ontologies existing and yet to be created are therefore nevertheless indispensable, and it is the task of the information retrieval system to make do with what it has at hand.

When a person makes a request from a modern search engine for something as simple as "tropical fish" hundreds of thousands of results will typically be returned. These results are fortunately returned in an order determined by the number of "appearances" on the web, in other words, how many times the specific page is linked to or referenced. Therefore, the topmost link should be the most frequently accessed page of its kind on the Web. This does not imply that the specific page returned is the one desired; it is merely the result of a best effort approach from the search indexer to retrieve the information required. Fortunately, this best effort is usually more than sufficient. If this same approach can be distilled and applied to the Semantic Web and its growing list of RDF data sources, it would be possible to search and index RDF triples in a similar manner. If, for example, a RDF triple equivalent of the search engine could be created, it would be possible for a software agent to query this RDF search engine and find the "most used" RDF triple, describing the concept it wishes to search for. It would then get far more search results by using the "most used" triple element, instead of its own.

6.2.1. QUERY ENHANCEMENT

What exactly is query enhancement? When querying information on a network as large as the web, query enhancement could fall into one of two categories, viz.

- optimizing the efficiency of the query in terms of speed and resource utilization, or
- enhancing the quality of the results retrieved by the query.

These goals may also only be achieved by altering the structure or makeup of the query, as it would be extremely challenging to change the structure of data across a large distributed network of indeterminate nodes. This use case is concerned with the optimization of the quality of the results. Specifically, it attempts to increase the volume of returned results, by adapting elements of the query to be in line with what is most commonly used on the web. In terms of the Semantic Web: context graph edges will be replaced with edges that are used more regularly on the Semantic Web.

6.2.2. AGENT ENABLEMENT

The process of query optimization is relatively simple. On the user end of the process the following steps occur, as illustrated in Figure 69. The query originator, which may be a person or a piece of software, builds a Semantos query that would retrieve the desired results. This query is then loaded into a software agent capable of traversing the Semantic Web. The agent starts looking for RDF data sources and executes its queries against those sources. At some indeterminate stage, the agent may visit a query enhancement service. This may be a deliberate response to not achieving a satisfactory volume of results from the first few data sources, or it may be by happenstance, as the agent may just so happen to pass by a service in any case. Regardless of when the agent visits the service, the Query Enhancement Service (QES) subsequently modifies the agent's query so as to yield better results. After being enhanced, the agent moves along to the following data sources, and perhaps even some more query enhancement services. When the agent has completed its run, it returns to the query originator with its payload.

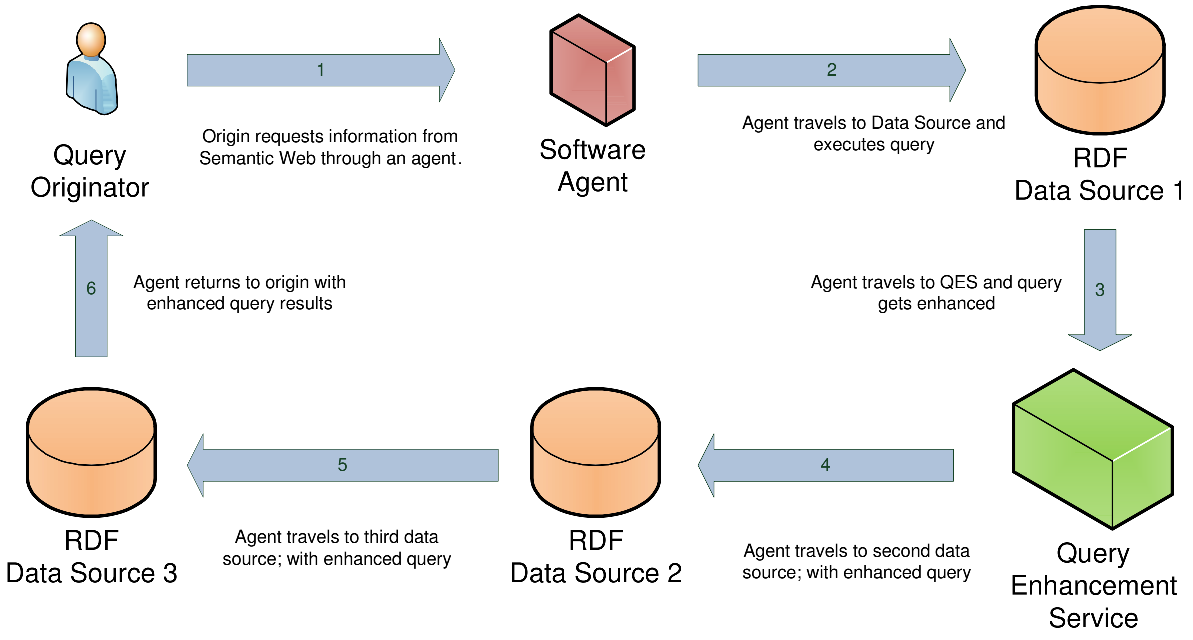

Query Originator
1
Origin requests information from Semantic Web through an agent.
Software Agent
2
Agent travels to Data Source and executes query
RDF Data Source 1
6
Agent returns to origin with enhanced query results
Agent travels to QES and query gets enhanced
3
RDF Data Source 3
5
Agent travels to third data source; with enhanced query
RDF Data Source 2
4
Agent travels to second data source; with enhanced query
Query Enhancement Service

Figure 69 Software agent enhanced through query enhancement service

6.2.3. Crawling the Semantic Web

Query enhancement attempts to replicate the successes of search engines on the web by reproducing key elements of these search engines. Prime amongst these is the ability of search engines to "crawl" the web and harvest information about websites. One of the most important pieces of information that is harvested in this fashion is the amount of web links which reference the particular page. From this amount, it is possible to ascertain the popularity of the particular page, and therefore produce more relevant search results. We duplicate this behaviour by crawling the Semantic Web and counting the number of times and individual predicate or subject occurs on any of the Semantic Web documents. By counting the occurrences of individual elements in the Semantic Web, it is possible to determine trends as to which mark-up elements are favoured. This information is then stored by the query enhancement service.

