Exploratory Semantic Query and Annotation Application for Smart Buildings

Michael Poulton
B.A.I. Engineering
Final Year Project April 2012
Supervisor: Dr. David Lewis

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

______________________________________________        ________________________
Name                                                  Date
Abstract
This dissertation aims to document a program capable of acquiring data regarding electronic devices within a “smart home” or “smart office” and annotating a 3D representation of the place in question as appropriate, thus building up a base of information for the user regarding their home or workplace, with, ideally, a minimum of effort on their part to procure and assemble the data. To achieve this, queries to the Semantic Web are employed. The Semantic Web is a tool which would receive little use by the casual user, due to poor accessibility, and that knowledge of the Semantic Web’s existence is not particularly widespread – it is still very much in its infancy. Further, it is a system intended to provide data which can be ‘understood’ and manipulated primarily by machines, as well as then producing an output comprehensible to a human.

In this case, keeping in mind the difficulty which the Semantic Web may pose to the casual user, the particular goal is to provide an easy-to-use system with which to perform queries, without requiring any specialised knowledge on the part of the user. The user interacts with this system via the 3D view of a building mentioned above – in this case powered with WebGL, and thus browser-based – with built-in options to annotate areas. When the user selects a region to annotate, they will receive a prompt to enter a simple text query (such as they would with a conventional search engine), and when that is carried out, the result returned – the system’s “best guess” at what the item in question might be – is noted on the 3D view.
## Contents

Abstract ........................................................................................................................................... 3
Contents ............................................................................................................................................... 4

Chapter 1: Introduction .................................................................................................................... 5
  1.1 Background and motivations ................................................................................................. 5
  1.2 Objectives ................................................................................................................................ 6

Chapter 2: State of the art ................................................................................................................. 8
  2.1 Overview .................................................................................................................................. 8
  2.2 The Semantic Web .................................................................................................................... 8
  2.2 RDF (Resource Description Framework) .................................................................................. 11
  2.4 SPARQL ................................................................................................................................... 13
  2.5 Sesame, and other frameworks ............................................................................................. 14
  2.6 Sindice ..................................................................................................................................... 17
  2.7 WebGL ..................................................................................................................................... 18
  2.8 Other technology used ............................................................................................................ 19

Chapter 3: Design ............................................................................................................................. 20
  3.1 Overview .................................................................................................................................. 20
  3.2 Framework .............................................................................................................................. 20
  3.3 Servlet container ..................................................................................................................... 20
  3.4 Query language ....................................................................................................................... 21
  3.5 Accessing Sindice ................................................................................................................... 22
  3.6 Cautionary note on setting up project ...................................................................................... 22
  3.7 Graphical interface .................................................................................................................. 23
  3.8 The core application ............................................................................................................... 23

Chapter 4: Implementation and results ............................................................................................. 25
  4.1 Overview .................................................................................................................................. 25
  4.2 Software and systems used ...................................................................................................... 25
  4.3 Implementation of the Java application .................................................................................. 28

Chapter 5: Conclusions .................................................................................................................... 32
  What I learned ............................................................................................................................... 32
  What was achieved and produced ............................................................................................... 33
  What future work might achieve .................................................................................................. 33
  The feasibility of the Semantic Web as a whole, and its use for the casual user ...................... 34

References .......................................................................................................................................... 36
Chapter 1: Introduction

1.1 Background and motivations

The Semantic Web is a gradually growing system, but still very much in its infancy. The internet at present is centred on the idea of providing information which is human-readable, but there is little of any real use to a computer which seeks to acquire and process information – for example, to take data from multiple sources, and correlate and compare it, and compile it according to some set instructions, for use by a human. This is due to the bulk of data presently stored on the internet being marked-up in HTML, and set-up in such a way that, while a human user will have no problem reading the output, there is little of use for a computer to extract; where the Semantic Web is concerned, however, the machine is dependent on the use of RDF, or Resource Description Framework, a mark-up language intended to provide data in such a way that the computer may access it as it would a typical database.

There is, however, an obstacle to the casual user: the Semantic Web is difficult to use correctly and will not, for the casual user, provide any useful information beyond what they would obtain in a typical internet search. The goal here is therefore to remove some of that difficulty, and to permit a user to employ the Semantic Web without having to understand it, but rather through use of a simple and intuitive interface - in this case, integrated with a system designed to produce 3D representations of building interiors, to produce a system suitable for a ‘smart home’.
Initially, the project was inspired by the work of a postgraduate student in Trinity College Dublin, in a dissertation entitled “Harvesting Home Area Network Knowledge from the Web” (Emer Duffy, 2011), and that dissertation was used as a sporadic point of reference in the course of my own project and dissertation. There is something of a distinction, however, in that the system which I have now worked on requires more user input than the previously mentioned dissertation, which was aimed at a more automated system.

