School of Electronics and Computer Science
Faculty of Engineering, Science and Mathematics
University of Southampton

Daniel Alexander Smith
May 2004

OntoBrowse: A World of Knowledge

Project Supervisor: Hugh Glaser
Second Examiner: Dr John N. Carter

A project submitted for the award of
MEng Computer Science

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

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

A project report submitted for the award of MEng Computer Science

by Daniel Alexander Smith

This paper describes the creation and function of OntoBrowse, a domain-independent ontology browser system that was developed to provide generic access to any triplestore ontology without the need to create a bespoke interface.

It features support for accessing multiple triplestores in one query session, bookmarks, Rendezvous sharing of bookmarks, multiple tabs, multiple windows, namespace caching and automatic generation of RDQL queries. OntoBrowse automatically loads images when referenced by URI and has a fully customisable user interface.

In addition, the CIA World Factbook was asserted into a triplestore in order to gain a conceptual understanding of knowledge systems and for use as a controllable testing ground for the ontology browser.

Contents

Abstract **1**

Acknowledgements **4**

1 Introduction **5**

2 Technology Choices **6**
2.1 Knowledge Representation 6
2.2 RDF Storage 6
2.3 Knowledge Acquisition 7
2.4 Software Development Language 8

3 Development **10**
3.1 RDF 10
3.2 The CIA World Factbook 11
3.2.1 History 11
3.2.2 Parsing 12
3.3 UI Design 12
3.4 Multiple Triplestores 13
3.5 Namespace Caching 13
3.6 Query Generation 16
3.7 Bookmarks 17
3.8 Rendezvous Sharing of Bookmarks 19
3.9 Renderer Panel 20
3.10 Docked Panels and State 21
3.11 URIs and Literals 21
3.12 UML 23
3.12.1 Use Case 23
3.12.2 Class Diagrams 24

4 3Store **26**
4.1 Introduction 26
4.2 Installation 26
4.3 Access 27
4.4 Bug Fixes 27
4.5 Possible Improvements 27

A User Manual **28**
A.1 The Features of OntoBrowse 28
A.1.1 What is OntoBrowse for? 28
A.1.2 What else can OntoBrowse do? 28
A.1.3 Why is OntoBrowse easy to use? 29
A.1.4 Does OntoBrowse utilise any advanced technology? 29
A.1.5 Does OntoBrowse only show results? 30
A.2 Basic Usage of OntoBrowse 30
A.2.1 Create a New Query 30
A.2.2 Using Bookmarks 31
A.2.3 The Image Renderer 33

B RDQL Grammar **34**

C 3Store SQL **36**

Acknowledgements

Hugh Glaser, for regular suggestions and support in a supervisory role.

Steve Harris, for developing the triplestore software, answering my queries and accepting my bug fix.

Nick Gibbins, for providing me with the code for the aktors.org web site people ontology browser.

The Eclipse development team, for creating a wonderful Java IDE.

Sun Microsystems for giving Java to the world.

Chapter 1

Introduction

The consistent growth of the semantic web and the popularity in particular of the Resource Description Framework (RDF) in knowledge systems has revealed the need for an ontological browsing system.

Currently available systems for browsing ontological databases are often bespoke systems specific to the relevant information domains. The CS AKTiveSpace [Glaser et al., 2004] is an example of such a system.

In order to gain an advanced understanding of knowledge systems, a triplestore will be created and asserted with new data. The CIA World Factbook has been chosen as the knowledge base to be asserted. This requires the per-country pages of the web site to be parsed, all knowledge extracted, and formatted as RDF data.

Once this RDF data has been asserted into a triplestore system, an application will be created to allow the user to display and browse through the ontological knowledge of the triplestore.

The application should allow the user to browse through multiple triplestores, bookmark entries and concepts, view documents referenced by URI and preferably be cross-platform.

Chapter 2

Technology Choices

2.1 Knowledge Representation

There are two clear choices when it comes to deciding on how to represent knowledge. There is RDF, a maturing technology, widely used with a lot of supporting applications, and OWL, the Web Ontology Language [McGuinness and van Harmelen F., 2004]. OWL is comparatively new and has an optionally rich feature set, manifested in three different levels of complexity depending on whether OWL Lite, OWL DL or OWL Full are used.

OWL is particularly useful when creating web content designed to be viewed by humans as well as machines. Given that the data being used here is already in the form of a web site, there is only a need to parse into a knowledge format, with no real need to embrace the complexity of OWL for republication.

2.2 RDF Storage

One of the indicators of the maturity of RDF is number of storage solutions that exist for RDF data. One of the following must be chosen to act as the server for the ontology:

- Jena at http://www.hpl.hp.com/semweb/jena.htm
- RDFStore at http://rdfstore.sourceforge.net/

- Sesame at http://sesame.aidministrator.nl/
- 3Store at http://sourceforge.net/projects/threestore/

RDFStore and Jena are both backend systems for local query of RDF data, and as such are not suitable for this application, other similar systems also exist, in the form of PHP and Perl libraries for the query of RDF data, which do not provide the server capability that this project requires.

Sesame and 3Store however both run as server applications centrally storing and indexing RDF data, accepting RDQL queries and returning results.

In terms of ability, Sesame and 3Store are very capable of the job. 3Store offers the benefit of a speedier core engine, thanks to the choice of C over Sesame's Java. Interfacing with 3Store also offers the benefit that it is being used by a number of publically accessible ontologies run by AKT, and as such these will be usable immediately.

3Store also has the advantage of local development, i.e. it is being developed in this department, whereas Sesame is being developed primarily in the Netherlands. This allows for easier debugging of any potential server issues as well as solving troubleshooting issues.