The next step is to determine the "likeness" or similarity of elements in the compiled/harvested dictionary. Several algorithms and techniques have been proposed for addressing this problem, and all are from the ontology mapping or alignment domain. Examples of techniques include Anchor-PROMPT (Noy and Musen 2003), GLUE (Doan, Domingos and Halevy 2003) and Quick Ontology Mapping (QOM). In this thesis, string similarity is used to measure the similarity of two elements on a scale from 0 to 1 (Maedche and Staab 2002) based on Levenshtein's edit distance (Levenshtein 1966).

$$distance(c, d) := \max\left(0, \frac{\min(|c|, |d|) - \mathrm{ed}(c, d)}{\min(|c|, |d|)}\right)$$

Once the similarity between elements and the occurrence probability in the World Wide Web, have been determined all the information necessary for the enhancement of queries, will have been obtained.

6.2.4. Enhancement

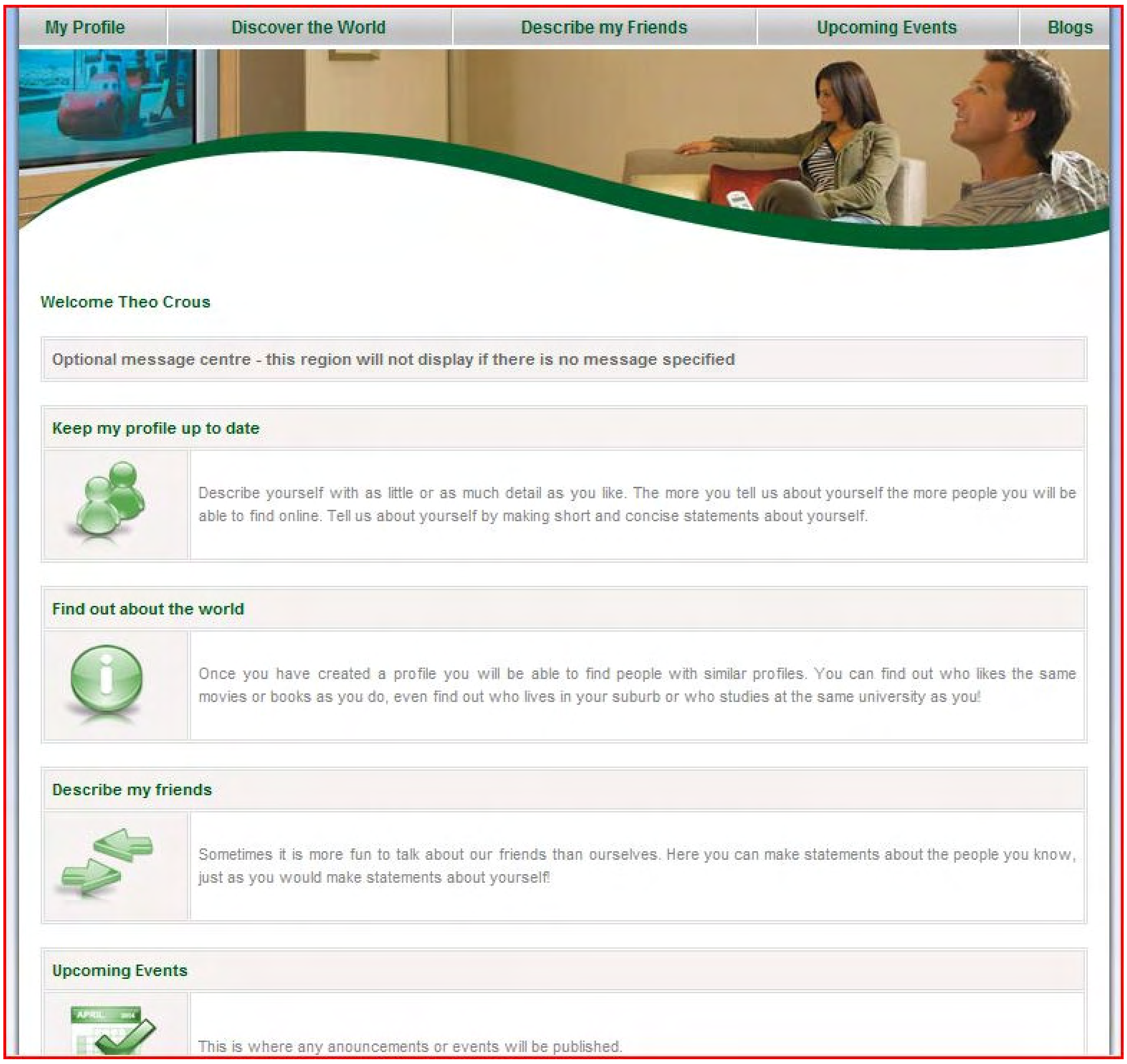
Once a query is submitted to the enhancement service, the service starts by replacing the predicate values of the triple elements. It is also possible to replace the subject and object elements, although experimentation has suggested that the results from queries enhanced in such a manner are a little untrustworthy. The predicate values are compared to the values contained in the dictionary (compiled by crawling the web). Although it would be far more effective to use statistical methods to compare elements, a simpler solution exists. Replacement is proposed as a function of the occurrence ratio in the wild, multiplied by the similarity index provided by Levenshtein's edit distance. This gives a weighted "appropriateness" value for each value in the dictionary, which allows the best selection to be made based on the highest value.

$$weight(c, d) \coloneqq distance(c, d) \times occurence(d)$$

Using LINQ, it is then a simple matter to replace the value of the predicate attribute with the most suitable value; which translates into replacing the context graph edge with a more appropriate edge. This operation is repeated for each of the triple elements in the query. After the query has been modified, it is returned to the requester; who, in turn, is now free to execute the query with the knowledge that the predicates used in the queries context graph, occur with some regularity on the web.