1.2 Objectives
The end goal for this project is to produce a system which has the potential to:

- Gives users an interactive view of their home or office space – the production of this 3D scene fell beyond the scope of this project, and a 3D graphical view already produced was employed and integrated, although ideally, it should be possible to link in any graphical representation of any place with a minimum of changes required to the program.

- Provide functionality with which to annotate that graphical view.

- Use that same interface to permit a user to query the Semantic Web without requiring them to do any more than type a search term or terms into a simple pop-up or prompt that is available from the graphical interface. The results returned to the user should, ideally, be a concise and useful description of the object which they are curious about.

The bulk of the work which has to be done there rotates around the portion of the system intended to query the Semantic Web – the production of the graphical interface falls
outside the scope of this, although mention will be made of how one might potentially integrate the querying system with the graphical interface. The dissertation will detail:

- The technology which is presently available to assist with the creation of such systems.

- In general terms, without excessively specific reference to any one set of utilities, but with suggestions for what might be used, how one might go about the creation of such a system.

- Further to the above, the specific approach which was taken in this particular case, along with the problems encountered, and how the completed system performed.

- Finally, in conclusion, details regarding how the system might be improved upon were future work to be performed on it. The conclusion also has a note regarding the possible history of the Semantic Web as a whole.
Chapter 2: State of the art

2.1 Overview
The following section will detail the technologies available for use in Semantic Web applications, and the ones which I chose to use in particular – any application which employs the Semantic Web will require a considerable quantity of ancillary software and external programs to run. Thus, I will discuss each of them, and make particular mention of which parts of each available system were used extensively.

2.2 The Semantic Web
The Semantic Web is a term created by Tim Berners-Lee, the director of the World Wide Web Consortium – or W3C – and, according to W3C’s website, it:
“... refers to W3C’s vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.”
(http://www.w3.org/standards/semanticweb/)

As opposed to the Web as it presently is in most cases, where sites are set-up with the intention of being human-readable and accessible to the average user, sites created with the Semantic Web in mind will either expand their functionality such that a machine may access the data available as it would a data store, or perhaps even have the site set up solely for the use of Semantic Web technologies.
The Semantic Web is dependent on creators of websites and anybody uploading data to the Web understanding and employing the various standards which are used. A machine cannot analyse data which is not laid out in a suitable format. The primary format used, the Resource Description Framework, or RDF, is discussed below. Of course, the need for this standardisation is one of the issues which constrains the Semantic Web and prevents it from expanding as rapidly as it might. Creating a Web page which is suitable for addition to the linked data cloud and for usage in Semantic Web applications is not something which the casual user will typically be able for, or be interested in doing – it requires understanding of mark-up languages beyond just HTML, and that they be used properly, as well. Thus, the Semantic Web has grown far more slowly than the Internet as a whole.

This limitation – namely the lack of accessibility to the average user, and that there is often little apparent reason for the casual user to create a page suitable for Semantic Web applications – means that the amount of data available in the linked data cloud pales in comparison to that on the Internet as a whole. The quantity available is constantly expanding, of course, as knowledge of the Semantic Web disseminates further, and the number of computer-literate users who may be willing to contribute to it increases. However, at the time of writing, it must be noted that the amount of data which can be reliably extracted is limited in quantity – relatively few websites are marked up for Semantic Web applications, although the number does expand constantly.
Another key idea of the Semantic Web is that, as mentioned, of linked data. www.w3.org gives an example of linked data as follows:

“A typical case of a large Linked Dataset is DBPedia, which, essentially, makes the content of Wikipedia available in RDF. The importance of DBPedia is not only that it includes Wikipedia data, but also that it incorporates links to other datasets on the Web, e.g., to Geonames. By providing those extra links (in terms of RDF triples) applications may exploit the extra (and possibly more precise) knowledge from other datasets when developing an application”

Ideally, with linked data, one would be able to look at a dataset available on the Semantic Web, and investigate links leading to another dataset, rather than simply raw data available, with no reference to any other data which may have a bearing on the topic. I found this particular concept to be of unfortunately limited use in the course of my own work, instead preferring to analyse raw data directly, without exploiting the linked data concept to any extent – this shall be discussed further in the implementation section.

