1 Introduction and Motivation

1.1 Introduction to The Semantic Web

The Semantic Web or "Web 3.0" is the next stage in the evolution of the World Wide Web envisioned by the World Wide Web Consortium (W3C) and Sir Tim Berners Lee to *"to create a universal medium for the exchange of data"*[1]. Superficially, this seems to be what the current web offers, but in this context, "data" refers to machine readable and understandable resources as opposed to "information" which refers to human-friendly resources such as web pages. Therefore the goal of the Semantic Web is to make the Web "machine understandable", so that programs can easily obtain and use data from websites. This is made possible by defining the semantics (meaning) of the information on web pages, which is to be done using a "linked data" model, where pieces of data are linked together to create meaningful data structures called "graphs" with the goal being to create a "Giant Global Graph" of all data.[2,3]

The implications of this are staggering; it would provide the framework for allowing web applications to compile various sources of data to create a new and interesting service. For example data could be pulled from your Social Networking page and your personal calendar and the MET office to plot the expected weather on a calendar next to your upcoming events. Alternatively systems could easily bring together various sources of scientific data to plot complicated graphs which would have previously taken a long time to research and make by hand. Such applications that pool multiple Semantic resources to deliver a new service are called "Semantic Mashups".

In order to achieve this, the human-readable information must be tagged and structured according to a protocol, one of the standards adopted to achieve this is RDF, the "Resource Description Framework". RDF is a family of standards based on XML, used to represent data in the Web as a machine readable graph of data[4]. This is a mammoth task as, for the most part, RDF data must be encoded by hand by individual webmasters, although community efforts like DBpedia[5] have built software to convert structured information from resources such as Wikipedia to RDF.

Once data has been encoded into RDF, software and users need to be able to form queries in some way so that useful or interesting information can be extracted from the RDF data. In order to do this, the SPARQL query language (recursive acronym: "SPARQL Protocol And RDF Query Language") was developed and became a W3C Recommendation in early 2008[6].

With the technology already in place, what remains is the slow transition of existent data on the Web to the new format, and the development of applications capable of taking advantage of this graph of data can begin.

1 W3C: Semantic Web Activity Overview http://www.w3.org/2001/sw/Activity.html *Accessed 02/02/2010*
2 W3C: Semantic Web http://www.w3.org/2001/sw/ *Accessed 27/4/2010*
3 Sir Tim Burners Lee's Blog at MIT: http://dig.csail.mit.edu/breadcrumbs/node/215 *Accessed 27/4/2010*
4 W3C: RDF Specification http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ *Accessed 02/02/2010*
5 DBpedia: About http://dbpedia.org/About Accessed 26/4/2010
6 W3C: Semantic Web Blog (Ivan Herman <ivan@w3.org>) http://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation *Accessed 02/02/2010*

1.2 Problems and Motivation

Data Accessibility

SPARQL is a declarative language with a syntax similar to that of SQL (Pronounced "Sequel", often called "Structured Query Language"[7]) and although simple in comparison to manually writing the queries in an imperative language such as Java, it is imposing and difficult to write for the average user. Further, when searching for information on the web, users are accustomed to natural language interfaces such as Google or Ask.com and are unlikely to switch to a system that is more difficult to learn and use. Even worse, writing SPARQL queries currently requires specialist knowledge of the target ontology, which is virtually unintelligible to anyone but experts. These factors mean that data encoded as RDF is virtually inaccessible to the end user and requires experts to learn a new Query Language to utilise.

Data.gov.uk http://data.gov.uk/

The UK government has disclosed a lot of data structured as RDF to Data.gov.uk. This data is useful but hard to extract because the interfaces are simply too complex and require familiarity with both SPARQL and RDF. A quick look at the public forums shows that even developers are having trouble accessing the data, and the general public are completely lost. Here's an example complaint:

> *"Why has the site been designed with absolutely no thought for the end user?*
>
> *Surely the whole point is to allow easier access to the Government-based data and statistics? As it stands it's a poorly designed site, which appears to be deliberately hiding its information."*
>
> http://data.gov.uk/forum/general-discussion/finding-data-hard#comment-702 (Accessed 26/4/2010)

DBpedia http://www.dbpedia.org/

As mentioned above, DBpedia is a community effort to encode Wikipedia's information as RDF data. DBpedia is more accessible than Data.gov.uk, with both a text search[8] and a simple query builder[9], but the text search doesn't unlock the power of the Semantic Web and the query builder requires knowledge of RDF predicates.

Developer Apathy

Generally developers would prefer to not have to learn a new language and new technologies to do what they want to do – anything that can generate code that they are unfamiliar with will be used extensively.

1.3 Project Goals

From the above examples we can see that developing a SPARQL query generator which can form syntactically correct SPARQL from a user friendly interface requiring little or no specialist knowledge, would be of tremendous value to the Semantic Web: If the average user, rather than just a few specialists and experts, could access the wealth of structured information then we would be a step closer to the Semantic Web being a universal medium for the exchange of data.

7 About.com "SQL Fundamentals" http://databases.about.com/od/sql/a/sqlfundamentals.htm *Accessed 13/4/2010*
8 DBpedia: Entity Search, Find, and Explore http://dbpedia.org/fct/ Accessed 26/4/2010
9 DBpedia: Query Builder http://querybuilder.dbpedia.org/ Accessed 26/4/2010

1.4 Reader's Guide

This report is split into 10 sections, each briefly described below:

- 1 Introduction and Motivation (p1)
  - This section. Brief introduction to the system and related technologies as well as the reasons behind undertaking the project.
- 2 Background (p5)
  - More information on Web 3.0 including technologies and existing systems
- 3 Design (p10)
  - Information on how the project was conceived and planned.
  - High level technical information on the structure of the project, and the project requirements.
- 4 Implementation (p20)
  - Describes how project development actually went
  - Technical walkthroughs of the completed features
- 5 Test and Evaluation (p29)
  - Information on how the system was tested and how problems were resolved.
- 6 Conclusion (p41)
  - Final thoughts on what the project achieved
- 7 Outlook (p42)
  - How the system could be further developed
- 8 References (p44)
- 9 Maintenance Manual (p46)
  - Installation Guide
  - Low level information on system components
- 10 User manual (p57)
  - How to access program features

Numbering System

Each section is assigned a number, and each subsection another number separated from the section number by a dot, so the third subsection of the Background section is numbered “2.3”. Subsection dividers are given a further number in the sequence, for example “4.2.1” represents Section 4, Subsection 2, Division 1. Subsection Headers are not given their own numbers.

Figures are given the number sequence of the passage they belong to and a letter to uniquely identify them, for example “1.2.3.a”.

1.5 Where to Find...

Information	**Sections**	**Page**
More information on Semantic Web Technologies	Background Existing Systems	5 8
Information on goals and planning	Design – Plans – Method	10 10 13
Final system features	Implementation	24
Technologies used	Architecture Client Server Data	16 16 17 18
System Internals	Architecture How it All Fits Together File Listing	16 25 51
Evaluation	Testing and Evaluation Conclusion Outlook Comparison With Existing Systems	29 41 42 39
User Guides	User Manual	57
Installation Instructions	Maintenance Manual	46

2 Background

2.1 Understanding Web 3.0 Technologies

2.1.1 RDF and Linked Data

The "Resource Description Framework" is an abstract and simple graph based data model, designed to store structured, linked data using an XML based syntax. It has been a W3C recommendation since 10th February 2004[10]. RDF breaks data up into Subject-Predicate-Object triples (Fig. 2.1.1.a), where the Predicate defines a relationship between the Subject and the Object. URI's (Uniform Resource Identifiers) are used to refer to resources. Another way of representing this data is as a graph (Fig. 2.1.1.b), with the Subject as a node, the object as a node, and the predicate being the link between the two notes.

Fig. 2.1.1.a

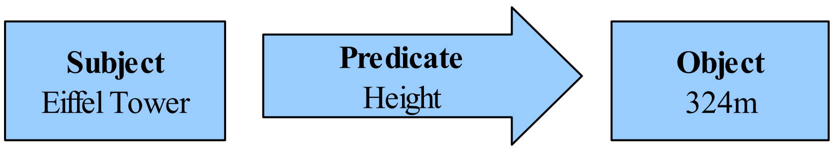
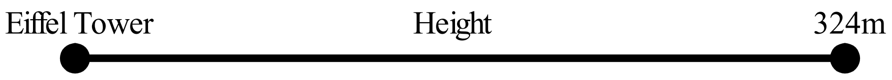


Fig 2.1.1.b



- The *Subject* is either a URI or a blank node.
- The *Predicate* must be a URI.
- The Object is either a URI, a literal value or a blank node.

RDF can be represented using XML, Notation 3 (N3) or one of several other notations. For example, the total population of Berlin, according to DBpedia[11] is 3431700, this can be represented in RDF as follows:

XML:

```
<rdf:RDF
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dbpprop="http://dbpedia.org/property/"
>

<rdf:Description rdf:about="http://dbpedia.org/page/Berlin">
         <dbpprop:PopulationTotal> 3431700 </dbpprop:PopulationTotal>
</rdf:Description>
```

N3:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
@prefix dbpprop : <http://dbpedia.org/property/>

[ http://dbpedia.org/page/Berlin dbpprop:PopulationTotal "3431700" .]
```

Both samples above define "Prefixes", which allow the use of shortened tags instead of using full URIs for each resource. In this case "rdf", the RDF definitions from the W3C website and "dbpprop", the set of properties from DBpedia are defined Prefixes, allowing "http://dbpedia.org/property/PopulationTotal" to become "dbpprop:PopulationTotal".

[10] W3C: RDF Specification http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ *Accessed 02/02/2010*
[11] DBpedia: Berlin http://dbpedia.org/page/Berlin Accessed 14/4/2010

As you can see, RDF is unambiguous but complex and so to efficiently extract data from it requires the use of a query language called SPARQL.

Ontologies

An "Ontology" (in information science) is a formal representation of data structures and concepts inside a specific domain. We need Ontologies to describe domains because knowledge is not consistent and what is true inside one domain is not true inside another. For example, imagine the domain of knowledge about cars and the domain of knowledge about planes, in cars it is true to say "The wheels are powered by the engine" but this is not true about planes.

In order to take into account these differences RDF data usually implements an Ontology (usually defined in OWL "Web Ontology Language"[sic]) that describes what data is permitted and how it is structured.

One of the major problems of searching the Semantic Web is that data sources each have their own domain and their own Ontology; so the data in Data.gov.uk is structured differently from the data in DBpedia and must be queried in a different way.

RDF Metadata Definitions

One of the advantages of RDF is that, since it is based on XML, it is extensible, and as a result several standards have been created in order to define different kinds of data for several different domains. This "data about data" is referred to as "metadata" and is used to make RDF more descriptive. Some widely used metadata definitions are listed below:

- **FOAF http://www.foaf-project.org/**

  "Friend Of a Friend" (FOAF) is a Semantic Web project based on social relationships between people, aiming to *"*[create] *a Web of machine-readable pages describing people, the links between them and the things they create and do"*[12] .

- **Dublin Core http://dublincore.org/**

  Often shortened to "DC", Dublin Core Metadata Initiative defines some simple and generic definitions for specifying resources.

- **Wordnet http://wordnet.princeton.edu/**

  Lexical database for the English language. Wordnet's ontology is also used by resources such as DBpedia.

To confuse the issue, many large RDF repositories define their own Metadata as their data does not conform to any of the major standards, so DBpedia and Data.gov.uk both have their own definitions which are not used outside of their domain specific ontologies.

For a system to query an RDF resource, it must be familiar with the metadata definitions used, as trying to query for a foaf:name attribute in a system which does not use FOAF will return no results.

[12] FOAF: About http://www.foaf-project.org/about Accessed 26/4/2010

2.1.2 SPARQL

“SPARQL Protocol And RDF Query Language” (SPARQL) is a declarative query language and protocol for extracting information from RDF data sources. It is a W3C recommendation[13], and was designed by the former RDF Data Access Working Group,now SPARQL Working Group[14]. SPARQL queries are series of clauses which define what information is desired, they are run against a data source specified by a URI, and return a result set, which is usually an XML document. SPARQL queries must contain the following components:

- PREFIX : allows shortened names to be used instead of full URIs, similar to RDF prefixes. Here the prefixes “rdf” and “foaf” are declared as well as their associated URIs.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

- SELECT: specifies which variables will be in the result set, “*” can be used to select all variables.

```
SELECT ?name
```

- FROM: the URI of the data source to be queried. In certain cases this can be omitted.

```
FROM <http://planetrdf.com/bloggers.rdf>
```

- WHERE: lists a series of constraints that narrow down the RDF data to the specific piece of information or the type of information that is wanted. Notice that each line ends with a “.”, also note that there are two variables used, but only one is selected by the SELECT statement.

```
WHERE{
?agent rdf:type foaf:Agent .
?agent rdf:type foaf:Person .
?agent foaf:name ?name .
}
```

The result set of the query above contains the object of the “name” predicate of all subjects which match the type “Agent” and the type “Person” from the resource “http://planetrdf.com/bloggers.rdf”. The data source contains the following (abridged) RDF:

```
<foaf:Agent rdf:nodeID="id2245050">
    <foaf:name>John Breslin</foaf:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    (...)
</foaf:Agent>
```

Here you can see that this data structure matches the FOAF predicates Agent and Person to match with the variable ?agent, and that it also has the predicate name with the object “John Breslin”, which will match to the variable ?name.

[13] W3C SPARQL Definition Doc http://www.w3.org/TR/rdf-sparql-query/ Accessed 14/4/2010
[14] W3C SPARQL Working Group http://www.w3.org/2001/sw/DataAccess/homepage-20080115#hist Accessed 14/4/2010

2.2 Existing Systems

Existing Systems for creating semantic queries broadly fall into two categories: SPARQL generators and question answering search engines. The SPARQL generators, in general, use SPARQL and RDF to find results, however they are in general incredibly hard to use. Question Answering Engines on the other hand tend to be very easy to use but don't actually use semantic web technologies, instead using a web crawler and text analysis on multiple web pages to extract answers.

2.2.1 SPARQL Generators

Semantic Web Search (http://www.semanticwebsearch.com/query/)

Semantic Web Search allows you to search SPARQL endpoints using a shorthand language which the query engine translates into SPARQL for you. You can also use the "Search Agent" feature to build a query from a natural language representation, starting with a very general sentence "Find any resource of any type with any property that contains any value", and slowly refining it to something which represents a more specific query. Every time you click on one of the variable parts of the sentence (underlined), you are presented with either an input where you can type or a drop down box which lists valid choices which narrow down the query.

It is important to note that the system doesn't take in a natural language string as input, and the natural language interface only serves to generate the shorthand language that it uses to generate a SPARQL query. Typing natural language into the search box will result in errors.

- ✓ Simplified syntax
- ✓ Some natural language interaction
- × Doesn't process natural language queries
- × Restricted set of ontologies, can't specify others

DBpedia Query Builder (http://querybuilder.dbpedia.org/)

This query engine is for constructing SPARQL queries to run on DBpedia. It isn't quite as unfriendly as SPARQL but it requires knowledge of the DBpedia ontology to use, making it very hard to use for the end user.

- ✓ Restricts input to more quickly form queries
- × Requires specialist knowledge
- × Intimidating interface

OpenLink iSPARQL (http://demo.openlinksw.com/isparql/)

The iSPARQL system builds queries using a graphical user interface to construct a graph. The system is graphical rather than code based, but isn't aimed at end users but rather experts who want to construct complex SPARQL queries in terms of an RDF graph. The interface has a lot of options for building complex and specific queries, and it requires a lot of technical input such as URIs, variables and types.

When I tested it I couldn't get it to work, as elements of the GUI kept sticking, adding massive amounts of nodes to the graph. I found the system frustrating and rather too complicated for creating simple queries.

- ✓ Graphically constructs complex queries
- × Requires expert level knowledge
- × Option-overload for novice users
- × Doesn't run on some browsers, including IE, runs poorly on Firefox

2.2.2 Question Answering Systems

Ask Jeeves/Ask.com (http://uk.ask.com/)

Ask Jeeves is perhaps the best known question answering service, and it even passes the "what is the height of the Eiffel Tower?" test, going so far as to list it's height including and excluding antenna, and giving an interesting fact related to it's height. However, Ask Jeeves is not a Semantic Web technology in the sense that it does not appear to use RDF graphs or any Ontologies to generate it's results, rather relying on a web crawler and text analysis of the web pages it indexes[15].

- ✓ Handles Natural Language questions.
- ✓ Retrieves accurate answers and related interesting information.
- × Can't query RDF graphs or similar structured data sources.
- × No SPARQL generation.

Wolfram Alpha (http://www.wolframalpha.com/)

Released by Wolfram who created the computational language Mathematica[16], Wolfram Alpha styles itself as a "Computational Knowledge Engine", it works by using Natural Language Processing to identify key parts of a user's question and then uses Mathematica to run a computation on it's knowledge base[17]. Like Ask.com it also knows the height of the Eiffel Tower, but it is restricted to what information is in it's internal knowledge base, and although it's knowledge base is extensive, it can't match the volume of data on the WWW.

- ✓ Handles Natural Language questions.
- ✓ Retrieves accurate answers.
- ✓ Can perform complex computations.
- × Can't query any data not on it's internal knowledge base.

[15] Some very limited details are available on the ask.com site: http://about.ask.com/en/docs/about/ask_technology.shtml Accessed 12/4/2010

[16] Wolfram Research: http://www.wolfram.com/ Accessed 12/4/2010

[17] Read Write Web "Wolfram|Alpha: Our First Impressions" http://www.readwriteweb.com/archives/wolframalpha_our_first_impressions.php Accessed 12/4/2010

3 Design

3.1 Conception

Rather than present the user with a text field expecting SPARQL input, I would like to present them with a simpler interface which would allow them to build SPARQL queries with little or know knowledge of the underlying technologies. Ideally this would be a Google-like minimalist interface, with a one line text box which accepts natural language questions such as "What is the height of the Eiffel Tower?", generates a SPARQL Query for a user selected RDF data source, such as DBpedia[18], and returns a response, such as "The height of the Eiffel Tower is 324 meters (1,064 feet)".

However Natural Language Processing is a hard goal to accomplish on top of SPARQL generation, so it was decided to concentrate on producing a system that focused on producing and running SPARQL queries from user friendly web forms.

3.2 Development Plans

It was plausible that the project would miss deadlines and that some of the goals would be unattainable, so it was decided to split the requirements and method into "Plan A", "Plan B" and "Plan C". Plan A would deliver the maximum amount of features and the more powerful system, but would rely heavily on all deadlines being reached. Plan B would still represent a powerful and useful tool, but would not include the loftier goal of translating Natural Language to SPARQL and instead would focus on providing access to government released data at Data.gov.uk, and community gathered data from DBpedia. Finally, Plan C represents the bare minimum requirements for a useful system.

3.2.1 Basic Requirements (Plan C):

Feature	Description	Priority
Functional Requirements:		
SPARQL Generation	It would be easier for both novice and advanced users to query RDF if SPARQL could be automatically generated from limited inputs.	1
Query User Defined resources	It's important that the user can define their own SPARQL endpoints to query against.	1
Display Generated SPARQL	It'd be useful for people learning SPARQL to see what the system generates.	2
Edit Generated SPARQL	Allow the user to refine SPARQL queries generated by the system.	3
Direct SPARQL Entry	It would be useful to allow direct SPARQL input, but accessibility is the top priority as many systems off this feature already.	2
Syntax checking for Direct Input	If the system allows direct SPARQL input, it would be nice if the system could check the syntax of entered SPARQL and display error information	3
Facilities to Store RDF Metadata	If the system stores commonly used metadata definitions, users won't need to know the exact RDF predicate/class information	1

[18] http://dbpedia.org/About

General Requirements:		
User Friendly Interface	In order to maximise data accessibility, the interface should be easy and intuitive to most users.	1
Browser Compatibility	Although Web sites are built with standard markup languages, different browsers render things differently, so a standard requirement for all web applications is that they run in the 3 most popular browsers, which as of April 2010 are Internet Explorer (IE), Firefox, and Google Chrome[19]	3
Speed	Queries should be executed as quickly as possible, as users expect fast results.	3

3.2.2 Plan B Requirements

In addition to the Plan C requirements:

- The system should be able to access and query DBpedia.org
- The system should be able to access and query Data.gov.uk

3.2.3 Plan A Requirements:

In addition to the Plan C requirements:

- The system should have an interface for pure natural language input, and be able to process well formed sentences into a SPARQL query, or forward the user to the interactive form if their input is garbled.

Implications

I decided to arrange my iterations around this idea of Plans A, B and C, Iterations 2-4 would focus on delivering Plan C requirements, and, if finished with enough time, I would start Iteration 5 in order to attempt Plan A or Plan B.

[19] W3C Schools Browser Statistics http://www.w3schools.com/browsers/browsers_stats.asp Accessed 4/5/2010

3.3 Considered Requirements

Several features were considered before a decision was reached on what the system should and should not do. Some considered requirements and reasoning behind their inclusion/exclusion are below:

Graph Based Query Building

RDF is a graph, and queries can be built up graphically using nodes and edges. This was considered but the availability of iSPARQL which fulfils this exact role, and the fact that it does require specialist knowledge to build queries graphically, meant that I felt it was best to approach the problem from a different angle.

Natural Language Processing (NLP)

This was considered and debated – natural language is familiar to all users and provides the most intuitive way of creating a query. However, natural language is also very complex and English, especially, can be very vague. So it was decided that, depending on time and availability of tools, NLP may be attempted as part of the project.

The Interactive Form

Initially the idea was for a purely natural language interface, but the difficulties of processing natural language inputs meant that building a restricted input was a wise back-up for users who weren't able to phrase a question properly. The form went through many design changes, initially starting as a series of fields which were in layout quite close to RDF, but eventually it was decided that it should have three levels of input for three types of user: Basic, Advanced and Expert. The Basic level of input was inspired by Semantic Search, which uses a sentence which has selectable inputs to build a query. This was appealing s the goal of the project was for some natural language input, so using a restricted natural language interface was a logical alternative.

Direct SPARQL Input

Considering the system was to be capable of running and obtaining results to SPARQL queries, it was not a huge leap to allow users to input their own SPARQL if so desired. Also considering that the system would need to check the validity of generated queries, it was natural that the system could be tailored to provide error checking for directly entered SPARQL. The direct input was to be the “Expert” level input of the interactive form.

3.4 Method

The system was developed over a 3 month period, which was divided up into a series of planned iterations:

Iteration 1 (19h February)**:**

- Deliverable: Project Plan (12th February)
- Activity: Research current systems for SPARQL query generation
- Activity: Research technologies for SPARQL generation, NLP, and dynamic page content
- Activity: Start Project Report
- Activity: set up IDE with tools required

Iteration 2 (5th March**):**

- Deliverable: SPARQL Query Handler back-end. This part of the program submits queries and return results.

Iteration 3 (19th March**):**

- Deliverable: Query Builder back-end. This part of the program generates SPARQL from given inputs.
- Activity: pilot features of the rest of the system.
- Deliverable: final requirements of the system.

Iteration 4(2nd April**):**

- Deliverable: Form GUI. This user interface is a front-end to the Query Builder. It constrains user input in such a way that it will almost certainly generate a valid query.
- **Completion of Plan C**

Iteration 5(23rd April**):**

- Deliverables:
  - **Plan A Features**: Simple Natural language Processing capabilities

    Or
  - **Plan B Features**: Focus on Data.gov.uk and attempt to improve it's accessibility.

Software Deadline (7th May**):**

Preferably the software should be ready by this date. This is a suggested deadline.

- Activity: Finish testing and tidying up the final build of the software

Final Deadline (14th May):

- Deliverables: Completed Software; Final Report.
- Activities: Possibly finishing the software; Finishing the the report.

3.5 Time Line

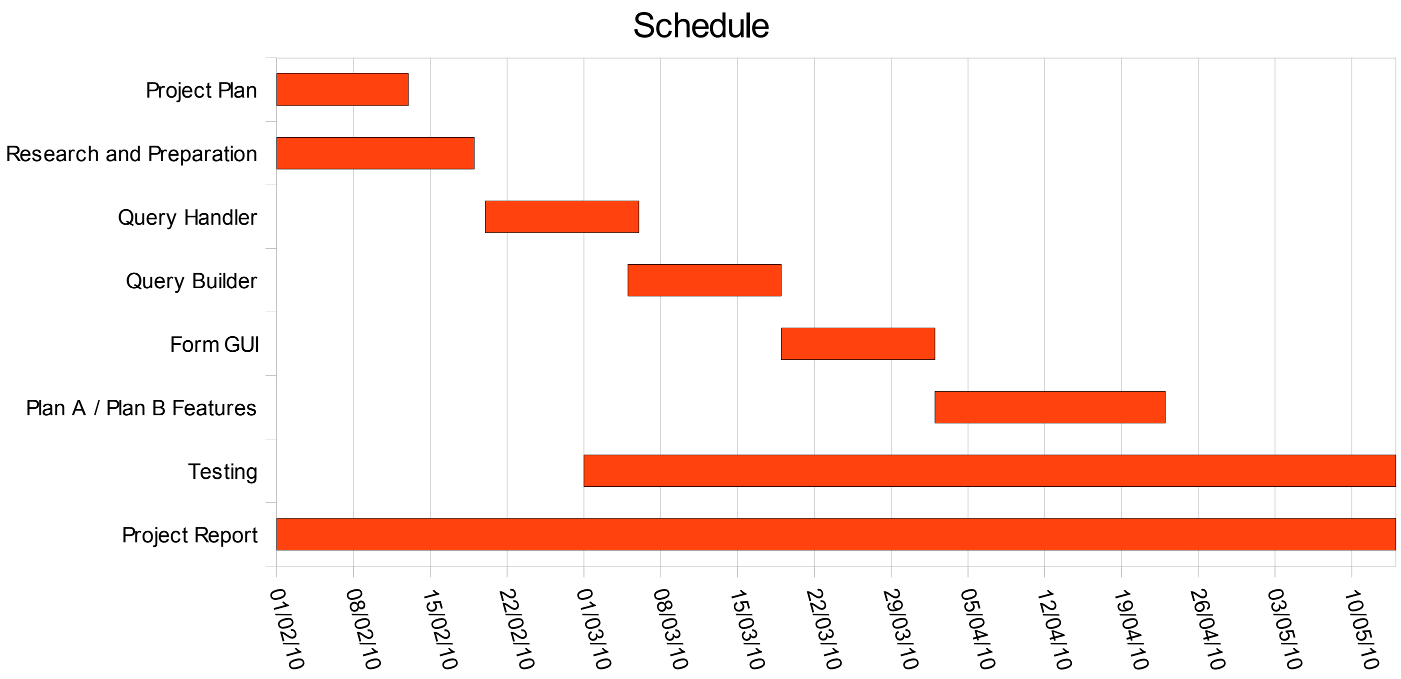


Fig. 3.5.a

The Gantt Chart above divides the project time into 7 day segments from the 1st of February to the 14th of May, the final deadline. The Iterations are referred to by the features associated with them, and run from the 1st of February to the 23rd of April, as described in the passage above (3.4).

3.6 Risk Analysis

Risk: Iteration 3 Slips

Iteration 3 entailed writing a QueryBuilder capable of building SPARQL queries from minimal inputs. SPARQL is however, a highly general language in which the same queries can be expressed in several different ways. I anticipated that this could turn out to be more difficult than initially thought.

Mitigation:

– 3 weeks of extra time were built into the original plan, some of this time could be used to finish this feature.

– It was possible to re-evaluate the functionality of this component and perhaps build it differently.

Risk: Plan A Unachievable

Plan A was always considered the loftiest and most difficult to achieve goal – to deliver both a working SPARQL generator with a simple user interface, and a natural language processor capable of translating English into a SPARQL-friendly format was always going to be a difficult task.

Mitigation

– Plan B was conceived as an alternative, slightly more realistic set of goals to achieve in the time allotted.

Risk: Time Problems

It is a general rule in software engineering that projects will take longer than planned, so it is always wise to overestimate the time required and to build in extra time to account for missed deadlines.

Mitigation

- After the projected final deadline was three weeks of “extra time before the software deadline. This time was to either add additional features, perfect existing ones or simply to finish the system if and when other features had slipped.

Risk: SPARQL Query Engines

As my project relied upon running SPARQL queries on external SPARQL engines, if these systems went down my project would be rendered incapable of executing generated SPARQL.

Mitigation

- I found several systems capable of running SPARQL queries and if one proved unreliable I could try another. One such system was SPARQLer on the department's own servers.
- If I encountered persistent trouble with using external systems it would be possible integrate 3rd party SPARQL query engine libraries into the project and run the queries locally.

3.7 Architecture

The design pattern I've used to create the system is a variation of the Client-Server architecture called the "Three-Tier Architecture", which splits the application into three layers[20]:

1. The Client Layer: contains the user interface and presentation information. Also has some basic input validation.
2. The Application Layer: Contains the bulk of the business logic and data access functionality.
3. The Data Layer: Contains the database.

A simplified architecture is presented below (Fig. 3.7.a), showing the types of files associated with each Tier and dividing them by both Tier and by physical location (I.e. Client-side or Server-side).

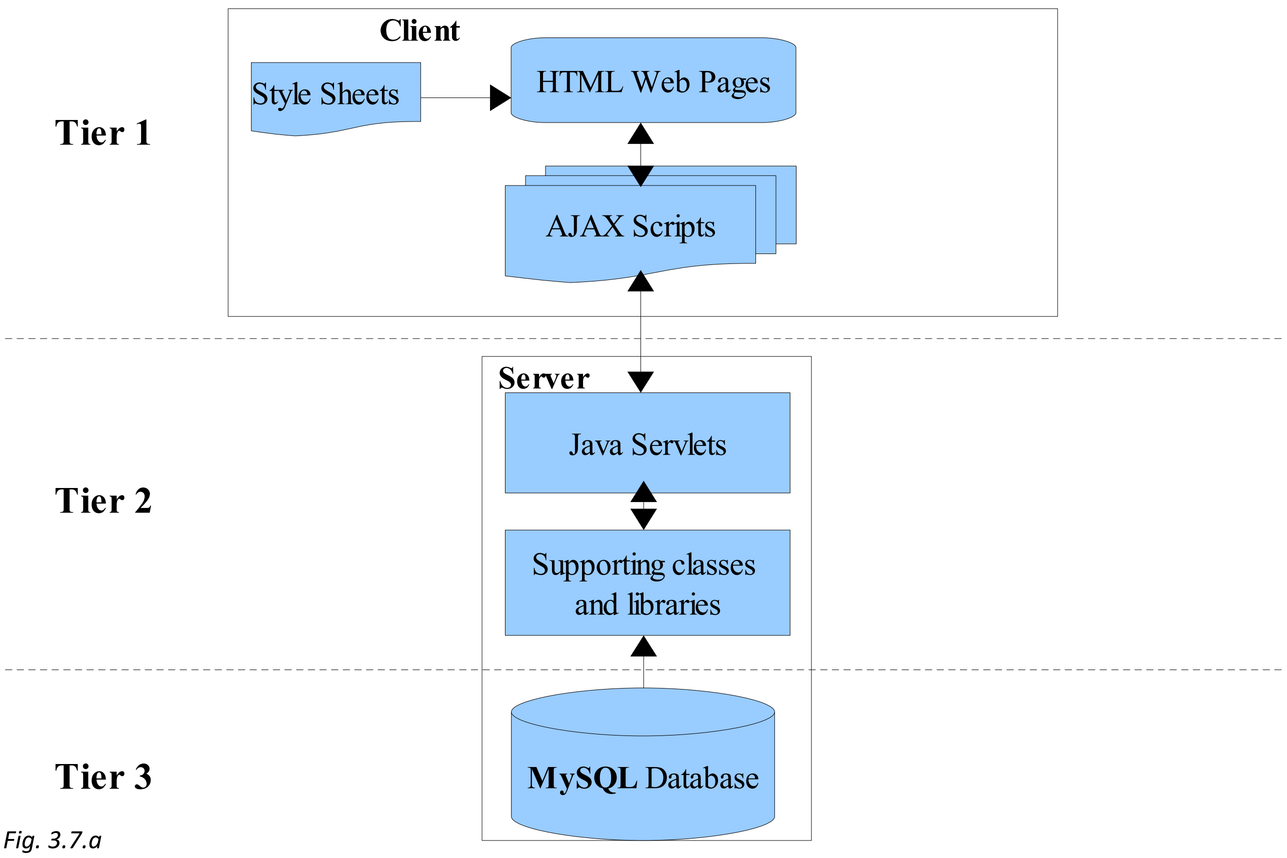


Fig. 3.7.a

[20] Linux Journal "Three-Tier Architecture" http://www.linuxjournal.com/article/3508 Accessed 12/4/2010

3.7.1 Client Layer Technologies

HTML

HyperText Markup Language (HTML) is the markup language most commonly used to structure the content of a web page. When building a web application, using some HTML is essentially unavoidable, but I chose it as the main method to structure the content of the user interface pages due to my familiarity with it.

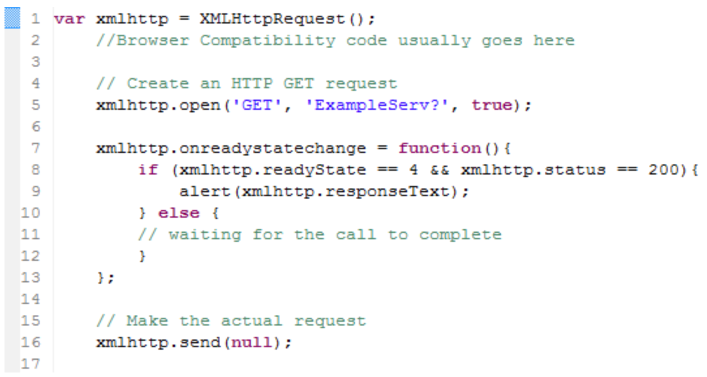
CSS

Cascading Style Sheets (CSS) are a way of encoding HTML style information in a separate file from the HTML itself, separating the style from the content and allowing a single style to be applied across an entire website. As well as being generally good practice to separate style and content, CSS makes it easy to keep the style of a website consistent and allows the author to easily change the style of the entire website without changing each individual page.

AJAX

Asynchronous JavaScript And XML (AJAX), is a set of web development techniques based on existing technologies, which allows web pages to access server side processing power without interfering with the display of the current page. The difference between AJAX style web pages and traditional JavaScript is that when a traditional JavaScript doGet or doPost HTTPRequest is issued, the page freezes while it waits for a response and then a new page is loaded on completion, by contrast AJAX requests are issued and run in the background and the web page can be interacted with by the user while the request takes place.

AJAX issues these asynchronous requests using a JavaScript class called XMLHttpRequest, which has a function assigned to it's ".onreadystatechange" property. This function is then called every time the request's state is changed, including the final state change (state 4) which represents the request has been completed and a response received.