6.2.5. Implementation

The practical implementation of query enhancement requires a sandbox approach to test the ideas, as it would be rather unworkable to implement such an experiment across the entire web. The sandbox chosen for the purpose of this thesis reflects the current trend of social networking in Web 2.0. The system is tested against a custom social network implementation, called Who R U, shown in Figure 70. The system allows a user to register and then provide details about him or her self. This is achieved by making statements about oneself, such as "I like fish", which translates to a subject (I) predicate (like) and object (fish). It is then possible for the user to further describe the object in question at various levels of detail, for example "I like fish", becomes "I like (tropical) fish".



Welcome Theo Crous

Optional message centre - this region will not display if there is no message specified

Keep my profile up to date

Describe yourself with as little or as much detail as you like. The more you tell us about yourself the more people you will be able to find online. Tell us about yourself by making short and concise statements about yourself.

Find out about the world

Once you have created a profile you will be able to find people with similar profiles. You can find out who likes the same movies or books as you do, even find out who lives in your suburb or who studies at the same university as you!

Describe my friends

Sometimes it is more fun to talk about our friends than ourselves. Here you can make statements about the people you know, just as you would make statements about yourself!

Upcoming Events

This is where any anouncements or events will be published.

Figure 70 Who R U Interface

Who R U, provides the user with a blank canvas to publish as much (or as little) information about themselves as he or she may wish. It is then possible for other people to query this information, perhaps to find out who else likes "tropical fish", and start a chat group.

As Figure 71 points out, there are four main elements to this application. At the very top of the layer architecture, is the Who R U website. This is the main graphical user interface that allows users to interact with the system. The website would also be described as the view component in a model-view-controller (MVC) architecture. The website interacts with the query enhancement service, which is a web service. Implementing the QES as a web service opens its functionality up for other applications to run on top of it, without any difficult integration- or glue code. Both the web site and the web service make extensive use of the query engine library, which is a dynamically linked library (DLL). The query engine provides the routines, algorithms and internal data structures which enable Semantos to execute queries against RDF data sources. At the bottom of the stack are the RDF documents. Each RDF document represents a single person and all the statements he has made of himself.

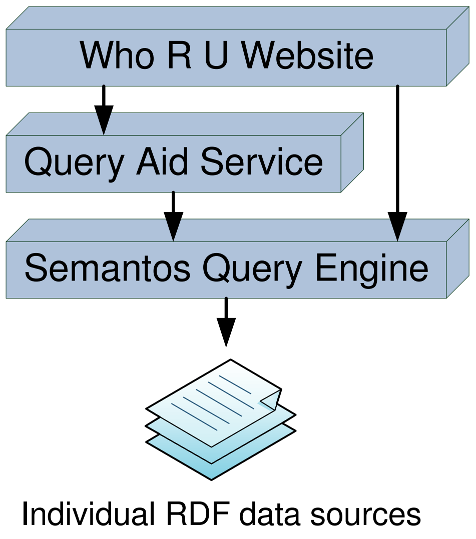
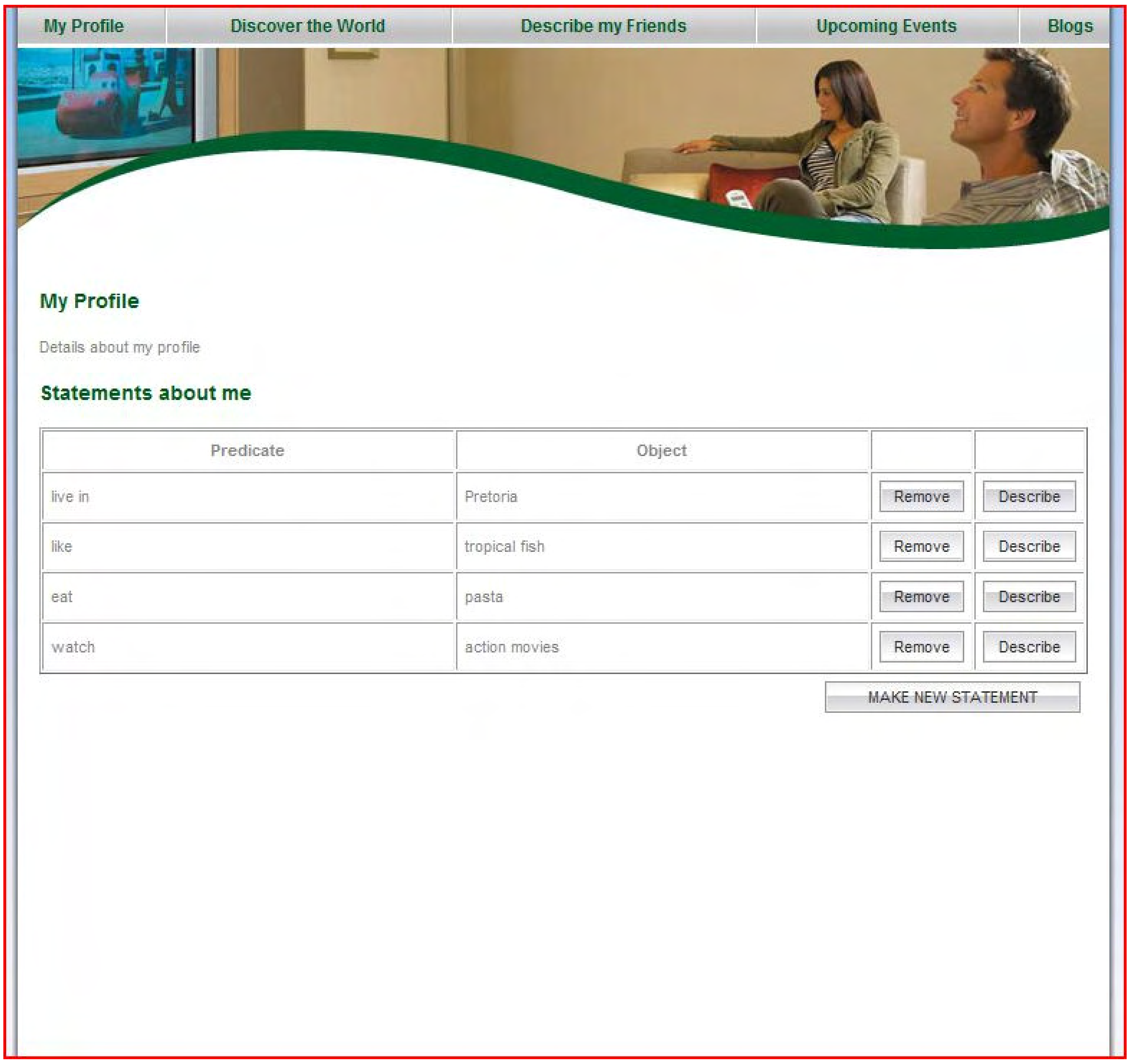