Since the Semantic Web is, at least in theory, a vast database of linked data and with pages in specific formats – namely RDF – there is a requirement for specific means by which to query it, and to properly extract information. These are detailed below – SPARQL is the primary one available for use. Additionally, Sindice shall be discussed – Sindice is a service which describes itself as “the Semantic Web index”, and is of great use where searching for data on the Semantic Web is concerned.
2.2 RDF (Resource Description Framework)

According to [www.w3.org](http://www.w3.org), RDF is:

“… a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.”

And further that:

“The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web.”

It takes a similar approach to older systems designed for the modelling of information, and is centred on what are known as “triples”. These triples consist of a subject, a predicate and an object. The subject is typically the entity being described, and, where Semantic Web applications are concerned, the subject is very often a URI (a uniform resource identifier) – which, in Semantic Web applications, will itself generally be a web address i.e. a URL. The predicate describes which particular property the subject in question has, which is presently being described. The object, finally, is the property which the subject has.
RDF is suitable for usage by machines to retrieve and process data, where other data formats may result in the loss of information in the process – RDF is designed such that it is standardised, and that data may readily be exchanged between applications.

Figure 1 - example of how RDF functions, with the green ovals representing subjects, the boxes being objects, and arrows the URIs – taken from http://www.w3.org/TR/rdf-primer/
2.4 SPARQL
SPARQL is a query language developed specifically for use with Semantic Web applications, able to perform queries over data stored in RDF format, and to perform operations on any data retrieved. Since RDF data is standardised with regard to how it is formatted and structured – and assuming no user error when putting the data in question in place, of course – a single SPARQL query should be sufficient to retrieve sets of data from multiple sources without needing to adjust it to account for the source being different.

SPARQL is the most widespread language for querying RDF data, as well as the one recommended by the W3C. It is not, however, the only one available – for example, Sesame, which is discussed below, supports both SPARQL and its own query language, SeRQL. However, since SPARQL is by far the most widespread, best supported and most thoroughly documented, it was chosen.
2.5 Sesame, and other frameworks
Where the core application was concerned, I wanted to use Java, since it’s a programming language which I’m familiar with. However, a suitable framework was required to build upon to properly employ the Semantic Web. The framework chosen was Sesame 2 (the distinction between Sesame and Sesame 2 is important, although since Sesame 2 has largely superceded Sesame by this point, the term ‘Sesame’ shall be used from this point onwards to refer to Sesame 2), which is described as:

“Sesame is a de-facto standard framework for processing RDF data. This includes parsing, storing, inferencing and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.” (www.openrdf.org/about.jsp)

The Sesame API, as described, proved to be easy-to-use, and was rather expansive, providing more than ample tools with which to perform the above-described operations. It was selected, specifically, since it is thoroughly documented, recommended by other Semantic Web related sources, and was compatible with Java applications.

Sesame provides ample support for performing SPARQL queries, as well as fetching and manipulating RDF data. Further to this, it supports its own query language, known as SeRQL. Similarities exist between SPARQL and SeRQL, and while SeRQL receives more thorough documentation and would appear to be the preference of the producers of Sesame, more information about SPARQL can be located from other sources, and it is the de facto standard for querying RDF data, even if alternatives are provided.
Alternatives to Sesame do, however, exist. Jena – available at http://incubator.apache.org/jena/ - is one particularly notable one, and possibly as widely used as Sesame. Sesame was the framework which I deemed to be better documented, however, and it was recommended by and noted as being compatible with Sindice (the Semantic Web index, which shall be discussed in more detail below). Further readily-available and continuously-updated alternatives are not available for Java, specifically; while other attempts have been made to provide a suitable framework, they are either particularly specific (http://kaon.semanticweb.org/ describes itself as being oriented towards business applications in particular, for example), or, in another case, for example, development has been discontinued (http://jrdf.sourceforge.net/); it is vital that a framework such as this be kept up-to-date with current technologies, with the constantly changing and developing state of the Semantic Web.