```
var xmlhttp = XMLHttpRequest();
    //Browser Compatibility code usually goes here

    // Create an HTTP GET request
    xmlhttp.open('GET', 'ExampleServ?', true);

    xmlhttp.onreadystatechange = function(){
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200){
            alert(xmlhttp.responseText);
        } else {
        // waiting for the call to complete
        }
    };

    // Make the actual request
    xmlhttp.send(null);
```

The script snippet above shows the basic structure of how AJAX handles requests and responses. An XMLHttpRequest is created, it's ready state change function defined and finally the request is sent. In this script, once the response is received the responseText is extracted and displayed in an alert box, but as the name suggests, it's possible to return an XML document instead of plain text.

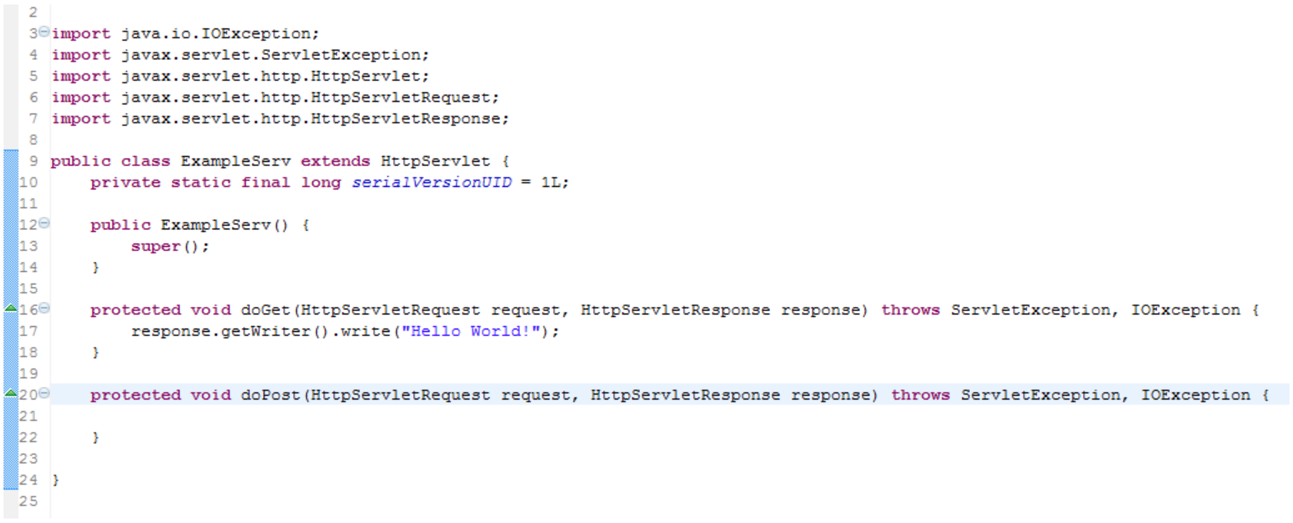
3.7.2 Server Layer Technologies

Java Servlets

Servlets are Java classes which conform to the Java Servlet API and extend the HttpServlet interface, they work by receiving requests and issuing responses. Requests handled by Servlets are instances of HTTPServletRequest, which usually represents an HTTP doGet or doPost request and contains associated parameters. DoGet and doPost are similar and the function of both is to pass parameters to a Servlet from an HTML page (usually a form) for server-side processing. Servlets are run in Web Containers and have their own URIs which allow HTTP Requests to be passed to specified Servlets. For example, the Servlet "ExampleServ" from the project "Project", being run on the localhost has the URI:

`http://localhost:8080/Project/ExampleServ?`

When this URI is invoked, a new instance of the Servlet ExampleServ is created by the Web Container and it's init() method run. The request is then passed to it's doGet or doPost method depending on how the request was issued. Now let's look at the code:



```java

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ExampleServ extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ExampleServ() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.getWriter().write("Hello World!");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }

}

```

Above is the code for a very basic "Hello World" Servlet: when it's doGet method is invoked by the URI above, it sends the response "Hello World!". If the URI was invoked by copy and pasting it into the address bar of a browser, the response would be displayed as an HTML page. This is not a particularly practical way to build web applications because we generally want the response to be displayed as part of a web page, and although we could write the entire web page to the response, it's far better to use a script to catch the response and write it to part of the web page. For this job, I used AJAX.

3.7.3 Data Layer Technologies

MySQL (Pronounced "My Sequel" or "My S.Q.L.")

A database was required to hold cached RDF Class and Property definitions, and although my initial choice was Derby, I switched to using MySQL after failing to get Derby to work with my web application. MySQL is the self styled "World's Most Popular Open Source Database"[21], and provides a relational database platform which is easily integrated into Web Applications. Essentially it didn't matter what database was used as long as it was fast and easy to query, MySQL proved to be the easiest system to set up and communicate with, so I chose to use it.

[21] MySQL homepage http://dev.mysql.com/tech-resources/articles/introduction-to-mysql-55.html Accessed 23/4/2010

3.7.4 Alternative Technologies

MVC Architecture

An alternative architecture often used for web applications is Model-View-Controller (MVC), which separates the state (model) from the presentation (view) and the logic (controller)[22]. However, since I was using AJAX to script the pages, a certain amount of program logic and state information would be present in the view, which defeats the purpose of MVC. Therefore the logical solution was to embrace the more traditional Client Server model, and allow for the presence of some logic in the Client Layer.

JSP/JavaBeans

In order to implement the system as an MVC architecture it would have been possible to use Java Server Pages (JSP) and JavaBeans instead of HTML and AJAX to talk to the Servlets. I did experiment with this, but I found that AJAX was preferable in terms of speed and ease for manipulating the page content asynchronously.

ASP.net

Microsoft's answer to JSP/Java Servlets is ASP.net (Active Server Pages) and C# (pronounced “See Sharp”), are part of the .net (“dot net”) framework. I experimented with the .net developer tools which are incredibly powerful and allow a proficient user to create powerful ASP web applications very quickly. However, I am unfamiliar with ASP and C# and picking up and using them was impossible in the allotted time.

Derby Database

Apache Derby is an open source Java database built to easily integrate with Java applications. Derby is very easy to set up as an integrated database in a small one user application, but I had trouble getting it operating in multi-user mode, and eventually had to switch to MySQL as it was taking too long to implement.

[22] Oracle Sun Developer Network http://java.sun.com/blueprints/patterns/MVC.html Accessed 12/4/2010

4 Implementation

4.1 Overview

The resultant system had several features for generating and running SPARQL queries from minimal user inputs as laid out in Plan C. However due to development problems and some poor design decisions on my part as well as unforeseen technical problems, it did not attain the Plan A features, and instead I had to enact Plan B.

4.2 Actual Iterations

Delays and complications meant that the iteration due dates had to be pushed back, this had been anticipated and was compensated for by the 3 weeks of extra time I had built into the project plan for this eventuality. Detailed below is how the project actually worked out, including missed deadlines and altered goals as well as the causes behind the deadline misses and the reasons behind the altered goals.

Iteration 1 (19h February)**:**

- Finished on time with no problems.
- Project Plan was delivered.
- Required research done.
- Report Started.
- A combination of Java Servlets and AJAX was decided upon as the platform.

Iteration 2 (5th March)**:**

- Finished on time though I had some trouble learning AJAX.
- Basic SPARQL handling was implemented, allowing direct SPARQL input, syntax checking and displaying of results.

Iteration 3 (19h March, Actual: 2nd April)**:**

- This iteration missed it's deadline by a week due to illness
    - Result: 1 week of slippage time was used to finish the iteration.
- A further week was lost due to problems with generating SPARQL:
    - It was determined that a Form GUI was needed first, so that the SPARQL generation could be tailored to the Form.
        - Generating SPARQL of arbitrary complexity was simply too large a task, the generator had to be scaled down to only fulfil the requirements of the form.
    - It was determined that a database was needed to cache the predicate information extracted from endpoints.
        - The Database proved a hassle; I initially chose Derby as the Database for the project as I was familiar with it. However getting it to work in a client-server environment turned out to be a challenge and I was forced to switch to MySQL at the 11th hour.

As a result of these issues, the goals of Iteration 3 and iteration 4 were changed.

New Goals:

1. Create a Database to cache predicate and domain information
2. Create a working prototype of the form input, including the Advanced and Expert Input
3. Have the capability to match predicates manually entered in the Advanced form to predicates cached in the database.
4. Have a working SPARQL generator that handles input from the Advanced form

Results

- All of the new goals were met and the iteration was finished by the 2nd of April.

Iteration 4(2nd April, Changed: 16th April):

As discussed above, Iteration 4 had to be modified as it's original goal (to deliver the form interface) had to be started in Iteration 3, in order to get SPARQL generation working.

New Goals:

- Achieve the minimum working spec for the "Plan C" requirements:
  - Deliver the "Basic" form interface, the half way house between Natural Language and the Advanced input.
  - Improve the features for the Form Interface, including:
    - Allow the user to see the SPARQL generated rather than simply run the query without asking. This is to allow users to learn SPARQL with the help of the tool.

Results

- The Plan C requirements were met, and the form interface was finished.

Iteration 5(23rd April, Changed: 14h May):

Due to delays with previous iterations, the deadline for Iteration 5 was pushed back to the final deadline of the course, using all remaining time. In theory this gave nearly a month to investigate and implement Plan B or Plan A features, but in practice this report needed to be compiled and finished and the system properly and rigorously tested and evaluated, leaving very little time to implement high level features.

In practice, several features were investigated (See "4.3 Beyond Plan C: Features Investigated ", next page), and with the remaining time I implemented an interface to perform queries against DBpedia. Finally I performed the last tests and bug fixes on the system before having it evaluated by a user group consisting of my peers and other people of varying levels of technical ability (See "5 Testing and Evaluation") .

4.3 Beyond Plan C: Features Investigated

Due to delays, there had not been enough time by the end of Iteration 4 to investigate fully whether Plan A or Plan B features should be undertaken, so it was decided to investigate several different options for high level features (that is, beyond basic Plan C features) to add to the system. The features investigated are outlined below:

4.3.1 Natural Language Processing

My initial goal had been to allow a Google like interface for querying Semantic data, but the way the semantic web works is that Ontologies are domain specific, so queries that work on DBpedia won't work on Data.gov.uk and vise versa, so the user will always have to specify what resource they wish to query and, if the system isn't familiar with it, what ontology needs to be loaded.

Still I did investigate using a NLP to build queries with the DBpedia or other ontologies loaded into the system and I found OpenEphyra[23], an open source Natural Language Processing (NLP) package. This initially looked promising to do the "part of speech" tagging necessary to extract the subject and predicate from a Natural Language question. Unfortunately the documentation for OpenEphyra is quite poor and even though I managed to extract the question analysis packages necessary, I didn't have enough time to trawl through the packages, class by class and try to figure out what bits to use. I did send an email to it's creators but didn't receive a response.

Conclusion:

I didn't have enough time to fully investigate OpenEphyra.

4.3.2 DBpedia

I had, on my initial research, seen that DBpedia was open source under the GNU license and freely linked it's SPARQL endpoint and had assumed that I could load it's ontologies to my database and query it like any other resource. Unfortunately their ontology, though open and well documented, is not provided in rdf format, and the actual linked data is behind a gateway query system which is the only way to gain access to their wealth of RDF data. Attempting to query their data from another SPARQL query processor returns empty XML documents.

So, in order to add DBpedia functionalities to the system, a custom DBpedia interface would have to be created, it's predicates added to the database and several systems changed so that only the DBpedia Query Form would have access to DBpedia specific Information.

Conclusion:

I added a new interface to interact with DBpedia and modified existing systems to incorporate it.

4.3.3 Data.gov.uk

One of the initial goals of Plan B was to integrate Data.gov.uk's ontology and create an interface for building queries. I did have enough time to create an interface for DBpedia, but creating a new interface for a large domain which uses a custom Ontology is time consuming and after adding DBpedia I didn't have enough time left to add Data.gov.uk and finish the report.

Conclusion:

Sadly I had to leave out Data.gov.uk, with another week I would be able to integrate it into the system.

[23] Ephyra: http://www.ephyra.info Accessed 1/5/2010

4.4 Libraries Used

In order to deliver some of the features, I had to use some external libraries. I full list of required .Jar files is available in the Maintenance Manual.

Jena Semantic Web Development Framework (http://openjena.org/)

Jena is a Java framework which provides facilities for handling RDF, RDFS, OWL and SPARQL in Java programs. Sparkle! uses the ARQ query processor part of Jena to provide Syntax checking on user entered SPARQL queries, but there is potential for greater integration. Jena is also capable of parsing ontologies and can be used to build an ontology from a target RDF file.

I had actually intended to integrate Jena more fully into the project, however because basic features such as the Database and SPARQL generation dragged on, I was never able to get around to it. Future iterations of Sparkle! could use Jena to build lists of predicates from any defines resource, extending the usefulness of the system.

MySQL Connector

The MySQL Connector is a driver that allows the JDBC (Java DataBase Connector) to create a connection to MySQL databases.

4.5 Final Feature List

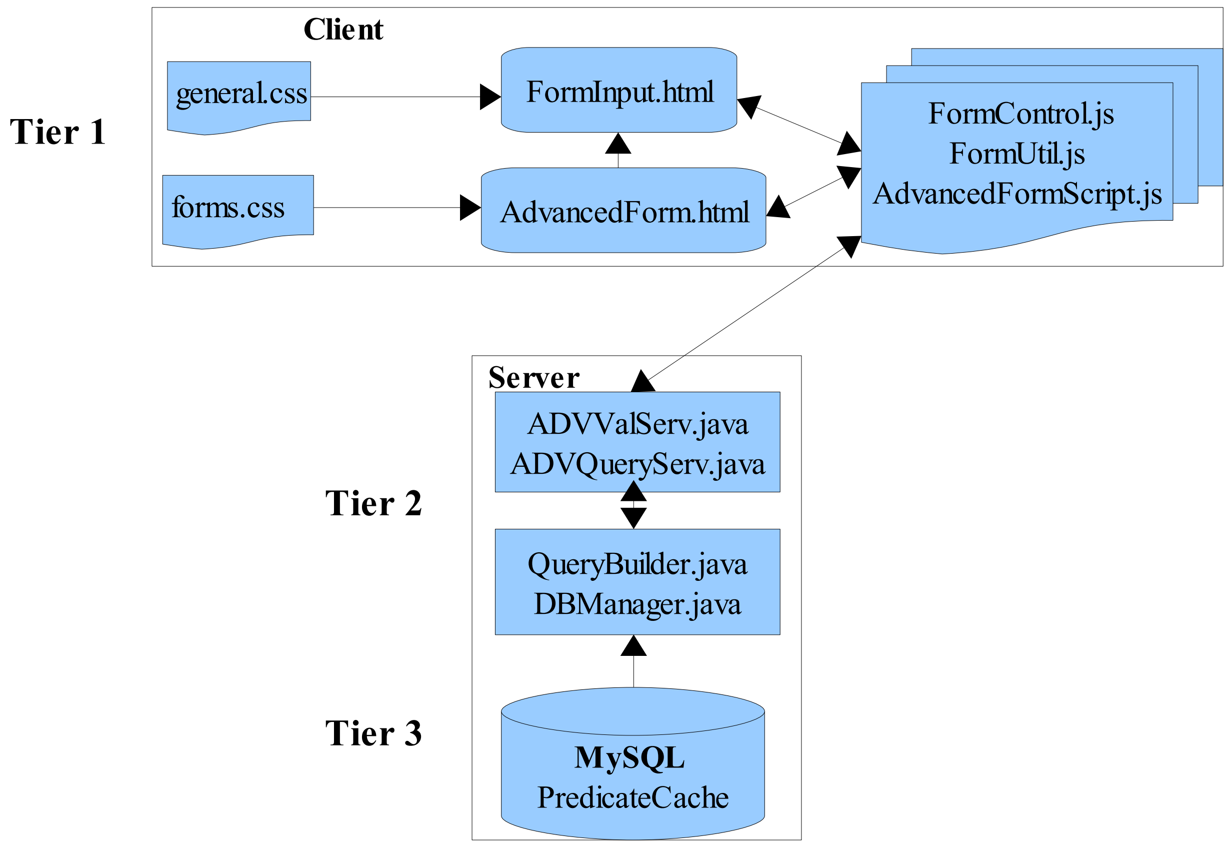
Feature	Description	Plan
SPARQL Generation	It would be easier for both novice and advanced users to query RDF if SPARQL could be automatically generated from limited inputs.	C
Query User Defined resources	It's important that the user can define their own SPARQL endpoints to query against.	C
Display Generated SPARQL	It'd be useful for people learning SPARQL to see what the system generates.	C
Edit Generated SPARQL	Allow the user to refine SPARQL queries generated by the system.	C
Direct SPARQL Entry	It would be useful to allow direct SPARQL input, but accessibility is the top priority as many systems off this feature already.	C
Syntax checking for Direct Input	If the system allows direct SPARQL input, it would be nice if the system could check the syntax of entered SPARQL and display error information	C
Facilities to Store RDF Metadata	If the system stores commonly used metadata definitions, users won't need to know the exact RDF predicate/class information	C
User Friendly Interface	In order to maximise data accessibility, the interface should be easy and intuitive to most users.	C
Browser Compatibility	Although Web sites are built with standard markup languages, different browsers render things differently, so a standard requirement for all web applications is that they run in the 3 most popular browsers, which as of April 2010 are Internet Explorer (IE), Firefox, and Google Chrome[24]	C
Speed	Queries should be executed as quickly as possible, as users expect fast results.	C
DBpedia.org	The system should be able to access and query DBpedia.org	B

[24] W3C Schools Browser Statistics http://www.w3schools.com/browsers/browsers_stats.asp Accessed 4/5/2010

4.6 How It All Fits Together

To explain how the system works, I shall describe how to use the Advanced Form which is part of the general Inputs form which allows the user to build general, non website specific queries and run them against a targeted resource or see the SPARQL. I'll use it to create a simple query against http://planetrdf.com/bloggers.rdf to extract the names of all bloggers listed there.

Advanced Form Architecture (Partial):



Above is the architecture of the Advanced Form interface. As described in the Architecture Section (3.7) , the system uses a 3 Tier architecture with the form inputs and accompanying scripts representing the client layer, the Java Servlets and utility classes in the server layer and finally the MySQL database as the data layer.

When the user first opens the FormInput.html page, no inputs are visible (Fig. 4.6.a), as the different levels of form (Basic, Advanced, Expert) are loaded on request. When the user clicks the Advanced tab, an AJAX script, loadHTML, from FormControl.js is triggered. It loads the AdvancedForm.html into the appropriate div.

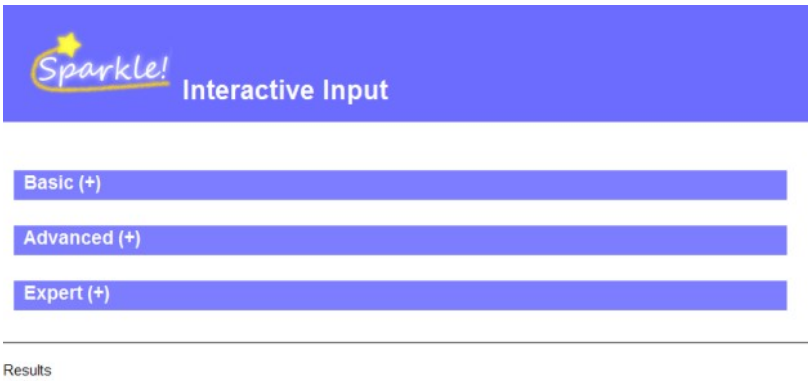
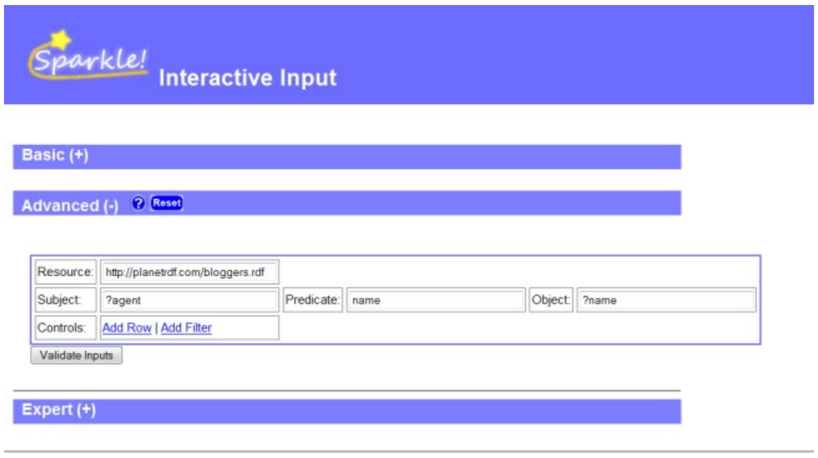


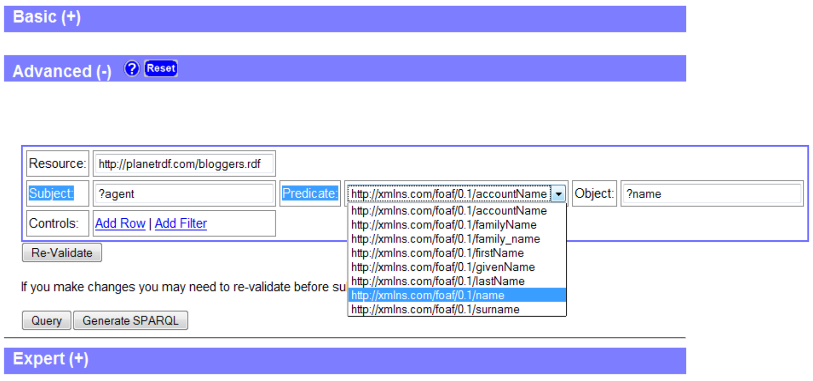
Fig. 4.6.a



Resource:	http://planetrdf.com/bloggers.rdf				
Subject:	?agent	Predicate:	name	Object:	?name

Fig. 4.6.b

Once the form has been loaded (Fig. 4.6.b), the user can then manipulate the form, adding triples and filters to the query. The next stage is to Validate the input to ensure it is formatted correctly so that it can be turned into a SPARQL query. This triggers an AJAX script which extracts the inputs from the form and creates an asynchronous XMLHTTPRequest which passed the form input to the Java Servlet ADVValServ via doGet. The validation servlet processes the form input and attempts to match any RDF predicates to predicates held in the database. As the inputs are checked and formatted, a new form is created by the servlet, and once the inputs have all been checked, the new form is returned to the AJAX script which replaced the old form with the new one. The new form lets the user refine their query. If the user makes large changes, the form requires the user to re-validate, if they only make certain small changes, it isn't necessary.



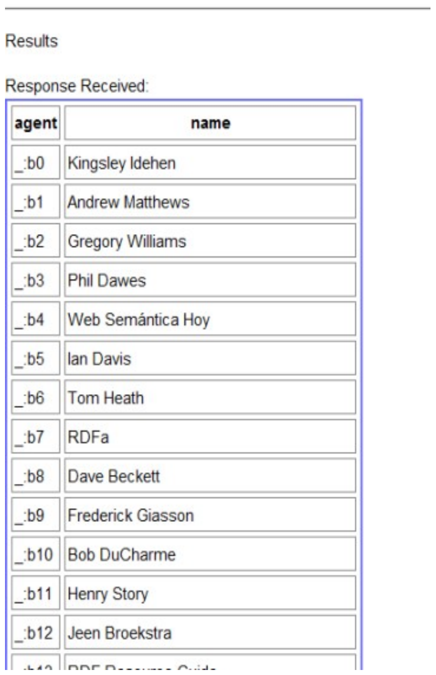
Resource:	http://planetrdf.com/bloggers.rdf				
Subject:	?agent	Predicate:	http://xmlns.com/foaf/0.1/accountName	Object:	?name
Controls:	Add Row | Add Filter				

http://xmlns.com/foaf/0.1/accountName
http://xmlns.com/foaf/0.1/familyName
http://xmlns.com/foaf/0.1/family_name
http://xmlns.com/foaf/0.1/firstName
http://xmlns.com/foaf/0.1/givenName
http://xmlns.com/foaf/0.1/lastName
http://xmlns.com/foaf/0.1/name
http://xmlns.com/foaf/0.1/surname

If you make changes you may need to re-validate before su

Fig. 4.6.c

Once the user has refined their query, they can opt to "Query" or to "Generate SPARQL". Both buttons use the same AJAX script and same Servlet, but they send slightly different commands. If the user selects "Query" the ADV_Query script extracts the validated form inputs and forwards them to ADVQueryServ via a doGet XMLHTTPRequest, the Query Servlet then uses QueryBuilder.java to create a SPARQL query and sends it to sparql.org for processing, the results are then returned to the script as an XML document. An XSLT transform is then run on the results and they are displayed as a table:

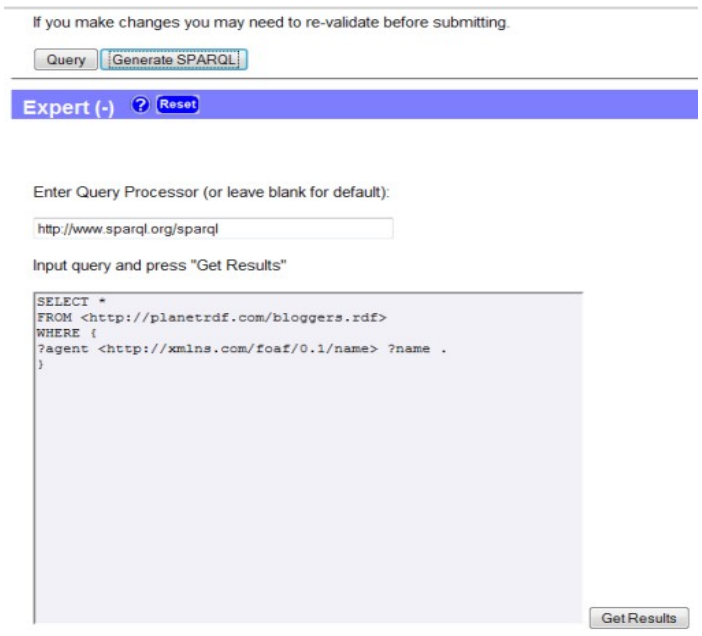


Results

Response Received:

agent	name
_:b0	Kingsley Idehen
_:b1	Andrew Matthews
_:b2	Gregory Williams
_:b3	Phil Dawes
_:b4	Web Semántica Hoy
_:b5	Ian Davis
_:b6	Tom Heath
_:b7	RDFa
_:b8	Dave Beckett
_:b9	Frederick Giasson
_:b10	Bob DuCharme
_:b11	Henry Story
_:b12	Jeen Broekstra

If however, the user clicks "Generate SPARQL", the same script is triggered but with a different control variable. This causes the script to use loadHTML to load the Expert Form tab and inserts "`&getsparql=T`" parameter into the request. With this parameter, the servlet still generates the SPARQL via QueryBuilder, but does not send it for processing, instead it returns the SPARQL as text, and the script loads the SPARQL into the Expert form (Fig 4.6.d).


If you make changes you may need to re-validate before submitting.
Query
Generate SPARQL
Expert (-)
?
Reset
Enter Query Processor (or leave blank for default):
Input query and press "Get Results"
Get Results

http://www.sparql.org/sparql