Figure 71 Who R U layer architecture

Although this test-bed application does not have a software agent implementation to execute the searches, it is easy to see the possibility of implementation. Each RDF document would represent a different data source; which, in this case, just so happens to be in the same physical location. Also, the Semantic Web crawler is omitted for the same reason – the RDF documents are all in the same location, extracting and counting the edge information in the RDF context graphs is simply a matter of iterating through the documents in a folder. In a real live application however, these RDF documents would be distributed all over the Web, and an agent and Web crawler would both be required in order to get to the information. This test-bed application, again shown in Figure 72, does serve as a suitable and interesting general implementation, so that for the purposes of this thesis theoretical ideas may be experimented with accordingly.



My Profile

Details about my profile

Statements about me

Predicate	Object		
live in	Pretoria	Remove	Describe
like	tropical fish	Remove	Describe
eat	pasta	Remove	Describe
watch	action movies	Remove	Describe

Figure 72 Making statements about yourself

6.2.6. SYNONYMS

As this implementation is concerned with written statements or comments, it would be prudent to modify the element matching function slightly, to incorporate lexical synonyms for words. A dictionary would, for example, classify the words "love" and "adore" as synonyms, and the words would therefore be considered a good match. In these instances the calculated distance between the words are forced to the maximum value of one, yielding a modified weight function as below:

$$weight(c,d) := max(distance(c,d), synonym(c,d)) \times occurence(d)$$

Now, given that people have different vocabularies (read ontologies), and that Who R U allows users complete freedom to use any word they like, it is quite likely that people will use different words that essentially mean the same thing. Someone may, for example, have commented about themselves: "I dig tropical fish", using "dig" as a synonym for "like". If this particular user were to look for people who also like fish, he would probably unknowingly, use the term "dig", which, as he may be the only person to phrase his interest in such a manner, perhaps yield poor results. This is where the QES comes into play, by changing the word "dig" into "like". This replacement would not yield all the possible results, as it is a best effort approach, meaning that the users who similarly stated that they "dig" fish will be omitted, but the users who "like" fish will be returned. This is a better result, as more people will be returned by a query looking for people who "like" fish.

Table 3 below illustrates the use of the synonym function with respect to the distance function. From this table it may be gathered that, when a synonym is present, the weight of the value is solely determined by the occurrence column. Please also note that Z represents the Levenshtein distance between the word "dig" and the word in the value column. It is also noteworthy that the index table uses the word, "dig", as a synonym for itself. What this boils down to is, that this application favours synonyms with high occurrence values over words that look similar.

Value	Occurrence	Distance	Synonym	Weight
Dig	320	Z	1	320
Like	1288	Z	1	1288
Enjoy	896	Z	1	896
Dug	2	Z	0	Z x 2
Digger	38	Z	0	Z x 38

Table 3 Possible replacement values for query using "dig"

6.3. SUMMARY

This chapter provided two distinct use cases that exemplify the unique implementation possibilities that Semantos offers. These examples were chosen to illustrate the versatility of Semantos and to justify the introduction of another RDF query language. The first example showed the possibility of using Semantos as an intermediary language between existing RDF languages, thereby strengthening Semantos's credibility as a language for integration applications. The second use case investigated the use of Semantos as a query enhancement service, whereby a search engine like functionality is enhanced with stored semantic knowledge about key phrases and RDF triples.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

"We can lick gravity, but sometimes the paperwork is overwhelming." -- Wernher von Braun

In this chapter we will state the conclusions that may be made from the research and experiments conducted. These conclusions will also indicate the strengths and weaknesses of the language as discovered from experimentation and design. In the future work section the following research steps are indicated. This future work does not only go toward establishing a "wish list" for Semantos, but indicates towards a vision of establishing a functioning and usefull semantic language capable of integration across a broad spectrum of data sources.

7.1. CONCLUSIONS

In this research, we found that the query languages for Enterprise Information Integration systems rarely provide sufficient functionality related to the leveraging of semantic information stored in corporate data stores. Due to this shortcoming it is more often than not very difficult to build queries related to the semantics of information. This is especially true if different data sources in the same coporate structure employs different semantics. In order to address this shortcoming in EII systems we have designed and implemented an information query language called Semantos. Given the criteria specified in the introduction to this research we evaluated the performance of Semantos and found it to be a suitable query language for Information Integration as it is an XML based RDF query language. We also demonstrated the applicability of Semantos using two realistic examples: a query enhancement service and a query translation service. Both cases clearly illustrated the ability of a Semantos query to be manipulated by automated software services to achieve key Information Integration goals.

In the final analysis, the Semantos language possesses the following strengths; which make it a strong candidate for any EII query language:

- Built on XML
- Serializable
- Human and machine interpretable
- Supports semantics

7.1.1. BUILT ON XML

This makes it a simple matter to parse, extend and work with the language. The XML base of Semantos makes it a very open and accessible query language. With more advanced XML processing technologies (like XLINQ) becoming available, working with XML becomes a logical choice.