Sesame, notably, is not a monolithic piece of software, and while it runs, as advertised, largely ‘out-of-the-box’, certain provisions must be made to enable this, such as the use of an appropriate Java servlet container, as described in section 2.7. Since it is not monolithic, the appropriate pieces must be picked and used, although this is a sufficiently simple process that it requires little more than the addition of the appropriate .jar file to a project’s classpath.
The internal structure of Sesame and how it interacts with external sources bear some further mention. From [http://www.openrdf.org/doc/sesame2/users/ch03.html](http://www.openrdf.org/doc/sesame2/users/ch03.html):

- **RDF Model**
  - **Repository API**
    - **SailRepository**
    - **HTTPRepository**
  - **Sail API**
  - **Rio**
  - **HTTPClient**
- **application**
- **HTTP Server**

![Diagram of Sesame layers](http://www.openrdf.org/doc/sesame2/users/ch03.html)

The above diagram illustrates the layers involved in Sesame. Since Sesame is centred on RDF, the RDF model is therefore the base of the framework. The brief descriptions provided by www.openrdf.org are as follows: “Rio, which stands for "RDF I/O", consists of a set of parsers and writers for various RDF file formats”, and “The Storage And Inference Layer (Sail) API is a low level System API (SPI) for RDF stores and inferencers.” Then, and potentially of the most interest, and thus quoted in full here to provide an idea as to the full functionality of Sesame:

“The Repository API is a higher [sic] level API that offers a large number of developer-oriented methods for handling RDF data. The main goal of this API is to make the life of application developers as easy as possible. It offers various methods for uploading data files, querying, and extracting and manipulating data. There are several implementations of this API, the ones shown in this figure are the SailRepository and the HTTPRepository. The former translates calls to a Sail implementation of choice, the latter offers transparent client-server communication with a Sesame server over HTTP.”
2.6 Sindice

Sindice is described by itself as ‘the Semantic Web index’. It functions as a means by which to lookup documents which are available on the Semantic Web. It is further described as follows:

“Sindice is a platform to build applications on top of this data [the data available from the Semantic Web]. Sindice collects Web Data in many ways, following existing web standards, and offers Search and Querying across this data, updated live every few minutes. Specialized APIs, and tools are also available.”

Sindice offers a SPARQL end-point, though when accessed it does warn users that “this is very beta!” A SPARQL endpoint is not a new idea – another exists for DBpedia, for example (see: [http://dbpedia.org/sparql](http://dbpedia.org/sparql)) – and provides functionality with which to perform SPARQL queries directly over the data which Sindice can retrieve. The Sindice APIs also provide considerable functionality where conducting searches and queries over the Semantic Web is concerned.
2.7 WebGL
WebGL is a system used to produce in-browser 3D graphics, and may be summed up as:

“WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0” (http://www.khronos.org/webgl/)

At its core, it is built upon JavaScript applications, to run in the browser. The server hosting the WebGL application may need to have particular libraries installed to run effectively, but a person accessing the application remotely need only have an appropriate browser – notably, Internet Explorer is not supported by WebGL, although other prominent browsers such as Firefox, Chrome, Safari and Opera are capable of running it. Further, the user’s computer must be capable of running WebGL – any reasonably modern machine should be able to deal with this. Thus, WebGL may be viewed as being somewhat portable, although it certainly has limitations, not least since Internet Explorer still holds a majority share with regard to the usage of browsers.

As long as the graphics system being used allows for the easy exchange of data between it and a Java application – even if that is confined to text strings – then any graphical program would be suitable. There is no particular requirement to use WebGL; in this case, it is what was to hand, with programs to base the system off already produced – I was using an amended version software already produced and available as of 1 April 2012 at http://www.scss.tcd.ie/~mcglinnk/webGL/SimCon_Threejs_f35_test.html - further details are available in the implementation section.
2.8 Other technology used
As was mentioned, a substantial quantity of ancillary software is required to produce and successfully run a Semantic Web application. While the specifics of what was chosen and ultimately used for this particular implementation shall be discussed in the appropriate section below, here is a more concise listing of the particular pieces of ancillary software employed:

• NetBeans was used as the IDE to develop the program.

• Apache Maven (see: http://maven.apache.org/) is a tool used to assist with the management and organisation of large projects. It should be noted that it does require a degree of registry modification to run, although should also be compatible with certain IDEs.

• Apache Tomcat, as the Java servlet container; as mentioned, a necessity to operate Sesame and the Semantic Web querying system.

• While they are included with the standard Sesame download, assorted Apache httpcomponents were also required, as well as slf4j (the Simple Logging Façade for Java), and if a user wished for some particularly advanced or new functionality available with this, they may be interested in searching for specific downloads above and beyond what is packaged with Sesame.
Chapter 3: Design

3.1 Overview
This section shall endeavour to cover, in reasonably general terms, such that it may be replicated with no need for specific pieces of software or code, the composition of an appropriate application and system to query the Semantic Web, as well as suggestions regarding the integration of such a system with a graphical interface.