```
SELECT *
FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
?agent <http://xmlns.com/foaf/0.1/name> ?name .
}
```

Fig. 4.6.d

Summary

All system functionality uses a similar sequence:

1. The user loads an HTML page with form inputs;
2. The user manipulates the forms to input desired parameters, then pushes a button;
3. The button triggers an AJAX script which creates a XMLHTTPRequest to the server;
4. A Java Servlet handles the request, processes the inputs and returns some data;
5. The script updates the webpage, and the user either refines their inputs or has received their desired results.

There are variations involving multiple scripts and multiple Servlets, but understanding this sequence will allow you to understand how the system functions.

5 Testing and Evaluation

System testing focused around testing first the deliverance of features, second usability, and third durability. The first round of testing involved myself testing the system against the functional requirements set out in the Development Plans (Design Section) . The second and third rounds involved other computing students attempting to use the system to perform a set series of tasks, and then being given free reign on the system to attempt to cause the system to stop or freeze in any way they could.

5.1 First Round – Functionality Testing

5.1.1 SPARQL Generation & Querying

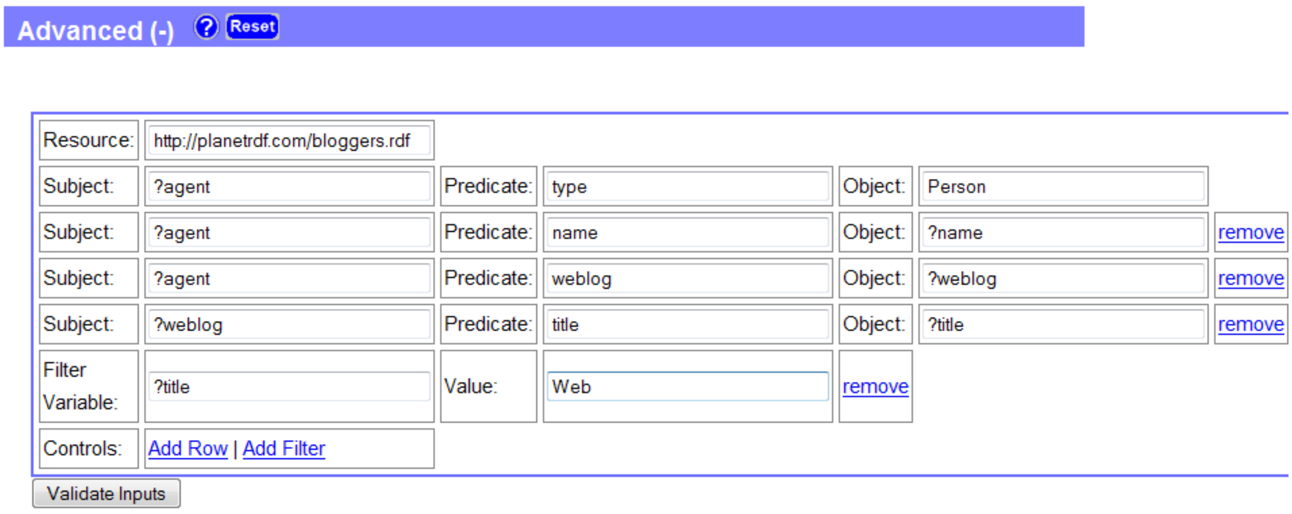
Test:

Three major features of the system are the generation of SPARQL queries, the displaying of the generated SPARQL and the ability to query user defined resources. It was decided that to test these features, it was best to try to use the system to create and display a non trivial query; in this case equivalent to the natural language task "list the names and weblog titles of all people on the bloggers list at planetrdf.com whose weblogs contain the word 'Web'". A correct SPARQL equivalent for such a query is shown below:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT *
FROM <http://planetrdf.com/bloggers.rdf>
WHERE{
?agent rdf:type foaf:Person .
?agent foaf:name ?name .
?agent foaf:weblog ?weblog .
?weblog dc:title ?title .
FILTER regex(?title, "Web") .
}
```

Results

I used the Advanced part of the input form, and entered the information as shown below:



Resource:	http://planetrdf.com/bloggers.rdf					
Subject:	?agent	Predicate:	type	Object:	Person	
Subject:	?agent	Predicate:	name	Object:	?name	remove
Subject:	?agent	Predicate:	weblog	Object:	?weblog	remove
Subject:	?weblog	Predicate:	title	Object:	?title	remove
Filter Variable:	?title	Value:	Web	remove		
Controls:	Add Row | Add Filter					

The form validated, and, after selecting the appropriate predicates from the drop down boxes, I selected

"Generate SPARQL". The Expert tab was automatically expanded and the resultant SPARQL is shown below:

```
SELECT *
FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
?agent <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
?agent <http://xmlns.com/foaf/0.1/name> ?name .
?agent <http://xmlns.com/foaf/0.1/weblog> ?weblog .
?weblog <http://purl.org/dc/elements/1.1/title> ?title .
FILTER regex(?title, "Web") .
}
```

The generated SPARQL looks different, but only because instead of using PREFIX statements to shorten resource identifiers, it uses the full URI, the results are the same.

Conclusion:

The system successfully generated an equivalent query and displayed the SPARQL generated.

5.1.2 Direct SPARQL Entry

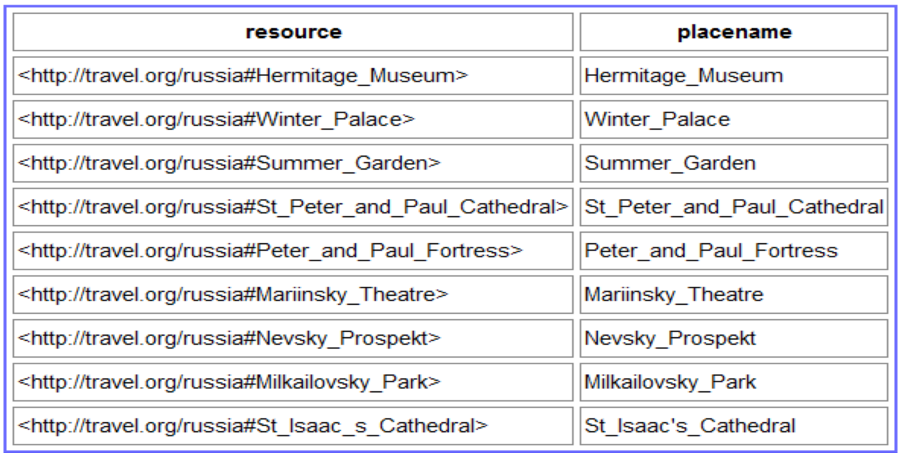
One of the expert level features is the ability to enter your own SPARQL query and have the system run it. There are two interfaces for this, the "Direct Input" screen and the "Expert" tab in the form input screen, so I tested both of them with the same query. The query I used uses a custom definition that isn't in the system database, and queries against RDF data for travel.org. The query (below) picks out tourist sites in St Petersburg.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX NS0: <http://travel.org/russia#>

SELECT ?resource ?placename
FROM
<http://www.atl.external.lmco.com/projects/ontology/ontologies/russia/
russiaA.rdf>
WHERE {
         ?resource rdfs:label ?placename .
         ?resource NS0:lie_in NS0:St.Petersburg .