7.1.2. SERIALIZABLE

For any web language it is critical for the query to be serializable. This endows a language with the strength to travel effortlessly over the web, as is very often required. This also provides an additional benefit regarding the language’s role as an EII query language, as it is very often necessary to save and later analyze the results of a query.

7.1.3. HUMAN AND MACHINE INTERPRETABLE

In the modern age man and machine are collaborating more than ever to achieve goals and/or tasks. Semantos possesses great flexibility in the fact that it can be constructed, modified and interpreted by both people and computers. This versatility contributes to the virtual symbiosis between man and machine.

7.1.4. SUPPORTS SEMANTICS

In an information rich environment, there is often a mismatch in meaning between two or more different data sources. This is more apparent on the World Wide Web than anywhere else. If there should be any hope of achieving even partial analysis of these heterogeneous data sources, it becomes critical for the semantics of these data sources to be made concrete and available.

For all its strengths, Semantos also suffers from a few shortcomings. These shortcomings are, paradoxically, a direct result of the strengths provided above. Therefore it would be necessary to manage these shortcomings, as they cannot be mitigated all together. These weaknesses include:

- Bulky
- Possibly too complicated for query optimization
- Not a backed standard

7.1.5. BULKY

As a direct result of the XML nature of the language, queries tend to be longer than languages that make use of custom syntax. This weakness can be overcome by using tools to process the language instead of hand crafting queries, which may become confusing if the query is longer and more complex.

7.1.6. POSSIBLY TOO COMPLICATED FOR QUERY OPTIMIZATION

Given that the Semantos query may be split up into different segments and distributed over the web for processing in different domains, to yield a single result, it is possible for the queries to be too complex for any form of useful optimization to take place. This is rather a result of the distributed and heterogeneous nature of the data sources, than of the query language. However, as this is the only environ in which the language will operate, it becomes an innate part of the language itself.

7.1.7. NOT A BACKED STANDARD

In a global landscape it becomes essential for any new technology, if it is to achieve any success, to be backed by a standard. Languages like XQuery and SPARQL are backed by the World Wide Web Consortium and that gives these languages a distinct advantage over Semantos. This is a weakness that may hopefully be overcome with time.

7.2. FUTURE WORK

Although this thesis has covered much ground with regard to the definition and creation of the information query language Semantos, there is still a great deal of work remaining. Several unexplored branches of research still need to be investigated more fully. Some of these branches will be highlighted below, so as to suggest to the reader the future direction which Semantos may take:

7.2.1. FULLY COMPLIANT PROCESSING ENGINE

Although several parts of the Semantos language processing code have been completed and have had some level of success at parsing and executing queries, the implementation needs to be taken further. Parts of the language processing code where reduced in functionality and one or two pieces have been left out all together. Although not key to the specification and definition of the language, which is the goal of this thesis, it would be vital for the continued level of research into the domain of the Semantos query language for the programming of a fully compatible and compliant query processing engine.

7.2.2. USE CASE SCENARIOS

In this thesis, two possible use case scenarios where the immediate benefits of Semantos could be seen have been presented. In order for the language to grow in maturity, and to solidify and benchmark the usefulness of an XML based information query language, more implementations and use cases are required. It would also be pertinent to determine how Semantos then measures up against other query languages. Measuring the levels of success when attempting the same problem with another language may also provide valuable insight into this existing query language and would benefit the Semantic Web community as a whole.

7.2.3. UPDATES

It is important to note that none of the established RDF query languages support any form of data modification or update syntax. Semantos also does not support update queries at this stage, although it is possible to implement the syntax, with ease, as XML is being used. Implementation of the update query processing itself, however, might be much more involved. It would therefore be very beneficial if this work could elaborate on possible update structures.

7.2.4. QUERY OPTIMIZATION

No work has been completed towards measuring the efficiency of the Semantos processing engine. As such, it is impossible to indicate how successful it would be when querying distributed or very large data sources. An equally important consideration would be to investigate possible query optimization techniques, as these would become important when scaling the language to a global data source network.

7.2.5. TRANSPORT PROTOCOL

Although it would be simple to suggest a standard transport protocol for a Semantos query, there may be a need to define a custom protocol. The SPARQL query language defines both a query language and a transport protocol for the web. Investigation into the usefulness of such a protocol may yield some efficiency improvements which may be made with regard to transporting data across the web.

REFERENCES

- Abiteboul, S, and A Bonifati. “Dynamic XML documents with distribution and replication.” *ACM SIGMOD Conference.* San Diego: ACM, 2003. 527-538.

- Baldassarre, M, D Caivano, and G Visaggio. “Enterprise Information Integration Management System (EII_MS).” *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR’05).* Washington: IEEE Computer Society, 2005. 192.

- Berners-Lee, T, J Hendler, and O Lassila. “The Semantic Web.” *Scientific American*, May 2001. 34–43.

- Berners-Lee, T, W Hall, and Hendler. *A Framework for Web Science (Foundations and Trends(R) in Web Science).* Hanover: Now Publishers Inc, 2006.

- Berners-Lee, Tim. *Notation 3: A readable RDF syntax.* 9 March 2006. http://www.w3.org/DesignIssues/Notation3.html (accessed June 22, 2008).

- Bierman, G, E Meijer, and W Schulte. “The essence of data access in C Omega.” *ECOOP 2005 - Object-Oriented Programming, 19th European Conference.* Glasgow: Springer, 2005. 287-311.

- Boag, S, and D Chamberlin. *XQuery 1.0: An XML Query Language.* 2007. http://www.w3.org/TR/xquery (accessed August 18, 2007).

- Bonifati, A, and S Ceri. “Comparative analisys of five XML query languages.” *ACM SIGMOD Record, 29(1)*, March 2000. 68-79.

- Braumandl, R, and M Keidl. “ObjectGlobe: Ubiquitous Query Processing on the Internet.” *VLDB Journal: Very Large Data Bases, Volume 10, number 1*, 2001. 48-71.