3.2 Framework
Firstly, a suitable framework must be chosen with which to work, which may to an extent be dependent on the user’s programming language of choice. Employing Sesame as a framework is a recommendation of the Semantic Web community, and by extension Java is then the programming language which should be used. Jena is a suitable alternative. Caution should be taken to ensure that if a different set-up is chosen, that it is capable of interfacing properly with the available Sindice APIs, which are integral to this particular system, and, in this case, one of the few pieces which cannot readily be amended – Sindice is a relatively unique system, presently (there are mash-up generators available, such as at http://sig.ma/ although that itself uses Sindice as one of its sources – and while it mentions http://www.okkam.org/, that project has been discontinued).

3.3 Servlet container
Additionally, and assuming that Sesame in particular is chosen – since other frameworks may very well have their own requirements in this regard, or built-in systems which do not require any such addition – the system will require a Java servlet container to operate. These are readily available, easy to set-up, and once installed and basic configuration is
completed, may be ‘forgotten’ about to some extent; the program is present, runs, and barring any errors in the set-up process, needs no further configuration, and holds little bearing on the system beyond that initial set-up. If the intention is to use one of these, and the particular system does happen to require one – Sesame will – then note should be taken of port numbers in particular. Alternatively, a remote server (a server running Apache Tomcat, for example) is of course an option, although for testing purposes, and ascertaining the feasibility of the system, keeping solely to the localhost may prove to be advantageous, both for the improved speed, and removing a potential source of errors.

3.4 Query language
A suitable query language must be selected. This is again going to be dependent on the framework selected, and particular requirements. SPARQL is an ideal choice, considering that it is the recommendation of W3C, and the de facto standard where the usage of the Semantic Web is concerned. If a programmer wishes to use another query language, there is also the possibility that it may not be updated to keep in mind; for as long as SPARQL is deemed the standard, it is more likely to be kept up-to-date, even as other query languages begin to fall out of use, and similarly as the Semantic Web expands, other related resources are very likely to be catered towards SPARQL as opposed to other query languages. Furthermore, SPARQL is more thoroughly documented than anything else presently available.
3.5 Accessing Sindice
Accessing Sindice in some shape or form is almost imperative for this project, and one distinctly inflexible choice. If a programmer so wished, they may be able to exploit sig.ma to a similar end, although it lacks the readily-available APIs which are provided by Sindice, and I, at least, can see little reason for why one would wish to deviate from Sindice, for as long as it is available.

Sindice does provide a SPARQL endpoint for those interested, although, as warned, it is very much in a Beta stage, and is often unstable and unreliable, as well as being somewhat limited. It remains an option, however. An alternative is to use the number of Sindice APIs which have been developed; these can be exploited to take the data which a search on Sindice would produce, and make use of it for a Java (or other suitable programming language) application. However it is accessed and the information retrieved, Sindice is a vital component of this system.

3.6 Cautionary note on setting up project
Even if an IDE is used, and I highly recommend that it is, the project is of such a scale and contains so many different components and external resources that it may be beneficial to use a system such as Apache Maven to help with the organisation of the project, or whatever other project management system is deemed suitable for a particular case. The system can easily go awry with Java, for example, if a file in the classpath is moved or deleted, and the exact problem may not always be immediately obvious. Addressing this
issue at the beginning may save some considerable time that might otherwise be spent debugging the program.

3.7 Graphical interface
Assuming that the intention is, as with here, to provide an easy-to-use interface which conceals much of the underlying program, and to make the Semantic Web accessible to a user, then a graphical interface should be a consideration. While WebGL was used in this particular case, any graphical program and visual representation of a building should suffice.

However, it will obviously require that the programmer add to it functionality such that it may communicate with the application which is actually going to conduct the querying, both to send queries, and to retrieve results, and annotate positions in the scene appropriately. This portion of the system is relatively standalone, and it should, ideally, be possible to shift around between various graphical programs and set-ups with relative ease; the bulk of the communication which will occur with the rest of the system is in the form of text strings. As long as this communication is enabled, any manner of graphical interface conceivable may be used.