}
```

Results

Both the Direct Input and Expert tab retrieved the same set of results (Fig.5.1.3.a below) which I then checked against the results when run through the "Vituoso OpenLink SPARQL Query" page[25]. The results were consistent.

[25] http://dbpedia.org/sparql Accessed 6/5/2010



resource	placename
<http://travel.org/russia#Hermitage_Museum>	Hermitage_Museum
<http://travel.org/russia#Winter_Palace>	Winter_Palace
<http://travel.org/russia#Summer_Garden>	Summer_Garden
<http://travel.org/russia#St_Peter_and_Paul_Cathedral>	St_Peter_and_Paul_Cathedral
<http://travel.org/russia#Peter_and_Paul_Fortress>	Peter_and_Paul_Fortress
<http://travel.org/russia#Mariinsky_Theatre>	Mariinsky_Theatre
<http://travel.org/russia#Nevsky_Prospekt>	Nevsky_Prospekt
<http://travel.org/russia#Milkailovsky_Park>	Milkailovsky_Park
<http://travel.org/russia#St_Isaac_s_Cathedral>	St_Isaac's_Cathedral

Fig 5.1.3.a

Conclusion

Both direct SPARQL entry methods yield accurate results, this feature works.

5.1.3 Syntax checking for Direct Input

The Syntax checking feature is designed to catch syntactic errors on SPARQL entered by the user using the JENA framework. I decided that to test this, I would input a complex query with several different errors in it. The query I used was the same query I used to test SPARQL generation, with the errors highlighted in red, as shown below:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT *
FRO <http://planetrdf.com/bloggers.rdf>
WHERE(
?agent rdf:type foaf:Person .
?agent foaf:name ?name _
?agent foaf:weblog ?weblog .
?weblog dctitle ?title .
FILTER regex(?title, "Web") .
)
```

Errors

1) foaf prefix link not properly closed
2) FROM clause misspelled
3) '(' instead of '{' on WHERE clause
4) missing '.' at the end of the 2nd constraint
5) missing ':' in 'dc:title'
6) WHERE clause closed with ')' rather than '}'

Results

The results were the same for both SPARQL interfaces, they both flagged every error with the corresponding error messages shown below:

1. Error: “Encountered " "<" "< "" at line 3, column 14. Was expecting: ... “
    - It didn't point out that a > was missing but did at least draw attention tot he right part of the code.
2. Error: “Lexical error at line 7, column 4. Encountered: " " (32), after : "FRO" “
3. Error: “Encountered " "(" "("" at line 8, column 6. Was expecting: "{" ... “
4. Error: “Encountered " "?agent "" at line 11, column 1. Was expecting one of: "graph" ... "optional" ... "filter" ... "{" ... "}" ... ";" ... "," ... "." ... “ \
5. Error: “Lexical error at line 12, column 16. Encountered: " " (32), after : "dctitle" “
    - This is perhaps the most cryptic one, it did draw attention to the right phrase but didn't pinpoint exactly what was wrong with it.
6. Error: “Syntax Error: Encountered " ")" ") "" at line 14, column 1. Was expecting one of: ... ... "graph" ... "optional" ... "filter" ... "true" ... "false" ... ... "(" ... ... "{" ... "}" ... "[" ... ... “

Conclusion:

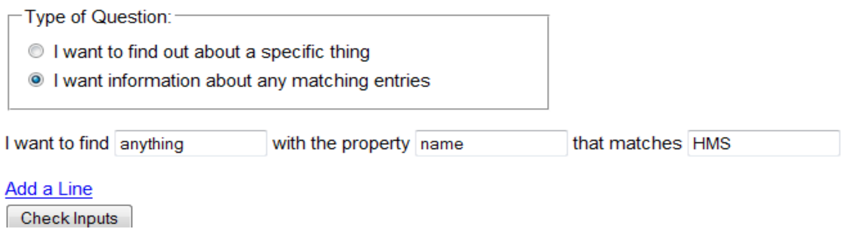
Some of the errors were a bit cryptic or not quite precise, but overall I feel they were satisfactory.

5.1.4 Ability to Query DBpedia

In order to test the DBpedia querying capability, it was decided to run three separate queries against it. The first query would be to find the names of all British Naval ships listed on DBpedia (by looking for “HMS” in the name), the second would be to find out the population of Aberdeen.

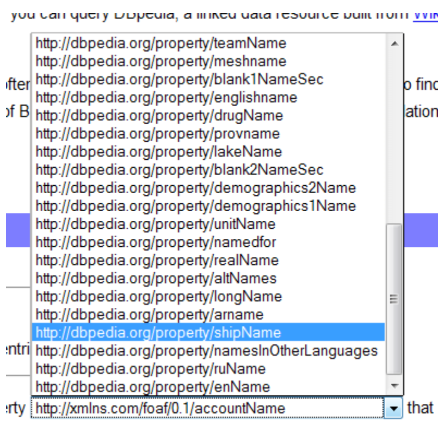
1st Query

For the first query I selected the general query interface on the DBpedia form, choosing to find “anything with the property name that matches HMS” as shown below:


Type of Question:
○ I want to find out about a specific thing
● I want information about any matching entries

I want to find [anything] with the property [name] that matches [HMS]

After validation I was presented with a large array of possible matches for the property “name”, which was unsurprising as it's a very general term. The best match was the DBpedia property “shipName”, which I selected, as shown below.

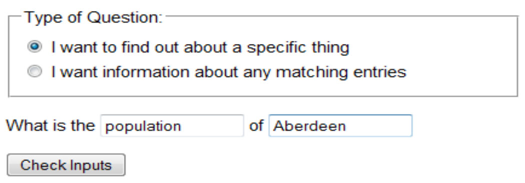


```
http://dbpedia.org/property/teamName
http://dbpedia.org/property/meshname
http://dbpedia.org/property/blank1NameSec
http://dbpedia.org/property/englishname
http://dbpedia.org/property/drugName
http://dbpedia.org/property/provname
http://dbpedia.org/property/lakeName
http://dbpedia.org/property/blank2NameSec
http://dbpedia.org/property/demographics2Name
http://dbpedia.org/property/demographics1Name
http://dbpedia.org/property/unitName
http://dbpedia.org/property/namedfor
http://dbpedia.org/property/realName
http://dbpedia.org/property/altNames
http://dbpedia.org/property/longName
http://dbpedia.org/property/arname
http://dbpedia.org/property/shipName
http://dbpedia.org/property/namesInOtherLanguages
http://dbpedia.org/property/ruName
http://dbpedia.org/property/enName
http://xmlns.com/foaf/0.1/accountName
```

With the property selected, I hit "submit" and after a moment or two received a huge list of ships which, as expected, all contained the letters "HMS" in their names.

2nd Query

For the second query I chose the specific query interface, and asked for the "population of Aberdeen" as shown below:

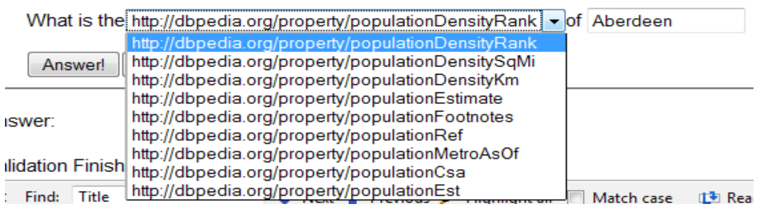


Type of Question:
- [✓ I want to find out about a specific thing]
- I want information about any matching entries

What is the population of Aberdeen

Check Inputs

When validated I was shown several different results for population (below), and it was not immediately clear which would give the best result.

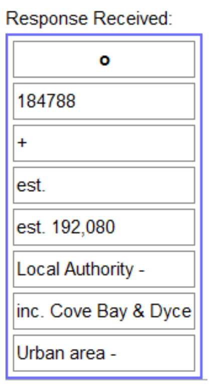


What is the http://dbpedia.org/property/populationDensityRank of Aberdeen

Answer!

```
http://dbpedia.org/property/populationDensityRank
http://dbpedia.org/property/populationDensitySqMi
http://dbpedia.org/property/populationDensityKm
http://dbpedia.org/property/populationEstimate
http://dbpedia.org/property/populationFootnotes
http://dbpedia.org/property/populationRef
http://dbpedia.org/property/populationMetroAsOf
http://dbpedia.org/property/populationCsa
http://dbpedia.org/property/populationEst
```

I took a guess and tried "populationEst", but this didn't return any results. I tried a few others before trying "populationRef" and getting some fairly ambiguous results:



Response Received:

o
184788
+
est.
est. 192,080
Local Authority -
inc. Cove Bay & Dyce
Urban area -

This isn't really the fault of my system, but rather a problem with DBpedia's dataset and ontology being so inconsistent, and is unfortunately unavoidable.

Conclusion:

The DBpedia form is functional, but the ambiguity and inconsistency of the DBpedia ontology and data makes it sometimes frustrating to use. A possible improvement on the form would be to automatically run the query against any inputs that are similar to the user input, and display all results and what properties they are under, this would make it easy to see all information on the population of a city, but in the case of looking for all ships with a certain name, could be very confusing as "name" is far more ambiguous than "population".

It's difficult to see how best to combat this problem from my end, it would be far preferable for DBpedia to standardise their ontology and remove the conflicting and repeated properties.

5.1.5 Browser Compatibility

A major problem with all websites is that web browsers render pages differently, and AJAX in particular has problems with browser compatibility. For this reason I decided to test how the application is rendered and AJAX scripting functionality across the 3 most popular browsers according to the W3C[26], Internet Explorer (version 8), Firefox (3.6.3) and Google Chrome (4.1.249.1064).

The tests are as follows:

Number	Description	Feature Tested
1.	Load FormInput.html and visually inspect how the page is rendered.	Tests that the page loads properly, with no artifacts that obscure the content.
2.	Expand the "Advanced" tab.	Tests basic AJAX functionality to fetch another page and load it.
3.	Add a row to the form.	Tests AJAX copy and insert functions.
4.	Remove a row from the form.	Tests AJAX select and table row delete.
5.	Enter a basic query and validate	Tests AJAX server communication.

Results:

Number	Internet Explorer	Firefox	Google Chrome
1.	Pass	Pass	Pass
2.	Pass	Pass	Pass
3.	Pass	Pass	Pass
4.	Pass	Pass	Pass
5.	Pass	Pass	Pass

Conclusion

All browsers passed all tests, meaning the system is works on all three major browsers.

[26] W3C Schools Browser Statistics http://www.w3schools.com/browsers/browsers_stats.asp Accessed 4/5/2010

5.2 Usability Testing

In this round of testing, I posed a short series of tasks to a group of technically minded people, some of whom had experience with SPARQL but others who had never used it, and observed how quickly they managed to finish the task, and noted what things they tried in order to achieve their goals. The sheet I used to keep track of the test candidates is contained in Appendix A.

Tasks:

1. Use DBpedia to find out the population of Berlin.
2. Find the names of all British Naval ships (names contain "HMS") listed in DBpedia.
3. Find the names of all bloggers listed on http://planetrdf.com/bloggers.rdf .
4. Display the SPARQL for the above query.
5. Display a help file.
6. Return to the Index page.

Results

Below is a selection of results, structured as a series of tables breaking down the tests by Task and then by Candidate number. Bugs are highlighted in bold.

Task 1		
C	**Time**	**Notes**
1	10min	Hit enter rather than query. Was confused by number of possible selections for "population". Tried wrong interface. Gave up.
2	6min	Confused by number of selections. Tried several of them before finding an answer.
3	9min	Also confused by number of matches for "population". Tried a couple of them before giving up.
4	6min	Read the help file, then formed a correct query.
5	5min	Used "pop" rather than population, spent some time checking different options before finding correct one.

Task 2		
C	**Time**	**Notes**
1	7min	Used wrong interface. Tried synonyms like “find British battleships”. Tried to use HMS as a property. I had to explain Subject-Predicate-Object triples and how to enter them correctly. Didn't look for help files.
2	7min	Used wrong interface. I had to explain how to break the question into subject-predicate-object. Didn't check the help files.
3	4min	Had a better grasp of the question and selected the right interface quickly, broke up the question quickly and found the answer with relative ease.
4	5min	Tried wrong interface, but realised it wasn't possible. Used correct interface fairly quickly.
5	6min	Changed interfaces to check the other. Used it and got the answer fairly quickly. Used “name” instead of “shipName” however and got irrelevant answers at first.

Task 3		
C	**Time**	**Notes**
1	3 min	Used Back button rather than logo. Wasn't sure which page to go to. Once located, went straight to Basic interface and worked out what to do in a matter of seconds. Had to be prompted to scroll down for results.
2	3min	Used the logo to navigate to Index. Found the general input quickly, used Advanced form.
3	4min	Used back button to navigate to Index. Found the general form quickly. Used Basic input to find the answer.
4	5min	Used back button. Tried Direct Input, realised it was wrong, found General input form. Used basic tab.
5	3min	Used back button to navigate. Found the general form quickly, used basic tab.

Task 4		
C	**Time**	**Notes**
1	5sec	Worked out to hit “SPARQL” button very quickly.
2	3sec	
3	30sec	Almost clicked “sparkle!” logo, needed to be prompted.
4	5sec	looked around for a moment before finding it.
5	2sec	

Task 5		
C	**Time**	**Notes**
1	2sec	Once prompted to find help, hit the “?” button almost instantly. Expected clicking the logo again to hide help, didn't see the “hide” link.
2	2sec	Also found the help section very easily. Found the “hide” link.
3	10sec	Hadn't noticed Help button.
4	1sec	Had already used help before.
5	2sec	Found Help instantly.

Task 6		
C	**Time**	**Notes**
1	5sec	Spent some time scrolling the page, then clicked Sparkle! logo.
2	1sec	Had used logo previously, new what to do.
3	1sec	Used Back button.
4	4sec	took a moment, then guessed and hit the Sparkle! logo.
5	5sec	Scrolled the page, then guessed to click the logo.

Changes

- Candidate 1 had trouble locating the form prototype, so I relabelled the link “General Query Form”.
- I added text to the interfaces, prompting users to click the “?” logos for help if they get stuck.

Conclusion

Several issues were highlighted by the usability tests:

1. Users had trouble understanding how to break natural language into Subject-Predicate-Object components, and would ask me for help. If I refused, they would give up rather than check the help files. This may be because they knew I knew how it works, but also may point to a fundamental flaw – people don't like to learn to use things.
2. Users generally identified the help logo correctly and instantly, but they rarely click it. Instead they try several things and then give up if the system doesn't act as they expect it to.
3. Users found the number of choices presented to them confusing.
4. Users didn't read the dialogue boxes.
5. Users didn't usually read any of the text on the page – they focused in on what they thought would get them closer to their goals.

Overall I was fairly happy with how quickly the users picked up how to use the system, but disappointed that they didn't use the help files or read system messages.

5.3 Durability Testing

To test the durability of the system, each candidate was asked to attempt to break the web interface in any way they could (short of deleting the code or powering off my laptop). I watched them as they attempted to do this, and filled out a form detailing what they tried and what happened (Appendix B). The results are shown below with bugs highlighted in bold:

C	Action	Result
1	Entered Random input into Expert tab.	System flagged syntax errors.
	Put a non rdf resource into advanced tab and then entered a nonsense query.	System flagged errors on predicates.
	Put “anything” into property box of basic	System asked user to refine the search.
	Bad address with a valid query in basic.	System flagged an XML parsing error.
	Clicked reset button on DBpedia page.	**Reset script didn't work.**
	Tried to enter nonsense queries to DBpedia.	System found matching predicates, formed query but didn't receive any results.
2	Also went straight for the Expert tab and entered nonsense.	System flagged Syntax Errors
	Created lots of inputs in Advanced Tab. Got bored at about 20 rows.	System added rows, no errors.
	Tried a bad address with a valid query in basic.	System flagged an XML parsing error.
	Created valid query in Advanced, repeatedly hit “Validate” then hit “generate SPARQL”.	System correctly revalidated input and generated valid SPARQL.
3	Tried entering nonsense on DBpedia forms.	System rejected inputs it couldn't match, but did match some of them.
	Tried using symbols like “@” “#” etc. on Advanced Form.	System rejected symbol inputs.
	Attempted basic SQL injection attack on Advanced Form.	System rejected input.
4	Opened and closed all the forms and help files.	System properly opened and closed tabs.
	Tried copy and pasting a large amount of text into Advanced tab.	System accepted it as it contained a URL. Failed to get any results.
	Entered nonsense into direct input form.	System flagged syntax errors.
5	Tried opening and closing all the interfaces and help tabs.	System opened and closed tabs correctly.
	Pushed all the reset buttons.	System reloaded form tabs correctly.
	Entered nonsense into expert tab	System flagged syntax errors.
	Tried a bad address with a valid query in basic.	XML parsing error.

Bug Fixes:

- Candidate 1 found that the DBpedia reset button did not work, so I fixed the script.

Conclusion:

Apart from one Scripting Error which I subsequently fixed, the system appears to be highly durable.

5.4 Comparisons With Existing Systems

- **SWS: Semantic Web Search (http://www.semanticwebsearch.com/query/)**
- **DBPQ: DBpedia Query Builder (http://querybuilder.dbpedia.org/)**
- **iSPARQL: OpenLink iSPARQL (http://demo.openlinksw.com/isparql/)**

Feature	Sparkle!	SWS	DBPQ	iSPARQL
SPARQL Generation	Yes, in 3 ways.	No.	Yes, but only basic.	Yes, graph based.
Display Generated SPARQL	Yes.	No.	Has button, Doesn't work.	Yes.
Edit Generated SPARQL.	Yes	No.	No.	Yes.
Query user defined resources	Yes.	Yes.	No.	Yes.
Facilities to Store RDF Metadata	Yes.	Yes.	No.	Yes.
Direct SPARQL Entry	Yes.	Yes.	No.	Yes.
Syntax checking for Direct Input	Yes.	Yes.	No.	Yes.
Stability	Stable.	Stable.	Stable.	Crashes often. Buggy.
Complexity	Depends on interface.	Complex.	Simple.	Very complicated.
Browser Compatibility	IE, Firefox, Chrome.	IE, Firefox, Chrome	IE, Firefox, Chrome	Firefox, Chrome. Not IE.

Conclusion

From my testing, I think Sparkle! compares favorably with similar systems, especially in terms of functionality and usability, but it is also farm more stable than the very complex though powerful iSPARQL.

5.5 Evaluation

Functional

Compared against the functional requirements (S 3.2.1), the system has met the Plan C requirements and attempted plan B functionalities by allowing the user to query DBpedia. However it has not met the Plan A requirements as discussed in section 4.3.

Usability

One thing that was brought up repeatedly during the usability testing was the number of options that simply weren't helpful because those properties weren't present on the target resource. As a result I think it would be beneficial in future versions to generate the list from what properties/predicates are actually present, this can be accomplished by parsing the target RDF using Jena.

Another result of the Usability testing was that it is really very hard to make the Semantic web accessible to users who aren't familiar with the concepts involved especially with regards to organising information into triples. One of the hardest problems all test candidates had was taking the natural language tasks I posed to them and identifying the components necessary to form queries. Once users were familiar with the process they became more efficient, but they were prone to giving up at the first hurdle, which could be a sign that the system is too hard to use or the concepts too demanding to learn quickly.

Durability

I was pleased that none of the testers managed to severely brake the system. Only one bug was discovered and subsequently fixed, so I believe that the system is highly durable.