- Bray, T, J Paoli, C Sperberg-McQueen, E Maler, and F Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* 29 September 2006. http://www.w3.org/TR/2006/REC-xml-20060816 (accessed August 18, 2007).

- Brickley, D, and R Guha. *Resource Description Framework (RDF) Schema Specification 1.0.* 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327 (accessed August 18, 2007).

- Broekstra, J. “Sesame RQL: a Tutorial.” *www.openrdf.org.* 10 February 2004. http://www.openrdf.org/doc/rql-tutorial.html (accessed August 23, 2007).

- Broekstra, J, A Kampman, and F van Harmelen. “Sesame: An Architecture for Storing and Querying RDF Data and Schema Information.” In *Semantics for the WWW*, by D Fensel, J Hendler, H Lieberman and W Wahlster. MIT Press, 2001.

- Carroll, J, and P Stickler. “RDF Triples in XML.” *Extreme Markup Languages 2004®: Proceedings.* Montréal, 2004.

- Choi, B, M Fernandez, and J Simeon. *The XQuery formal semantics: A foundation for implementation and optimization.* May 2002. http://citeseer.ist.psu.edu/choi02xquery.html (accessed August 18, 2007).

- Christophides, V, G Karvounarakis, and I Koffina. “The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware.” *Proceedings of the First International Workshop on Semantic Web and Databases (SWDB), Co-located with VLDB 2003.* Berlin, 2003. 381-393.

- De Laborda, C, and S Conrad. “Querying Relational Databases with RDQL.” *Berliner XML Tage*, 2005. 161-172.

- Delen, D, D Nikunj, and B Perakath. “Integrated modelling: the key to holistic understanding of the enterprise.” *Communications of the ACM Volume 48, Issue 4*, April 2005. 107-112.

- Ding, L, T Finin, A Joshi, R Pan, S Cost, Y Peng, P Reddivari, V Doshi, and J Sachs. “Swoogle: A Search and Metadata Engine for the Semantic Web.” *Proceedings of the 13th ACM international conference on Information and Knowledge Management.* Washington: ACM, 2004. 652-659.

- Doan, A, P Domingos, and A Halevy. “Learning to match the schemas of data sources: A multistrategy approach.” *VLDB Journal 50*, 2003. 279-301.

- Frasincar, F, G Houben, R Vdovjak, and P Barna. “RAL: An Algebra for Querying RDF.” *World Wide Web*, 2004. 83-109.

- Giachetti, R. “A framework to review the information integration of the enterprise.” *International Journal of Production Research*, 2004. 1147-1166.

- Greco, G, S Greco, and I Trubitsyna. "Optimization of bound disjunctive queries with constraints." *Theory and Practice of Logic Programming*, November 2005. 713-745.

- Grust, T, S Sakr, and J Teubner. "XQuery on SQL hosts." *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB).* Toronto: VLDB Endowment, 2004. 252-263.

- Gutierrez, C, C Hurtado, and A Mendelzon. "Formal aspects of querying RDF databases." *Proceedings of First International Workshop on Semantic Web and Databases.* Berlin, 2003. 293-307.

- —. "Foundations of Semantic Web Databases." *ACM Symposium on Principles of Database Systems (PODS).* Paris: ACM, 2004. 95-106.

- Haase, P, J Broekstra, A Eberhart, and R Volz. *A Comparison of RDF Query Languages.* 2004. http://www. aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf (accessed August 18, 2007).

- Halevy, A, N Ashish, D Bitton, M Carey, D Draper, J Pollock, A Rosenthal and V Sikka. "Enterprise information integration: successes, challenges and controversies." *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.* Baltimore: ACM, 2005. 778-787.

- Hauch, R, A Miller, and R Cardwell. "Information intelligence: metadata for information discovery, access, and integration." *Proceedings of the 2005 ACM SIGMOD international conference on management of data.* Baltimore: ACM, 2005. 793-798.

- Heflin, J, D Dimitrov, A Qasem, and N Wang. "Information Integration via an End-to-End Distributed Semantic Web System." *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006.* Athens: Springer, 2006. 764-777.

- International Organization for Standardization. "ISO/IEC 9075:2003, Information technology - Database languages - SQL." 2003.

- Karvounarakis, G, and Magkanaraki. "Querying the Semantic Web with RQL." *Computer Networks, 42(5)*, 2003. 617-640.

- Karvounarakis, G, and V Christophides. *The RDF Query Language (RQL).* 18 July 2003. http://139.91.183.30:9090/RDF/RQL/ (accessed June 22, 2008).

- Karvounarkis, G, S Alexaki, V Christophides, D Plexousakis, and M Scholl. "RQL: A declarative query language for RDF." *In Proceedings of the Eleventh International World Wide Web Conference.* 2002. 592-603.

- Klyne, G, and J Carrol. *Concepts and Abstract Syntax.* 2004. http://www.w3.org (accessed August 18, 2007).

- Levenshtein, I.V. *Binary codes capable of correcting deletions, insertions, and reversals. Cybernetics and Control Theory.* 1966.

- Madnick, S. "From VLDB to VMLDB (Very MANY Large Data Bases): Dealing with Large-Scale Semantic Heterogeneity." *Proceedings of the 21st VLDB Conference.* Zurich: Morgan Kaufmann Publishers Inc, 1995. 11-16.

- Maedche, A, and S Staab. "Measuring similarity between ontologies." *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW).* Siguenza: Springer, 2002. 251-263.

- Maier, D. "Database desiderata for an XML query language." *QL'98 – The Query Language Workshop.* Boston, 1998.

- Manolescu, I, D Florescu, and D Kossmann. "Answering XML queries over heterogenous data sources." *Proceedings of the 27th International Conference on Very Large Databases.* Roma, 2001. 241-250.

- Meijer, E, B Beckman, and G Bierman. "LINQ: Reconciling Objects, Relations and XML in the .NET." *SIGMOD 2006.* Chicago: ACM, 2006. 706.

- Meijer, E, M Torgersen, and G Bierman. "Lost In Translation: Formalizing Proposed Extensions to C#." *OOPSLA 2007.* Montreal: ACM, 2007. 479-498.