3.8 The core application
Once the above choices have been made, and the appropriate set-up done, then the core application – that is to say, the part which conducts the queries and filters out the data as appropriate – is at a point where work can be started on it in earnest.
In this particular case, and as a recommendation for working in a general case, whatever the framework or other details may be, the program should function as follows:

- Firstly, once the application is started, and running, it should link to the graphical interface, and be prepared to receive and process queries from the interface.

- When the application receives a query from the interface, hopefully in the form of a simple text string, the Sindice APIs should then be employed to conduct a search and to retrieve a list of URLs from Sindice for potentially relevant information. Since the number of results returned from Sindice may potentially be vast, it may be prudent to limit the number of results which is extracted from Sindice to a certain quantity, in order to increase the efficiency of the program.

- Take the retrieved URLs and the data therein as raw datasets, or otherwise, then conduct queries on them to determine various properties, depending on what exactly is required; for example, common terms used to describe aspects of an item may be of interest. Care should be taken with being too liberal with deciding that data may be of use, or simply because the mark-up language has a tag indicating that it describes one thing or another – it may very well be of little relevance to what is actually sought.

- Take the system’s “best guess” at what the item in question may be, and return that to the graphical interface as output. It may also be stored in a local file or database, if so desired by the programmer.

There are clearly numerous ways in which one might expand upon this. The specific approach taken here is detailed in the implementation section.
Chapter 4: Implementation and results

4.1 Overview
Following on from the design section, this section shall document the specific implementation, the decisions made, and the problems encountered in the course of creating the system.

4.2 Software and systems used
This sub-section will document the pieces of software which were used, and why, as well as the various existing systems that were employed. The various pieces of ancillary software described were initially rather difficult to locate and to set-up. Simply getting the whole system to reach a point at which it was ready to run as a cohesive whole took quite some time to set-up. That in and of itself is enough to raise concerns about the portability of such
a system as this; it would require further development to be available in a suitable ‘release mode’.

I initially made the decision to create the program using Java since it was a language which I was familiar with and reasonably proficient with from previous work. Fortunately, much of the technology presently available for the Semantic Web, where the creation of applications which can access it is concerned, is created with Java in mind. Further to this, I used the NetBeans IDE – from simple familiarity and personal preference, although presumably any Java IDE would suffice.

Having made the decision to use Java, a framework was then required which was suitable for the production of Semantic Web applications. Sesame (http://www.openrdf.org/) was suggested, and seemed to be eminently suitable to the task at hand. Jena was an alternative. Jena, however, did not seem to have as much readily available documentation as was present for the Sesame framework. Additionally, after further searches for information regarding the Semantic Web, it appeared that other resources – for example, Sindice in particular – were tooled towards the usage of Sesame in particular. While setting up Sesame, the Sail Native RDF jar file was including in the listing of dependencies (being a Maven project, this is much the same as adding it to the classpath in a typical project), and for purposes of logging, the slf4j-simple file.
Since Sesame requires the usage of a Java servlet container, and following the suggestion provided with the Sesame user guide, a copy of Apache Tomcat (http://tomcat.apache.org/) was procured and set up. Notably, if this is run on Windows systems, it should be set up in such a way that it is run as an administrator, otherwise considerable difficulties may be encountered when running the program.

With the system as a whole beginning to become somewhat ungainly where the quantity of classpath additions and apparently necessary ancillary software was concerned – a substantial amount of time was wasted seeking and trying to fix problems which were caused by an incorrectly set-up classpath, or files in it being moved around – I shifted the entire project to an Apache Maven system. Apache Maven does require some legwork to set-up. The NetBeans IDE contains an option, with Maven present, to set-up a project as a Maven project. .jar files may then be added to the dependencies for the project via the idea, and monitored rather more easily than is possible with a typical project. One downside of Maven is that the build and compile times for projects are slightly longer than they might otherwise be.

After testing on my own part, to use the Sindice APIs, I discovered a project named sindice4j (http://code.google.com/p/sindice4j/) – or Sindice for Java. It provides a number of Java libraries for querying Sindice – it enables searches through the use of a Java application such as are otherwise carried out on the Sindice site, through the Sindice search tool; this is distinct and different, however, from queries to the sites in question.
4.3 Implementation of the Java application

Bearing in mind that the above-mentioned portions had been carried out, or were conducted in the course of the project, before it was completed (Apache Maven, for example, was not present from the start, but as mentioned, was instead added to aid with problems which I was experiencing related to the classpath), the construction of the Java application was then relatively straightforward.