Conclusion

From the testing regime I imposed, I think it's clear that the system met it's functional requirements and is also fairly durable. I was slightly unhappy with the usability testing and I feel there would be room for future interface improvements, especially with regards to how options are presented to the user.

Overall I feel that the usability testing made it clear that the way forward in querying the Semantic Web is Semantic Question answering of Natural Language questions rather than helping users to form SPARQL queries through an interactive interface.

6 Conclusion

6.1 In Retrospect...

Looking back, some things are clear to me now that should have be clearer at the start of this project, especially with regards to usability and making the Semantic Web more accessible: Generally users are not willing to learn how Linked Data works just so they can perform clever queries on Semantic Resources; they are more interested in using tools they are already familiar with, like Search Engines, to find the information they want. If I were to start the project again tomorrow, with the same original goal to make the Semantic Web accessible to the end-user, I would have focused exclusively on Natural Language Processing and Part-of-Speech tagging to translate English language questions into SPARQL queries. The mistake I made was getting sidetracked in allowing advanced and expert users to generate SPARQL for their own purposes, and I spent the majority of my development time tackling that set of challenges with little appreciation for the difficulty of the ultimate goal I had originally set myself.

I am also unhappy with my decision to use cached standard definitions as a basis for creating queries. I had at first assumed that, like most web standards, they were at least fairly widely implemented; this turned out to be a highly erroneous assumption, and my forays into finding RDF data have shown that these "standards" are by and large completely ignored in favor of custom Ontologies. If I were to restart the project tomorrow, I'd use Jena's document extraction and abstraction abilities to build a local copy of the target Ontology being used so that any resource could be properly queried. This would also solve the problem that currently, there are often many conflicting properties/predicates given as possible matches to user input, and selecting the right one is essentially educated trial and error.

Hindsight is, of course, always perfect and these things were not at all clear to me at the start.

6.2 Achievements

I am not entirely disappointed with my work - I feel that it helps technically minded users to generate SPARQL queries and access resources like DBpedia which are quite hard to use. I'm also pleased with it's durability and presentation and that even for a SPARQL generation tool, my test candidates (most of whom did not have experience with SPARQL or linked data) managed to use it to form queries.

I am also pleased with the quality of the presentation, and the visual and functional features I managed to produce with AJAX scripting which I was previously entirely unfamiliar with.

The system did meet most of it's original functional requirements, and although there is room for improvement this is unsurprising given the magnitude of the task and the time alloted. Still, with the knowledge I have today, I think I would rebuild the system with a different focus and slightly different goals in mind.

6.3 Overall

I feel that the system does represent an attempt to help move the Semantic Web forward, and although not a perfect attempt, I am happy to have tried to contribute something to a wider movement. On a personal note, I feel that my skills as a programmer have been tested and refine, and I have gained technical knowledge and understanding of Semantic Web topics which I did not posses previously.

7 Outlook

7.1 Possible System Improvements

My application does not represent a perfect system, and although it does what it is supposed to, there are ways in which it could be improved, especially by adding further features. So in this section I shall propose and briefly explore some possible ways of extending the system.

Local SPARQL Processing

Currently the application relies on 3rd party websites to process and dispatch SPARQL queries, this works, but does depend on the availability of resources outside of my control. Therefore it'd be preferable for the system to have it's own SPARQL query engine, and the JENA package which is already used by the system is capable of doing this, and it shouldn't take long to modify the existing system to use these capabilities.

Ontology Cacheing

One of the limitations of the current form system is that it requires the use of locally saved ontologies to generate SPARQL queries. There are ways of using JENA to capture an ontology from a remote data source and create a local representation of it, this would allow generating queries for virtually any RDF resource rather than limiting the available sources that conform to the metadata definitions in the database.

Different Types of Filters

Currently the application only generates one type of filter, the regex filter, which is used for filtering on text values. There are other types of filters however, for example, integer filters which allow you to create queries such as “Find me the name of all cities with a population over 2,000,000”. Using filters like this would be especially helpful for querying DBpedia.

Interface Simplification

One of the things I noticed from the Usability testing was that users never read dialog boxes, instructions or help files. They read links and options, but none of the other information on the page unless they absolutely have to. Dialog boxes in particular tend to be clicked away instantly. Instead users click on what looks likely to get them closer to their goal, ignoring help information or signs to the contrary. Then if they've made an erroneous selection, they are more likely to keep trying the same interface and eventually give up than to try a different interface that they've already written off as not getting them closer to their goal.

So from this I've reached a few conclusions:

- Interfaces should be goal orientated.
- Interfaces should be clearly marked ideally using graphics or short, concise words.
- Interfaces should be as intuitive as possible, they should look like they do what they are supposed to do.
- Don't put system messages in Dialog boxes.

Unfortunately usability testing came late in my project and I only had a short amount of time to implement these improvements, but for the future the interface could be improved on the back of these results.

Multiple Resource Queries

One of the strengths of SPARQL is that you can use it to query multiple resources at once by simply using multiple FROM clauses. Unfortunately I didn't have enough time to implement this feature into my SPARQL generation methods. It is a minor feature that would only take a short amount of time to add.

Natural Language Processing with OpenEphyra

As noted in section 4.3.1, I did find a natural language processing package that is capable of the part of speech tagging required to turn natural language into a SPARQL query. Unfortunately, this system has very poor documentation and the creator didn't respond to my emails, so I had to abandon my attempts to integrate it into the project. Given more development time it should be possible to create a natural language interface to generate SPARQL from, perhaps for querying DBpedia.

8 References

The Semantic Web

- W3C: Semantic Web Activity Overview http://www.w3.org/2001/sw/Activity.html *Accessed 02/02/2010*
- *W3C: Semantic Web http://www.w3.org/2001/sw/ Accessed 27/4/2010*
- Sir Tim Burners Lee's Blog at MIT: http://dig.csail.mit.edu/breadcrumbs/node/215 *Accessed 27/4/2010*
- W3C: RDF Specification http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ *Accessed 02/02/2010*
- DBpedia: About http://dbpedia.org/About *Accessed 26/4/2010*
- W3C: Semantic Web Blog (Ivan Herman <ivan@w3.org>) http://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation *Accessed 02/02/2010*

Motivation

- About.com “SQL Fundamentals” http://databases.about.com/od/sql/a/sqlfundamentals.htm *Accessed 13/4/2010*
- Data.gov.uk http://data.gov.uk/ *Accessed 26/4/2010*
- DBpedia: Entity Search, Find, and Explore http://dbpedia.org/fct/ *Accessed 26/4/2010*
- DBpedia: Query Builder http://querybuilder.dbpedia.org/ *Accessed 26/4/2010*

Web 3.0 Technologies

- *W3C: RDF Specification*
- DBpedia: Berlin http://dbpedia.org/page/Berlin *Accessed 14/4/2010*
- FOAF: About http://www.foaf-project.org/about *Accessed 26/4/2010*
- W3C:: SPARQL Working Group http://www.w3.org/2001/sw/DataAccess/homepage-20080115#hist *Accessed 14/4/2010*
- W3C SPARQL Definition Doc http://www.w3.org/TR/rdf-sparql-query/ *Accessed 14/4/2010*

Existing Systems

- Semantic Web Search : http://www.semanticwebsearch.com/
- DBpedia Query Builder: http://querybuilder.dbpedia.org/
- OpenLink iSPARQL: http://demo.openlinksw.com/isparql/
- Ask Jeeves/Ask.com: http://uk.ask.com/
- Wolfram Alpha http://www.wolframalpha.com/
- Read Write Web “Wolfram|Alpha: Our First Impressions” http://www.readwriteweb.com/archives/wolframalpha_our_first_impressions.php *Accessed 12/4/2010*

Development Plans

- W3C Schools Browser Statistics http://www.w3schools.com/browsers/browsers_stats.asp *Accessed 4/5/2010*

Architecture

- Linux Journal “Three-Tier Architecture” http://www.linuxjournal.com/article/3508 *Accessed 12/4/2010*
- MySQL homepage http://dev.mysql.com/tech-resources/articles/introduction-to-mysql-55.html *Accessed 23/4/2010*
- Oracle Sun Developer Network http://java.sun.com/blueprints/patterns/MVC.html *Accessed 12/4/2010*

High Level Features Investigated

- Ephyra: http://www.ephyra.info Accessed 1/5/2010

Testing and Evaluation

- Virtuoso OpenLink SPARQL Query http://dbpedia.org/sparql Accessed 6/5/2010
- W3C Schools Browser Statistics http://www.w3schools.com/browsers/browsers_stats.asp Accessed 4/5/2010

9 Maintenance Manual

9.1 Requirements

In order to run Sparkle! you will need the following programs installed and configured:

- Eclipse: http://www.eclipse.org/
- Apache Tomcat: http://tomcat.apache.org/
- MySQL Server: http://dev.mysql.com/
  - You may find it useful to download and install MySQL Workbench to provide a graphical User Interface for MySQL server.

You may need Administrator privileges in order to install and properly configure the above applications.

9.2 Quick Start

Unfortunately there isn't a quick start because several components will need to be configured for Sparkle! to run on your system/webserver. Specifically the MySQL needs to be set up so that the system can access it and any proxy information must be set in the code. Please be patient and follow the instructions below, do not skip the MySQL and proxy configuration steps, you may be able to skip Eclipse configuration if Eclipse behaves itself.

9.3 MySQL Configuration

User: root

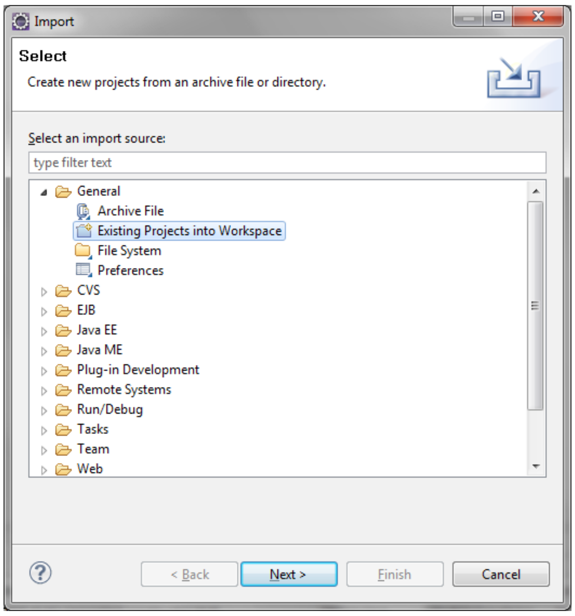
Password: web3.0

Database: predicatecache

Sparkle! interacts with MySQL though a class called "DBManager" in the search.cache package. By default, DBManager will attempt to access the MySQL database using the user name "root" and the password "web3.0". You can either set up a MySQL administrator account using this username/password combination, or, after installing the Eclipse project (below) edit the DBManager class to use a different account. Sparkle expects a database called "predicatecache" to exist on the MySQL server, so you will need to follow the MySQL tutorials and create the database using either the command line input or MySQL Workbench, no tables need to be manually added to the database.

9.4 Sparkle! Installation As an Eclipse Project

The system can be distributed as an Eclipse project inside a .tar file which must first be extracted using a program of your choice. Once the folders have been extracted, open Eclipse and choose "File ⟶ Import", in the dialog box, open the "General" tab, then choose "Existing Projects..." and click Next, as shown below:


Import

Select

Create new projects from an archive file or directory.

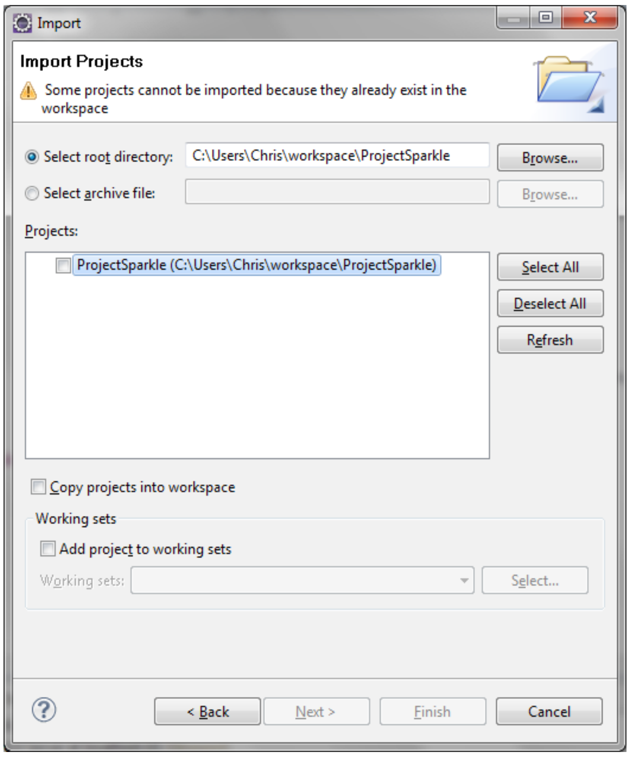
Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- CVS
- EJB
- Java EE
- Java ME
- Plug-in Development
- Remote Systems
- Run/Debug
- Tasks
- Team
- Web

< Back | Next > | Finish | Cancel

On the following dialog, click "Browse" and navigate to where you extracted the project file and select the folder called "ProjectSparkle", which should then come up as an available project. Click to select it, and optionally copy it to your workspace. Then click "Finish" and it should be imported into Eclipse automatically.


Import

Import Projects

Some projects cannot be imported because they already exist in the workspace

Select root directory: C:\Users\Chris\workspace\ProjectSparkle Browse...

Select archive file: Browse...

Projects:

ProjectSparkle (C:\Users\Chris\workspace\ProjectSparkle)

Select All | Deselect All | Refresh

Copy projects into workspace

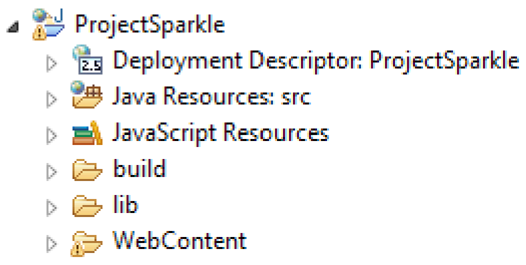
Working sets

Add project to working sets

Working sets: Select...

< Back | Next > | Finish | Cancel

Properly installed, the project should have the following file tree:


ProjectSparkle
Deployment Descriptor: ProjectSparkle
Java Resources: src
JavaScript Resources
build
lib
WebContent

Confusingly and annoyingly, Tomcat doesn't load libraries from your Eclipse project workspace (or atleast I haven't found out how to make it) so you will need to copy the ProjectSparkle libraries into Tomcat's library file. This will depend entirely on where you installed Tomcat, but the location of my Tomcat lib file was:

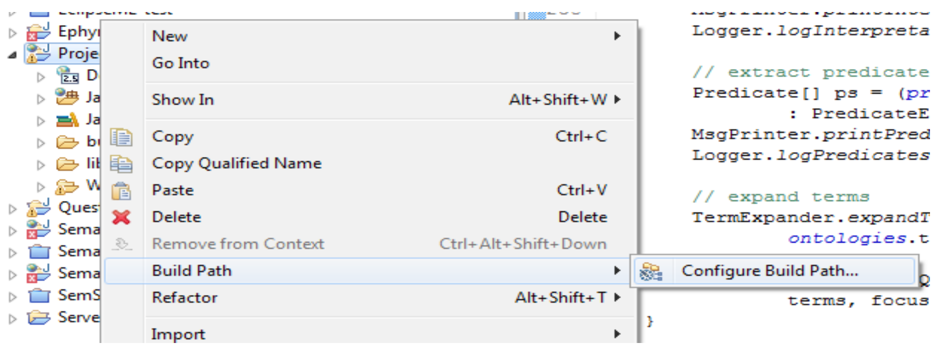
```
C:\apache-tomcat-6.0.26-windows-x86\apache-tomcat-6.0.26\lib
```

Once you've found your Tomcat/lib folder, copy and paste all the Jar files from ProjectSparkle/lib into it. Make sure you copy and paste, **don't move the files** as Eclipse requires a copy of them in the project folders so that it can build the project.

Now, before we run the system for the first time, it's worth checking the build path to make sure everything is configured correctly in Eclipse.

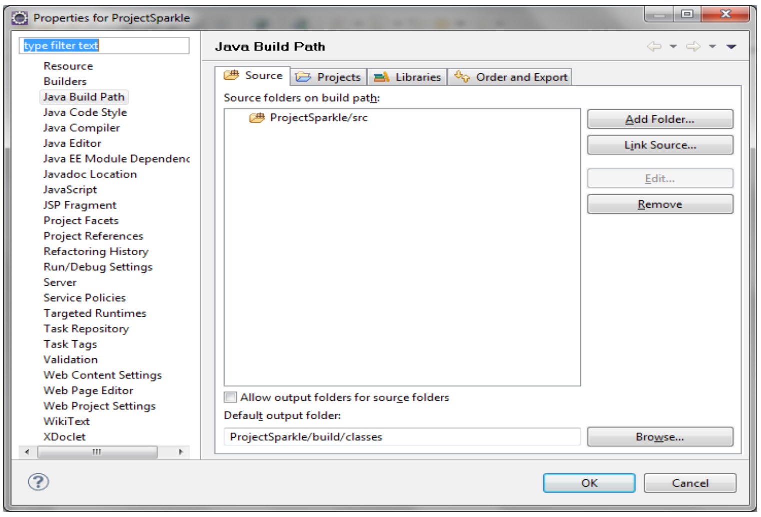
9.4.1 Configuring the Build Path

Before running the project it's worth checking the build path to ensure that Eclipse knows how to build and deploy the system. This information should be automatically entered by Eclipse from the project folder, but Eclipse is rather unreliable with it's build instructions. So, first, right click on the project folder and select "Build Path ⟶ Configure Build Path..." as shown below:



```
Logger.logInterpreta

// extract predicate
Predicate[] ps = (pr
        : PredicateE
MsgPrinter.printPred
Logger.logPredicates

// expand terms
TermExpander.expandT
        ontologies.t
                   terms, focus
}
```

You should then see a multi-tabbed dialog like the one below.