- Meijer, E, W Schulte, and G Bierman. "Programming with Circles, Triangles and Rectangles." *Proceedings of the XML 2003 Conference.* Philadelphia, 2003.

- —. "Unifying Tables, Objects and Documents." *Proceedings of DP-COOL 2003.* Uppsala, 2003.

- Melton, J. "SQL, XQuery, and SPARQL: Whats Wrong With This Picture?" *XTech 2006:Building Web 2.0.* 2006.

- Melton, J, and S Muralidhar. *XML Syntax for XQuery 1.0 (XQueryX), W3C Working Draft.* 2005. http://web4.w3.org/TR/2005/WD-xqueryx-20050404 (accessed August 18, 2007).

- Microsoft. "DryadLINQ." *http://research.microsoft.com/en-us/projects/dryadlinq/.* May 2006. http://research.microsoft.com/en-us/projects/dryadlinq/xlinq_overview.doc (accessed November 12, 2008).

- Miller, L, A Seaborne, and A Reggiori. "Three Implementations of SquishQL, a Simple RDF Query Language." *The Semantic Web - ISWC 2002.* Sardinia: Springer-Verlag, 2002. 423-435.

- Nilsson, Mikael. *Cetis.* 6 September 2001. http://zope.cetis.ac.uk/content/20010927172953/#foot173 (accessed June 22, 2008).

- Noy, N.F, and M.A Musen. "The PROMPT suite: interactive tools for ontology merging and mapping." *International Journal of Human-Computer Studies 59*, 2003. 983-1024.

- Onose, N, and J Simeon. *XQuery at your web service.* 2004. http://citeseer.ist.psu.edu/onose04xquery.html (accessed August 18, 2007).

- *Open Directory RDF Dump.* 04 05 2004. http://rdf.dmoz.org/ (accessed 08 13, 2008).

- Prud'hommeaux, E, and A Seaborne. *SPARQL Query Language for RDF.* 14 June 2007. http://www.w3.org/TR/rdf-sparql-query (accessed August 18, 2007).

- Shadbolt, N, T Berners-Lee, and W Hall. "The Semantic Web Revisited." *IEEE Intelligent Systems 21(3)*, 2006. 96-101.

- Sikka, V. "Data and metadata management in service-oriented architectures: some open challenges." *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.* Baltimore: ACM, 2005. 849-850.

- Vdovjak, R, F Frasincar, G Houben, and P Barna. "Engineering semantic web information systems in hera." *Journal of Web Engineering 2(1&2)* (Rinton Press), 2003. 3-26.

- *World Wide Web Consortium.* 2007. http://www.w3.org/ (accessed August 18, 2007).

- Zillner, S, and W Winiwarter. “Integrating ontology knowledge into a query algebra for multimedia meta objects.” *Proceedings of Web Information Systems - WISE 2004, 5th International Conference on Web Information Systems Engineering*. Brisbane, 2004. 629-640.

APPENDIX A: XML SCHEMA

The full and formal XML schema for the Semantos query structure is provided here, in Figure 73, in order to aid future development of Semantos applications. This schema provides the fully detailed specification so as to resolve any remaining ambiguity regarding the XML structure of a Semantos query.

```
<?xml version="1.0" encoding="utf-8"?>
 <xs:schema
  xmlns:semantos="http://www.retrorabbit.co.za/semantos/schema"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://www.retrorabbit.co.za/semantos/schema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element
  name="fetch">

  <xs:complexType>

   <xs:sequence>

    <xs:element
     name="source"
     minOccurs="1">

     <xs:complexType>

      <xs:attribute
       name="name"
       type="xs:string"
       use="required" />

       <xs:attribute
        name="uri"
        type="xs:string"
        use="required" />
     </xs:complexType>

    </xs:element>

    <xs:element
     name="ontology"
     minOccurs="0">

     <xs:complexType>

      <xs:attribute
       name="name"
       type="xs:string"
       use="required" />

      <xs:attribute
       name="uri"
       type="xs:string"
       use="required" />

     </xs:complexType>

    </xs:element>

    <xs:element
     maxOccurs="unbounded"
     name="namespace"
     minOccurs="0">

     <xs:complexType>

      <xs:attribute
       name="name"
       type="xs:string"
       use="required" />

      <xs:attribute
       name="uri"
       type="xs:string"
       use="required" />
     </xs:complexType>

    </xs:element>

    <xs:element
     name="entity">

     <xs:complexType>

      <xs:sequence>

       <xs:element
        maxOccurs="unbounded"
        name="attribute">

        <xs:complexType>

         <xs:attribute
           name="name"
           type="xs:string"
           use="required" />

        </xs:complexType>

       </xs:element>

       <xs:element
        name="graph">

        <xs:complexType>

         <xs:sequence>

          <xs:element
           maxOccurs="unbounded"
           name="triple">

           <xs:complexType>

            <xs:attribute
```

```
             name="subject"
             type="xs:string"
             use="required" />

            <xs:attribute
             name="predicate"
             type="xs:string"
             use="required" />

            <xs:attribute
             name="object"
             type="xs:string"
             use="required" />

          </xs:complexType>

        </xs:element>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

  <xs:element
   name="filter">

    <xs:complexType>

      <xs:sequence>

        <xs:element
         name="condition">

          <xs:complexType>

            <xs:attribute
             name="attribute"
             type="xs:string"
             use="required" />
```

```
                <xs:attribute
                 name="operator"
                 type="xs:string"
                 use="required" />

                <xs:attribute
                 name="value"
                 type="xs:string"
                 use="required" />

               </xs:complexType>

              </xs:element>

             </xs:sequence>

            </xs:complexType>

           </xs:element>

          </xs:sequence>

          <xs:attribute
           name="name"
           type="xs:string"
           use="required" />

         </xs:complexType>

        </xs:element>

       </xs:sequence>

      </xs:complexType>

     </xs:element>

</xs:schema>
```

Figure 73 Full Semantos schema

APPENDIX B: EXAMPLES

Presented in this appendix are several examples of Semantos queries. They are provided here in order to improve the reader's understanding of the language constructs.