As detailed in the design section, the Java application had three primary sections. The first of these was centred on using the Sindice APIs to extract a listing of URLs from Sindice which would be relevant to the topic at hand. After initial attempts, the set of Java libraries for Sindice mentioned above came to light, and I instead employed those to perform searches over Sindice. My primary and lingering criticism of those libraries is that the search is rather slow; whereas Sindice searches performed directly through the Sindice site never take longer than a matter of tenths of a second, no matter the quantity of search results, searches performed using the set of libraries here and the APIs are unfortunately orders of magnitude slower. This meant that there was a need to limit the number of results that would be retrieved; otherwise the application would run so slowly as to be entirely unfeasible. Numbers of results ranging up to 100 would operate acceptably quickly (often in the region of 1 second), although there were drastic drops beyond that, with attempts to procure as many as 500 URLs taking upwards of 10 seconds. With the method which I finally settled on, I was able to obtain ‘acceptably’ accurate end results with as few as 40 addresses being considered.
The next stage was, as mentioned in the design stage above, query the URLs which had been retrieved using the Sindice APIs to locate specific data. Here was where the Sesame framework really became necessary, along with the various methods that it provided, and the ability to perform queries via Java.

This particular system sought to find terms within the description of an item which were similar – while deliberately excluding some excessively common words, with as assorted conjunctions, the definite article etc. – and to see how many instances there were of that term appearing. While testing was not especially rigorous, the system nevertheless proved to be able to obtain and pass back some basic information about an item, based off perhaps a vague description. Serial numbers were particularly helpful; if a number for an item were to be searched for, the results would be far more concise, and the queries over the results were able to pick up certain common terms with greatly increased frequency – given a serial number, the system was generally able to identify the manufacturer, any other particular names associated with that number, and what type of device it was.

Unfortunately, as a side-effect of the system constructed, sometimes irrelevant or useless data would slip in. In the case of retailers, this was a particular issue, as well as where sites were in languages other than English (in which case, requiring that the language tag indicate that it was English sufficed). Besides potentially tightening up the bounds and restrictions on the queries, an alternative way to prevent this from occurring would be if it were possible to
expand the number of URLs being queried without the system becoming unreasonably slow. I was not able to achieve this, so a number of my results in the end – while many results were accurate – also contained irrelevant pieces of data, such as sporadically retailer’s names, or other brands being mentioned. As is reflected on in the conclusions, a more solid grasp on queries and on how to approach such problems may assist, although computing power and the time taken to conduct the queries as is are both still concerns. When the data was retrieved, the ‘answer’ which had been discerned was added to a text file, along with the query which had been entered to begin with.

Finally, the whole system had to be integrated with a graphical interface – the purpose stated was to produce a system to use Semantic queries to produce data with which to annotate a smart building. As was noted above, the project used an already produced system in WebGL. This was taken from the site to which it was uploaded – the source code is readily available through browsers, simply by accessing the HTML, and embedded JavaScript – and put onto my own computer. To run on the localhost necessitated the presence of the three.js libraries on the machine as well (all of this was downloaded from http://www.scss.tcd.ie/~mcglinnk/webGL/, which remained accessible as of 1 April 2012). The WebGL program was adjusted such that the query box was visible to the user upon accessing the program. Asynchronous transmission between the WebGL program and the Java query application was then conducted. The time taken to perform the queries was too great to be able to perform synchronous transmission in what I viewed as a ‘responsible’ sense – that is to say, while using synchronous transmission would not have affected the program in the majority of cases, since it is unlikely that the user would put in two queries in
a row without first waiting for an answer to one, the possibility was there, and with the several second gap between the WebGL application ‘asking’ the Java application for an answer and actually being provided with one, it seemed prudent to observe that fact. Ajax (or asynchronous JavaScript and XML) was employed here.
Chapter 5: Conclusions

What I learned
I had not previously worked on a project of such a large scale before – that is to say, a system set up with numerous different components, and not a single monolithic piece of software, nor a single Java application such as those which I had worked on leading up to this point in time. Learning how to and then also producing a system which was dependent on multiple pieces of software was new to me, and a worthwhile endeavour in and of itself – the discovery of such tools as Apache Maven helped considerably, for example, when the project had started to become somewhat unwieldy. This was a surprisingly challenging aspect of the project – gathering together and setting up all the necessary pieces of software and ensuring that they ran properly together; I did not initially expect to have such a mash-up of different systems and programs as I finally ended up with.