Source folders on build path:
ProjectSparkle/src

Allow output folders for source folders

Default output folder:
ProjectSparkle/build/classes

The only tabs we are interested in are Source and Libraries, the contents should be as follows:

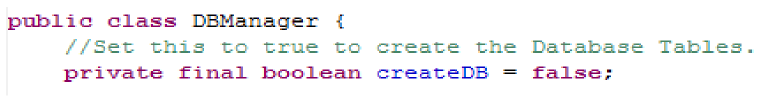
- **Source**
  - ProjectSparkle/src
    - If this is missing, click "Add Folder" and add ProjectSparkle/src
- **Libraries**
  - Every .jar file present in the ProjectSparkle/lib folder
    - If these are missing, select "Add JARs..." and select every .jar file in the lib folder.
  - Apache Tomcat v6.0
    - If this is missing you need to install Apache Tomcat (http://tomcat.apache.org/), and check you meet the rest of the Requirements listed in the section above.
  - EAR Libraries
  - JRE System Libraries (I used JRE6)
  - Web App Libraries

9.4.2 First Time Set Up

The first time you run Sparkle! You will need to set the class DBManager in src.search.cache to create the database tables the system needs and you will also need to load the data into them.

Creating the Tables

Open the src folder in the Project Explorer window in Eclipse, there should be several packages, including search.cache, open it and you should find a class called "DBManager" which manages access to the MySQL database. Open the file, and look for the class variable `createDB`, as shown below:

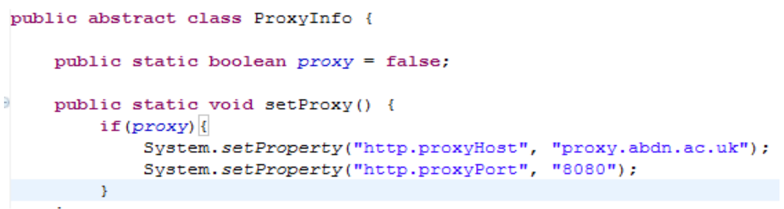


```
public class DBManager {
    //Set this to true to create the Database Tables.
    private final boolean createDB = false;
```

Set the variable to true for the first time you run the project (leaving it set to true may have a minor negative impact on performance), then build the project by right clicking on the project folder and selecting "Build Project" or alternatively use the shortcut Ctrl+B.

Setting up a Proxy (or not)

If you are behind a proxy system, you will need to set up the proxy information in the utility class search.util.ProxyInfo, shown below:



```
public abstract class ProxyInfo {

    public static boolean proxy = false;

    public static void setProxy() {
        if(proxy){
            System.setProperty("http.proxyHost", "proxy.abdn.ac.uk");
            System.setProperty("http.proxyPort", "8080");
        }
```

By default the proxy is switched off, but the system is set up for the University of Aberdeen's proxy. All you need to do is set the value of `proxy` to true and replace the proxy info used with your local proxy's host name and port number.

After editing ProxyInfo you will need to rebuild the project by right clicking on the project folder and selecting "Build Project" or the shortcut Ctrl+B.

Loading Data

Now the system has been set up so that the first time DBManager is invoked, it will create the database tables, but we still need to load data into them. For this task I created a web interface, which will become accessible when you run the project. Right click the Project Folder and select "Run As → Run on Server" if presented with a dialog box, choose "Tomcat v6.0 Server at localhost" and then Finish to run the project, if you don't get this option, Tomcat may not be set up in Eclipse.

Once the project has been loaded onto Tomcat, Eclipse should open the index page automatically, but instead navigate to "Dbload.html". Dbload contains one button marked "Load", click it and If you set createDB to true, as explained above, it will create the tables and populate them with information.

If you want to check the data in the database, you can use MySQL Workbench to manually open the database tables and check their contents.

9.5 Sparkle! Direct Deployment to Tomcat

In order to do this you will need a copy of ProjectSparkle.war and write permission to your Tomcat Server. This is the easiest way of running Sparkle, but you may need to set it up in Eclipse first, in order to configure and set up the MySQL database and the Proxy Information. For that reason I recommend first setting it up as an Eclipse project and following the configuration and first time set up steps listed as part of that set up.

Once you have got Sparkle working in Eclipse, you can then Right Click on the Project Folder → Export → WAR file, to create the necessary file.

In order to run Sparkle! On Tomcat from the War file:

1. Stop Tomcat
2. Copy the WAR file to the Tomcat/webapps folder
3. Start Tomcat

And it should now be available at [webserver URL]/ProjectSparkle

9.6 File Listing

Here I list information about all classes and most files in the project. This isn't a full in depth description of the features of each class, but rather a high level description of what function they have within the project.

I haven't described files which are part of the project or server configuration, such as WEB-INF/web.xml, and also ignores logos and icons, but otherwise, this listing is exhaustive and includes files which are extremely similar to one another on the assumption that this will be used to look up a particular file at a time rather than being sequentially read through.

Files are organised in the list first by their position in the folder hierarchy, and then alphabetically.

9.6.1 Tier 1: Client

Web Pages		
File Name	**Folder**	**Description**
DBload.html	/WebContent	Admin class for loading data into the database
DBPForm.html	/WebContent	Form input for DBpedia
DirectInput.html	/WebContent	Basic form for entering raw SPARQL queries. Provides syntax check and query functionality.
FormInput.html	/WebContent	Provides the multi-purpose interactive form input for building SPARQL queries from basic user inputs.
Index.html	/WebContent	Homepage that provides links to the system features.
AdvancedForm.html	/WebContent/forms	Template for the advanced form inputs. Loaded into FormInput.html by AJAX on user request.
BasicForm.html	/WebContent/forms	Template for basic form inputs. Loaded into FormInput.html by AJAX on user request.
ExpertForm.html	/WebContent/forms	Template for expert form inputs. Loaded into FormInput.html by AJAX on user request.
HiddenAdv.html	/WebContent/forms	File that is loaded into the advanced form div to hide the advanced form again if the user closes it after opening it.
HiddenBasic.html	/WebContent/forms	File that is loaded into the basic form div to hide the advanced form again if the user closes it after opening it.
HiddenExpert.html	/WebContent/forms	File that is loaded into the expert form div to hide the advanced form again if the user closes it after opening it.
generalQ.html	/WebContent/forms/basic	One of two possible types of query forms that can be loaded into the basic interface. Handles general queries.
specificQ.html	/WebContent/forms/	One of two possible types of query forms that can be loaded into the basic interface. Handles specific value queries.

	basic	
File Name	**Folder**	**Description**
DBPgeneralQ.html	/WebContent/forms/DBpedia	One of two possible types of query forms that can be loaded into the DBpedia interface. Handles general queries.
DBPspecificQ.html	/WebContent/forms/DBpedia	One of two possible types of query forms that can be loaded into the DBpedia interface. Handles specific queries.
ADVFormHelp.html	/WebContent/help	Help file for the Advanced Form Inputs. Loaded to FormInput.html on user request.
BASFormHelp.html	/WebContent/help	Help file for the Basic Form Inputs. Loaded to FormInput.html on user request.
empty.html	/WebContent/help	Used to hide help pages once they've been opened and then dismissed by the user.
EXPFormHelp.html	/WebContent/help	Help file for the Expert Form Inputs. Loaded to FormInput.html on user request.

Stylesheets		
File Name	**Folder**	**Description**
forms.css	/WebContent/CSS	Style information specific to the FormInput page and other inputs that are loaded to it.
general.css	/WebContent/CSS	Style information used across the entire website to ensure headers, backgrounds, text, etc styles are consistent.

AJAX Scripts		
File Name	**Folder**	**Description**
AdvancedFormScript.js	/WebContent/scripts	Scripts for sending the user information entered to the server for validation or processing as well as facilitating adding and removing user inputs on the Advanced Form.
BasicFormScript	/WebContent/scripts	Scripts for sending the user information entered to the server for validation or processing as well as facilitating adding and removing user inputs on the generalQ form.
DBPFormScript.js	/WebContent/scripts	Scripts for sending the user information entered to the server for validation or processing as well as facilitating adding and removing user inputs on the DBPgeneralQ form.
Dbtest.js	/WebContent/scripts	Scripts for initiating a load to the database. Prints back information on how many predicates were loaded.
directquery.js	/WebContent/scripts	Scripts used to check the syntax of SPARQL entered into DirectInput.html and either display errors or send the query server side and display returned results.
ExpertFormScript.js	/WebContent/scripts	Scripts for sending the user information entered in the Expert Form to the server for validation or processing.
FormControl.js	/WebContent/scripts	Contains loadHTML which is used to do some clever presentation tricks with AJAX, to load and dismiss the content of a web page to give the appearance of showing/hiding information on user request.
FormUtil.js	/WebContent/scripts	Contains several utility functions for FormInputs.html related scripts, most of which have multiple dependants.

9.6.2 Tier 2: Server

Servlets		
Class Name	**Package**	**Description**
DBServ	search.cache	Loads pre-defined RDF Metadata definitions into the database
DBPValServ	search.SPARQL	Validates user input from the DBpedia form, and generates a new form with validated inputs, allowing the user to submit the query.
ADVValServ	search.SPARQL	Validates user input from the Advanced input form, and generates a new form with validated inputs, allowing the user to submit the query.
BASValServ	search.SPARQL	Validates user input from the Basic input form, and generates a new form with validated inputs, allowing the user to submit the query.
QueryServ	search.SPARQL	Sends off queries for processing and returns the resultant xml document.
SPARQLSearch	search.SPARQL	Handles SPARQL inputed by the user from DirectInput.html
SyntaxCheck	search.SPARQL	Checks the syntax of SPARQL queries entered by the user to DirectInput.html returns an error if the query is not correctly formatted.

Utils		
Class Name	**Package**	**Description**
QueryBuilder	search.SPARQL	Builds SPARQL queries of various types from various combinations of variables and inputs.
ProxyInfo.java	search.util	Sets up the proxy information if necessary.
SPARQLfilter	search.util	Data storage class, which represents the data needed to create a FILTER clause in SPARQL
Triple	search.util	Data storage class, represents a triple to be put into the WHERE clause of a SPARQL query. The triple elements are strings representing either Properties, Classes or variables.
DBManager.java	search.cache	Provides the necessary Database access methods used by the validation servers to match Natural language to Metadata. Also includes data entry methods used by DBServ.

9.6.3 Tier 3: Data

MySQL Database Tables		
Table Name	**Attributes**	**Description**
definitions	D_ID : primary key Dname : short name URI : link to resource	Contains information about the metadata definitions stored in the database. Used to differentiate predicates by domain.
pmappings	D_ID : foreign key P_ID : foreign key	Maps the links between the definitions and predicates tables.
predicates	P_ID : primary key PredicateText: short name URI: link to the predicate definition.	Contains information about all predicates stored in the database. Used for comparing user input to known predicates.

9.6.4 Libraries

Classes		
File Name	**Package**	**Description**
arq-2.8.1.jar	Jena-2.6.2	ARQ implementation of the SPARQL query language for Jena
icu4j-3.4.4.jar	Jena-2.6.2	Part of the JENA framework
iri-0.7.jar	Jena-2.6.2	Part of the JENA framework
jena-2.6.2.jar	Jena-2.6.2	Part of the JENA framework
jena-2.6.2-tests.jar	Jena-2.6.2	Part of the JENA framework
junit-4.5.jar	Jena-2.6.2	Part of the JENA framework
log4j.1.2.13.jar	Jena-2.6.2	Part of the JENA framework
lucene-core-2.3.1.jar	Jena-2.6.2	Part of the JENA framework
mysql-connector-java-3.0.17-ga-bin.jar		Driver for connecting to MySQL databases.
slf4j-api-1.5.6.jar	Jena-2.6.2	Part of the JENA framework
slf4j-log4j12-1.5.6.jar	Jena-2.6.2	Part of the JENA framework
stax-api-1.0.1.jar	Jena-2.6.2	Part of the JENA framework
wstx-asl-3.2.9.jar	Jena-2.6.2	Part of the JENA framework
xercesImpl-2.7.1.jar	Jena-2.6.2	Part of the JENA framework

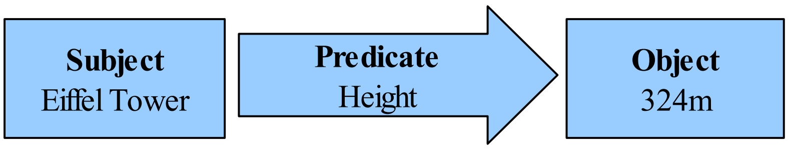
10 User Manual

10.1 Introduction

Welcome to the Sparkle! user manual, Sparkle is a Semantic question answering and query generation system developed at the University of Aberdeen. In this manual, I'll briefly describe some concepts behind Sparkle! and then cover how to use it. If you are not familiar with RDF, I strongly recommend you read the introduction to Subject-Predicate-Object section.

10.2 Introduction to Subject-Predicate-Object

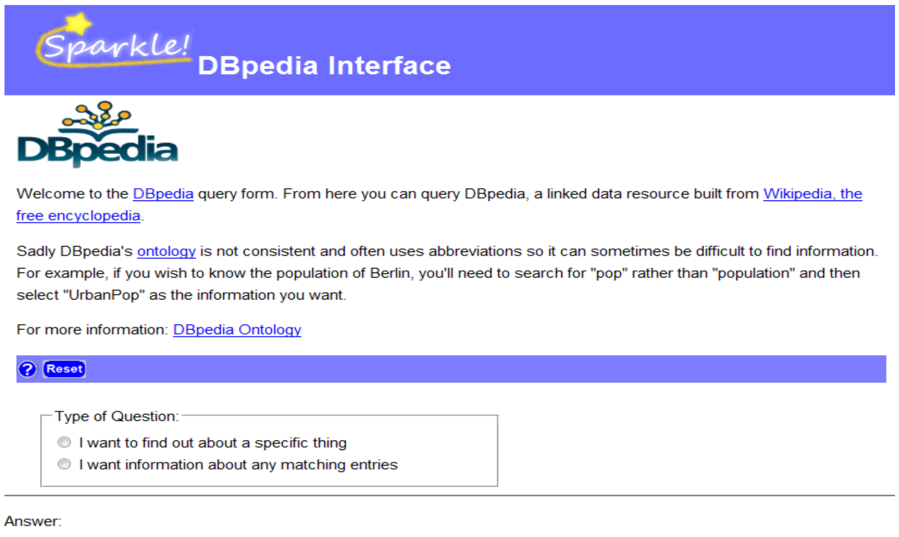
When you make a statement about the world, such as "The height of the Eiffel Tower is 324m", there are three components – the subject (the Eiffel Tower), predicate (height of) and an object (324m). The predicate maps an object to a subject, as shown below:



The Semantic Web structures information in this way, and so to query the Semantic Web, questions have to be structured accordingly.

10.3 Querying DBpedia

To run a query on DBpedia, for technical reasons you must use the DBpedia interface which you can find via a link on the index page. When you first open the DBpedia interface (shown below), you'll be presented with the choice between two types of question that you want to ask; the first is a specific value question, for example, "What is the population of Berlin?" the second is more general, for example "What cities start with the letter B?".



Welcome to the DBpedia query form. From here you can query DBpedia, a linked data resource built from Wikipedia, the free encyclopedia.

Sadly DBpedia's ontology is not consistent and often uses abbreviations so it can sometimes be difficult to find information. For example, if you wish to know the population of Berlin, you'll need to search for "pop" rather than "population" and then select "UrbanPop" as the information you want.

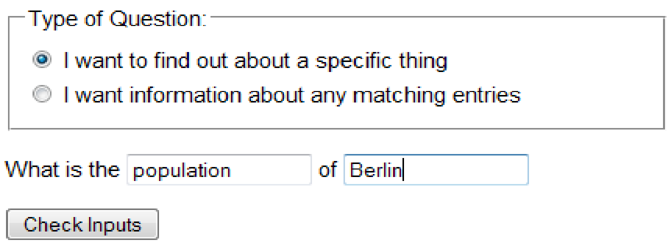
For more information: DBpedia Ontology

Type of Question:
- ○ I want to find out about a specific thing
- ○ I want information about any matching entries

Answer:

Specific Questions

When asking a specific question, select the correct interface and you will be presented with the structure of a question with two fields you can type into (as shown below) . The first field is for the property you are looking for (e.g. population) and the second field is for the subject (e.g. Berlin).

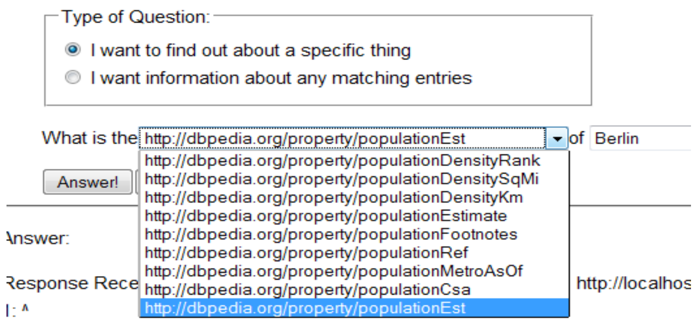

Type of Question:

[✓ I want to find out about a specific thing]

I want information about any matching entries

What is the population of Berlin

Once you have entered a property and a subject for your question, select "Check Inputs" and the system will validate your entries and provide you with a list of possible properties so you can choose the one that best matches your selection as shown below. If the system cannot find a matching property try being less specific, for example, if you entered "Urban Population" and got no results, try "Urban" by itself.