1. First Example

The first example, Figure 74, queries a company's data stores to find all the employees that work under a specific manager. What is interesting about this example is that it has multiple data source documents, one for the employees and another for managers. It also makes use of two namespaces, providing shortcuts to the XML namespaces regarding people and workplaces.

```
<semantos:fetch>

 <semantos:source
  name="employees"
  uri="http://www.rr.co.za/data/employees.rdf"/>

 <semantos:source
  name="managers"
  uri="http://www.rr.co.za/data/managers.rdf"/>

 <semantos:namespace
  name="people"
  uri="http://www.rr.co.za/data/humans#" />

 <semantos:namespace
  name="workplace"
  uri="http://www.rr.co.za/data/workplace#" />


 <semantos:entity
  name="employees">
```

```
<semantos:attribute
 name="fullname"/>

<semantos:attribute
 name="number"/>

<semantos:attribute
 name="salary"/>

<semantos:graph>

 <semantos:triple
  subject="person"
  predicate="people:fullname"
  object="fullname"/>

 <semantos:triple
  subject="person"
  predicate="workplace:ispaid"
  object="salary"/>

 <semantos:triple
  subject="person"
  predicate="workplace:code"
  object="number"/>

 <semantos:triple
  subject="manager"
  predicate="workplace:manages"
  object="person"/>

 <semantos:triple
  subject="manager"
  predicate="people:surname"
  object="surname"/>

</semantos:graph>

<semantos:filter
 type="and">
```

```
   <semantos:condition
    attribute="surname"
    operator="eq"
    value="Crous" />

  </semantos:filter>

 </semantos:entity>

</semantos:fetch>
```

Figure 74 First Semantos example query

The attribute elements specify which nodes of the context graph are to be projected to form the result set. Figure 75 illustrates this relationship between the graph nodes and the attributes. The filter provided for this query specifies the condition that the surname of the manager entity must be equal to "Crous". This query will therefore return the full names, numbers and salaries of all employees that are managed by anyone with the surname "Crous".

fullname	number	salary
Peter Kane	008339	100 000
Ben	008376	120 000
Sally Strut	008329	80 000

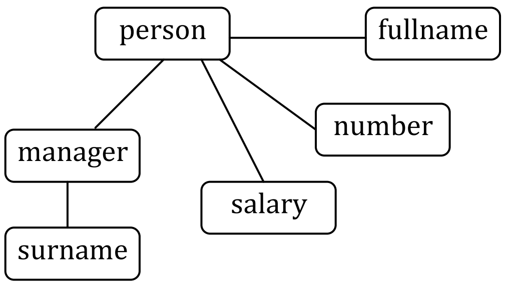


Figure 75 Projected columns from contextual graph for Example 1

2. Second Example

The second example, Figure 76, queries a comedians' list of categorized jokes. In this particular query we are looking for all jokes that are suitable for a specific audience with specific tastes. In this example we introduce some more complex filters, which include nested filters and conditions. This query also illustrates the fact that ontologies and namespaces are optional.

```
<semantos:fetch>
 <semantos:source
  name="jokes"
  uri="http://www.rr.co.za/data/jokes.rdf"/>

 <semantos:entity
  name="jokes">

  <semantos:attribute
   name="name"/>
  <semantos:attribute
   name="description"/>

  <semantos:graph>

   <semantos:triple
    subject="joke"
    predicate="joke-name"
    object="name"/>

   <semantos:triple
    subject="joke"
    predicate="joke-description"
    object="description"/>

   <semantos:triple
    subject="joke"
    predicate="joke-category"
    object="category"/>

   <semantos:triple
    subject="joke"
    predicate="joke-age"
    object="agerestriction"/>

 </semantos:graph>
```

```
  <semantos:filter
   type="and">

   <semantos:condition
    attribute="agerestriction"
    operator="le"
    value="16" />

    <semantos:filter
     type="or">

     <semantos:condition
      attribute="category"
      operator="eq"
      value="oneliner" />

     <semantos:condition
      attribute="category"
      operator="eq"
      value="knockknock" />

    </semantos:filter>

  </semantos:filter>

 </semantos:entity>

</semantos:fetch>
```

Figure 76 Second Semantos query example

This query is very simple in the sense that it does not have a very complex context graph. It also only projects two of the graphs' nodes, namely the joke's name and description. The filter constructs are more interesting and illustrate several filter element features of Semantos. The filter above can be expressed as follows:

$$agerestriction \leq 16 \text{ AND } (category = \text{oneliner OR } category = \text{knockknock})$$

3. Third Example

Example 3 provides more insight into the ontological strengths of Semantos, as seen in Figure 77. In this query we will illustrate the use of ontologies by querying a pet shop data store for all pets suitable for being kept in a small apartment. In this query we also make use of the “in” operator for filtering the results returned.

```
<semantos:fetch>

 <semantos:source
  name="pets"
  uri="http://www.rr.co.za/data/pets.rdf"/>

 <semantos:namespace
  name="pet"
  uri="http://www.rr.co.za/data/pets#" />

 <semantos:ontology
  name="people"
  uri="http://www.rr.co.za/ont/humans.rdfs#" />

 <semantos:entity
  name="pets">

  <semantos:attribute
   name="name"/>

  <semantos:graph>

   <semantos:triple
    subject="pet"
    predicate="pets:name"
    object="name"/>

   <semantos:triple
    subject="pet"
    predicate="canLive"
```

```
     object="indoors"/>

  </semantos:graph>

   <semantos:filter
    type="and">

    <semantos:condition
     attribute="size"
     operator="in">
      <value>tiny</value>
      <value>small</value>
      <value>medium</value>
    </condition>

   </semantos:filter>

  </semantos:entity>

</semantos:fetch>
```

Figure 77 Third Semantos query example