Having spent as much time as I did reading about it and working with it, I developed a reasonable understanding of the Semantic Web, which of course lay at the centre of the project, and the assorted tools associated with it; while in the end, little use for the linked data cloud was found, I came to have a solid understanding of that concept over time, as well. I developed a reasonable grasp on the Sesame framework, too, having centred the project on that. I cannot claim to have found SPARQL to be as versatile or as useful as I initially anticipated, however – this may indicate a lack of understanding of the query language (or of RDF) on my own part.
What was achieved and produced
With the completion of the project, I would hazard to say that presently, and with the currently available technology, it is far better viewed as presently being a proof of concept – namely, that a smart home is indeed a feasible idea, and that the Semantic Web may be used to produce useful sets of data through simple queries – rather than being a feasible program in and of itself. As is, the system is capable of extracting a limited degree of useful information from the Semantic Web – although this information is, from what testing was conducted, at least accurate in a majority of cases. However, it would not be sufficient for any manner of actual ‘smart home’; the information which the system presently produces is too limited for it to be considered useful for such a purpose. The casual user would not, in all likelihood, learn more than that which they already know from the system as is.

What future work might achieve
Were further work to be conducted on the project, preferably with a more thorough understanding of and ability with SPARQL than I presently possess – either through continued practice and learning on my part, or by another interested in continuing the work – then I would think that substantially more useful data may be extracted from each query.

This may extend, for example, to picking out the resource with the best ‘description’ of an item, and applying that to the result which is extracted from the query. Care would have to be taken, of course, to avoid the information used here being a piece taken from an online retailer, for example – many of the results procured in my own testing were often from such sources, and they would, if the project were attempting to obtain detailed descriptions of an
item, skew the results to a possibly undue extent from what they might otherwise have been were the data available to be entirely objective.

If those obstacles, amongst others, were to be overcome, future iterations of this project may also employ and exploit the linked data cloud to a greater extent than I did in my own project – linked data was largely ignored in favour of instead analysing and extracting results from raw data sets, which were in turn taken from Sindice using the APIs available there.

One might also consider that as the Semantic Web itself develops, future work may be made considerably easier. For example, the Sindice SPARQL endpoint’s functionality may be expanded beyond its presently rather limited range, or restructuring of portions of the Semantic Web may also occur. Of course, with the Semantic Web being as ever-changing as it is, that also raises the problem of ensuring that a system designed to query it is kept up-to-date, and that queries written at one point in time are concise enough that they do not become redundant with changing technology and the expansion of the Semantic Web itself.

The feasibility of the Semantic Web as a whole, and its use for the casual user
Towards the end of my time spent working on the project, the following new came to light, which may change matters considerably (article uploaded on 15 March, available as of 1 April 2012):

http://online.wsj.com/article/SB10001424052702304459804577281842851136290.html
In short, it details Google’s intentions to provide direct answers to queries in searches, as well as a list of potentially useful links. This is of course largely what I had endeavoured to achieve: to produce a system which, in response to a query, could provide an answer immediately, with no requirement for a user to sift through assorted links to achieve it. I had reflected, initially, after completing the project, that it would still be much easier for the casual user to go to Google, and to search for what they wanted to find out, and find far more reliable results there than the Semantic Web could produce for them at present. It will be interesting to see, however, how much the substantial resources of Google might further the Semantic Web, or at least the employment of Semantic technologies.

Otherwise, the above development aside, I am doubtful that the Semantic Web will see a great deal of uptake in the coming years. It remains inaccessible to the casual user, and even in cases were a user-friendly interface is designed, there is only a limited degree of information that it can provide to a casual user, and presently little real incentive to them to use it over a typical search engine. It is, of course, useful to those who wish to treat the Semantic Web and linked data cloud as the database that it is, and treat it as such – this is not the case, however, with the average user, nor with anyone who does not want to take the time to sift through the data retrieved and have to constantly refine queries for everything discerned for each question that they may have. Of course, with the investment of a large corporation which is aiming specifically at the casual user, it may all very well change.
References

*Harvesting Home Area Network Knowledge from the Web*, Emer Duffy, September 2011

Data related to RDF and Semantic Web, available as of April 1, 2012,

http://www.rdfabout.net/

Sindice for Java documentation and libraries, available as of April 1, 2012, from

http://code.google.com/p/sindice4j/

World Wide Web Consortium data, available as of April 1, 2012, from W3C: www.w3.org

OpenRDF (Sesame) information, available as of April 1, 2012, from www.openrdf.org