Type of Question:

[✓ I want to find out about a specific thing]

I want information about any matching entries

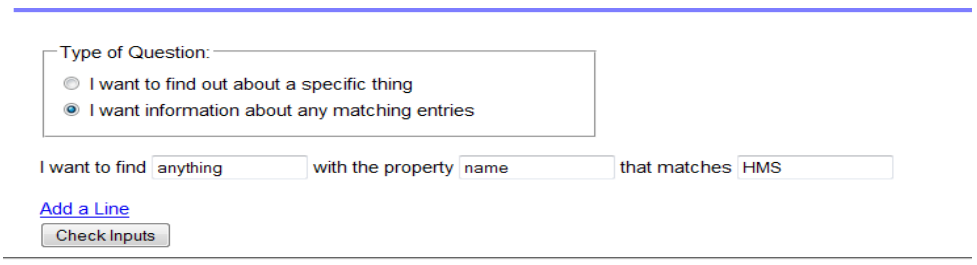
What is the http://dbpedia.org/property/populationEst of Berlin

http://dbpedia.org/property/populationDensityRank
http://dbpedia.org/property/populationDensitySqMi
http://dbpedia.org/property/populationDensityKm
http://dbpedia.org/property/populationEstimate
http://dbpedia.org/property/populationFootnotes
http://dbpedia.org/property/populationRef
http://dbpedia.org/property/populationMetroAsOf
http://dbpedia.org/property/populationCsa
http://dbpedia.org/property/populationEst

Once you are satisfied with your selection, click "Answer" to run your query. Running a query can take some time especially if DBpedia is busy, but the system will display a message once it has a result for you. If you don't get a result on the first try, you may have to select another property – DBpedia is not a consistent resource and often the answer to your question will be under a different heading than is immediately obvious.

General Questions

The second type of DBpedia query form is the "General" form, and it is for finding a broader range of results, like "Find me all British naval vessels listed on DBpedia". Like the specific form, to ask a question you must phrase it in a certain way and fill in the blanks in the question presented on the form. So, for example, if you want to find all British naval vessels, the fastest way is to look for anything with a name that contains "HMS", as shown below:

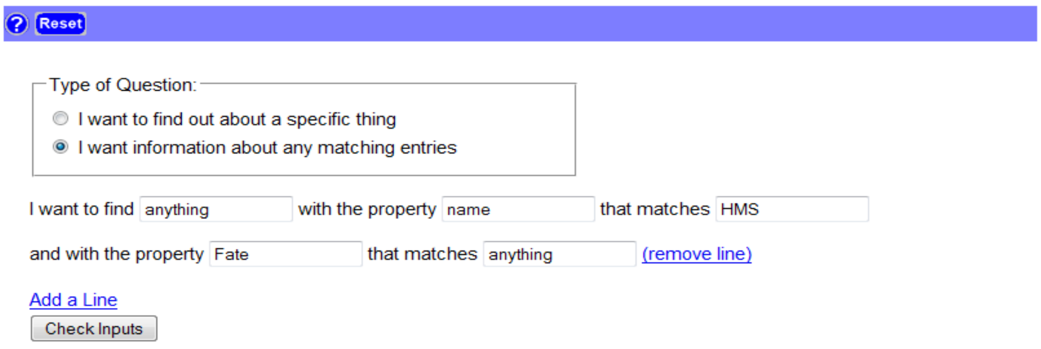

Type of Question:

I want to find out about a specific thing

[✓ I want information about any matching entries]

I want to find anything with the property name that matches HMS

Unlike the specific form, the general form can look for several different things at once, all you have to do is add an additional line and fill it out, for example we could add an additional line to the query to find out the fate of each ship, as shown below:



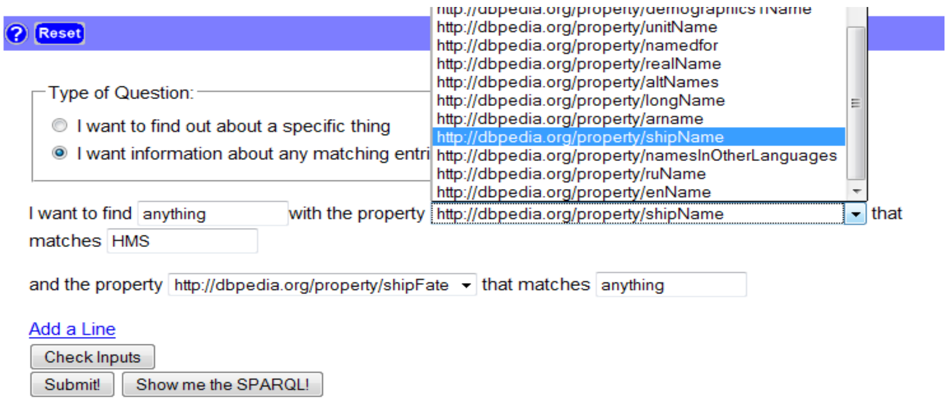
Type of Question:

I want to find out about a specific thing
[✓ I want information about any matching entries]

I want to find anything with the property name that matches HMS

and with the property Fate that matches anything (remove line)

Once more, the form has to be validated before a query can be run, so click "Check Inputs" and the system will present you with a list of options similar to what you are looking for, as shown below:



Type of Question:

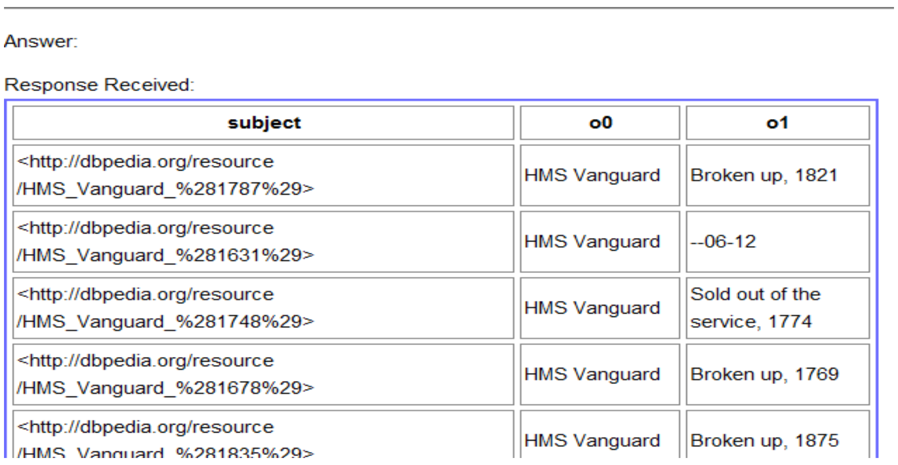
I want to find out about a specific thing
[✓ I want information about any matching entri]

http://dbpedia.org/property/demographics1Name
http://dbpedia.org/property/unitName
http://dbpedia.org/property/namedfor
http://dbpedia.org/property/realName
http://dbpedia.org/property/altNames
http://dbpedia.org/property/longName
http://dbpedia.org/property/arname
http://dbpedia.org/property/shipName
http://dbpedia.org/property/namesInOtherLanguages
http://dbpedia.org/property/ruName
http://dbpedia.org/property/enName

I want to find anything with the property http://dbpedia.org/property/shipName that matches HMS

and the property http://dbpedia.org/property/shipFate that matches anything

Here there is a specific option for "shipName" and "shipFate" which is exactly what we're looking for. Finally we hit submit, and the results are displayed below the form, as shown below.



Answer:

Response Received:

subject	o0	o1
<http://dbpedia.org/resource /HMS_Vanguard_%281787%29>	HMS Vanguard	Broken up, 1821
<http://dbpedia.org/resource /HMS_Vanguard_%281631%29>	HMS Vanguard	--06-12
<http://dbpedia.org/resource /HMS_Vanguard_%281748%29>	HMS Vanguard	Sold out of the service, 1774
<http://dbpedia.org/resource /HMS_Vanguard_%281678%29>	HMS Vanguard	Broken up, 1769
<http://dbpedia.org/resource /HMS_Vanguard_%281835%29>	HMS Vanguard	Broken up, 1875

10.4 Querying Other Resources

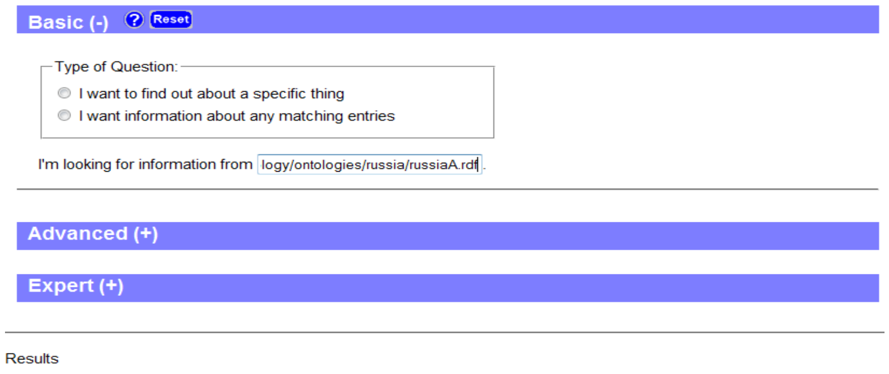
There is a general interface available for querying your own RDF resources, it is available from the index page link "General Query Form". When you first open the general form, it will have three closed tabs labeled "Basic", "Advanced" and "Expert". The Basic form allows you to form basic queries by filling in the blanks in a question exactly like the DBpedia form, the Advanced form which allows you to generate more complex SPARQL queries from a table of inputs structured like SPARQL, and finally the Expert tab which is for writing your own SPARQL queries or editing system generated ones.

Basic Interfaces

Just like the DBpedia interface, the Basic form has two options, one for specific questions and one for general questions. Unlike the DBpedia interface, you will need to specify which resource you are querying. You can also specify your own RDF properties to use, as an example I'll build a query against a resource which contains tourist information about Russia.

```
Resource:
http://www.atl.external.lmco.com/projects/ontology/ontologies/russia/russia
A.rdf
```

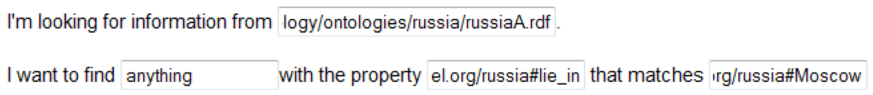
First we enter the resource into the form, as shown bellow:



Type of Question:
I want to find out about a specific thing
I want information about any matching entries

I'm looking for information from logy/ontologies/russia/russiaA.rdf

As an example, I'll create a general query to find information on tourist destinations in Moscow. To do so, I'll have to use the custom predicate "http://travel.org/russia#lie_in" and the custom resource "http://travel.org/russia#Moscow". The general form, like the DBpedia form does have facilities for matching predicates like "name" to commonly used metadata definitions, but using those facilities are covered in depth in the DBpedia tutorial above.



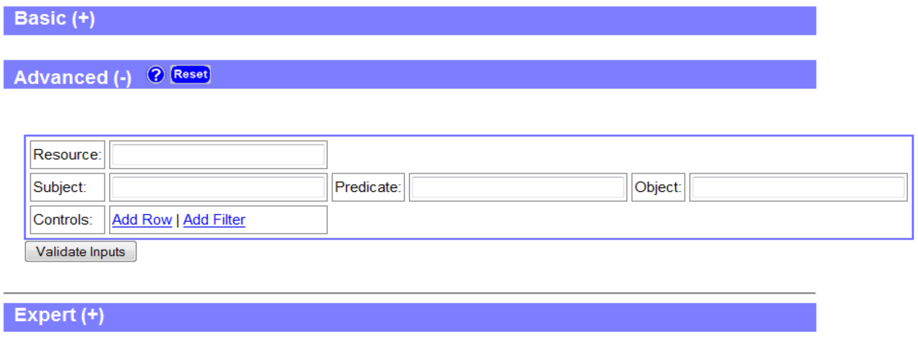
I'm looking for information from logy/ontologies/russia/russiaA.rdf

I want to find anything with the property el.org/russia#lie_in that matches ırg/russia#Moscow

Above is the form filled in with the appropriate URIs.

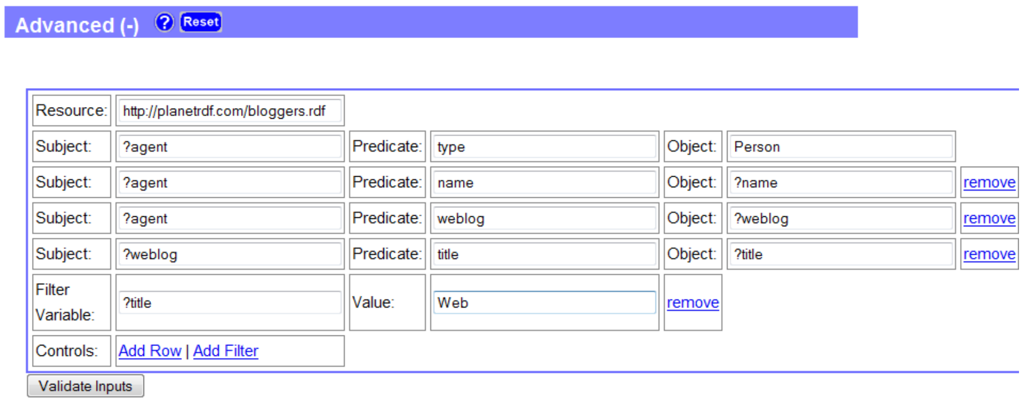
The Advanced Interface

in addition tot he basic interface, the Advanced interface provides users with some familiarity with RDF/SPARQL the tools to build more advanced queries without using SPARQL itself. The advanced form itself is a series of inputs starting with the target resource, followed by a table of constraints, shown below.



Resource:					
Subject:		Predicate:		Object:	
Controls:	Add Row | Add Filter				

The Advanced form is more extendable than the basic one, and allows you to add or remove more constraints and filters to create complex queries. An example is shown below:

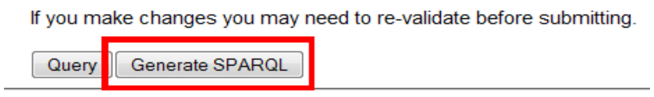


Resource:	http://planetrdf.com/bloggers.rdf					
Subject:	?agent	Predicate:	type	Object:	Person	
Subject:	?agent	Predicate:	name	Object:	?name	remove
Subject:	?agent	Predicate:	weblog	Object:	?weblog	remove
Subject:	?weblog	Predicate:	title	Object:	?title	remove
Filter Variable:	?title	Value:	Web	remove		
Controls:	Add Row | Add Filter					

Like the basic form, the Advanced form needs to be validated before you can use it to run a query.

10.5 Generating SPARQL

You can see the SPARQL code generated by any form input that has been validated, simply click "Generate SPARQL" as shown below:

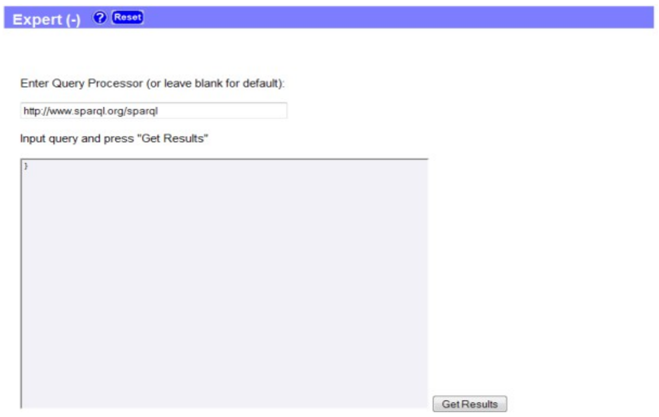


If you make changes you may need to re-validate before submitting.

On the DBpedia form this will display the SPARQL in the result box, on the general input form this will open the Expert tab and enter the SPARQL for you, you can then edit the query and run it.

10.6 Writing And Running Your Own SPARQL

Sparkle! has facilities for running your own SPARQL queries, including syntax checking. To run your own SPARQL queries against any resource, go to the "General Query Form" and expand the "Expert" tab as shown below:

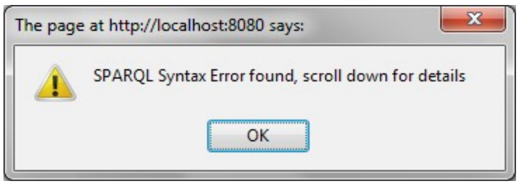


Enter Query Processor (or leave blank for default):

http://www.sparql.org/sparql

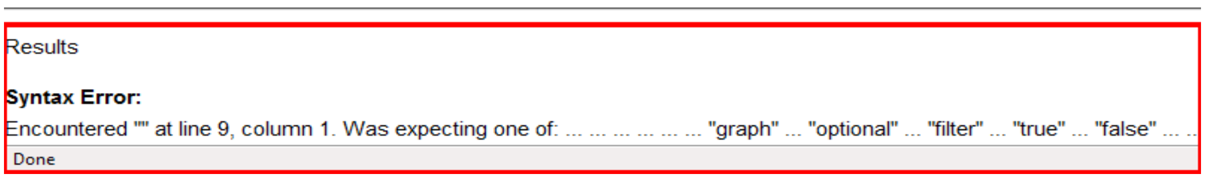
Input query and press "Get Results"

You can specify your own query processor to run the SPARQL on, or leave the box empty for the default. You can then enter your own SPARQL into the form, and click "Get Results" to run the query. If the query contains a syntax error, the system will display a dialog box with the error and then write it to the results, as shown below:



The page at http://localhost:8080 says:

SPARQL Syntax Error found, scroll down for details



Results

Syntax Error:

Encountered "" at line 9, column 1. Was expecting one of: ... ... "graph" ... "optional" ... "filter" ... "true" ... "false" ... ..

Done

Appendix A : Usability Tester's Sheet

Candidate Name:
Candidate Number:

Task	Time	Notes
1		
2		
3		
4		
5		
6		

Appendix B: Durability Testing, Tester's Sheet

Candidate Name:
Candidate Number:

Action	**Result**