Given these advantages, 3Store shall be used for RDF storage on this project.

2.3 Knowledge Acquisition

I have chosen to parse the data using the scripting language perl. Perl has very strong regular expression pattern matching, and as such is very powerful tool when parsing large volumes of data.

Another option would have been to use a utility called Dome [Leonard and Glaser, 2001]. Dome is a utility for harvesting external data into XML on a regular basis. If the CIA World Factbook were updated regularly without format change, Dome would be ideal. The Factbook however, is updated online once a year only, often with a format and design change. This would mean the Dome harvester would have to set up again to use the new design, something which can be done much more quickly in Perl.

2.4 Software Development Language

Before development can begin, the decision of in which language to develop must be made. In order to make the ideal choice for this project, the following constraints and considerations were taken into account:

- Given the strict time contraints of this project, the development language must not only be one which is known to deliver a good return on investment, in terms of coding time, but also one to which I am familiar.
- Due to the widespread use of different operating systems, especially within this department and the knowledge technologies community as a whole, it would be preferable if the end solution be cross-platform or at the very least be easily portable.
- It should have available a simple-to-use XML parsing API.
- It should allow the straightforward creation of an intuitive GUI, so that good HCI can be achieved.

Upon application of these considerations, the following languages remain:

- Java, using Swing for the GUI
- C/C++, with a cross-platform windowing toolkit, such as GTK+ or wxWindows for the GUI
- PHP, with extensive use of JavaScript, for a web-based system

Highlighted in the above list is an important design consideration. Whether the system be a desktop application or a web-based application.

World wide web browsers exist for virtually all platforms and thus a web-based solution would be ideal for this constraint, however in order to create a viable web-based solution that would act as dynamically as a desktop application (i.e. without a visible page refresh) one would have to employ complicated JavaScript on the client-side. While possible, the development time on such as project is not feasible for the time contraints put upon this project.

Leaving a choice between C/C++ and Java for creating a desktop application solution, the differences come in the form of the following:

- Level of personal experience with either language
- Amount of time required to produce high-quality application
- Ease of debugging
- Performance

While my own personal experience with both is enough to choose either, I have written more applications in Java, in terms of quantity as well as complexity.

Java also boasts much easier debugging, as well as a centralised API documentation.

In terms of the available Integrated Development Environments (IDEs), Java was once lacking the level of support to which was available for C/C++, through products such as Microsoft's Visual Studio for example. A new IBM-sponsored product, Eclipse, now provides similar levels of development support to the Java programmer.

While the performance of a compiled C/C++ program would be greater than that of running Java bytecode, the performance of Java is adequate for this project.

Given my own level of experience with Java, the comparatively shortened development time, ease of debugging and the benefits of the Java Swing GUI construction, C/C++ shows no advantages for this project.

The final decision has been taken to use Java, with significant use of the Swing widget system for the GUI as well as the Simple API for XML (SAX), now included in the standard Java API, for XML processing.

Chapter 3

Development

3.1 RDF

The World Wide Web holds an amazing amount of information on many subjects. While for humans this is simple to search through and use, it is more difficult from a machine-understandable perspective. Given the volume of information the web contains, it is not possible to manage it manually.

The idea behind RDF [Beckett, 2004] is to use metadata (data about data) to "catalogue" this information. It must be understood however that the distinction between data and metadata in this context is application-specific and not absolute.

RDF emphasizes facilities to enable automated processing of web resources in many ways. For example in the cataloguing of web pages, or describing collections of pages that may abstractly represent a single logical document.

At this stage RDF is known for using XML syntax, however this is only one possible incantation of RDF and alternate ways to represent RDF may emerge.

In order to achieve complete domain-neutrality, one must make no assumptions about a particular application domain and RDF does this. Through the use of an appropriate schema authored for a specific domain, the data is given reason. Schemata can be fully specific or one can make incremental updates to the base schema, and indeed to any interim schemata. The use of multiple-inheritance is entirely supported, allowing mixing of definitions as well as providing multiple views of data if required.

Primarily RDF is simply a model for representing named properties and property values, which can be modelled in an entity-relationship diagram. This basic data model consists of three object types. Resources, properties and statements. Everything that is described by RDF expressions are called resources, usually a web page or some other online resource, accessible via a named URI. A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement can be another resource, or it can be a literal. A literal is allowed to have XML markup and is not evaluated by the RDF processor.

RDF has a basic legal abbreviated syntax, which allows well-formatted XML DTDs to be directly interpreted as RDF models. This is achieved through the use of relaxed syntax definition and allowing multiple syntactic models to accurately model identical data.

Modelling containers is also defined quite specifically in RDF. There are three container types, a bag, a sequence and alternative. A bag is an unordered list, a sequence is an ordered list and alternative is a list that represents alternatives for the (single) value of a property.

3.2 The CIA World Factbook

3.2.1 History

The Central Intelligence Agency was established on 26 July 1947 and officially began operating on 18 September 1947. On 13 January 1948, the National Security Council issued Intelligence Directive (NSCID) No. 3, which authorized the National Intelligence Survey (NIS) program as a peacetime replacement for the wartime Joint Army Navy Intelligence Studies (JANIS) program. Before adequate NIS country sections could be produced, government agencies had to develop more comprehensive gazetteers and better maps. The US Board on Geographic Names (BGN) compiled the names; the Department of the Interior produced the gazetteers; and CIA produced the maps.

The Hoover Commission's Clark Committee, set up in 1954 to study the structure and administration of the CIA, reported to Congress in 1955 that: "The National

Intelligence Survey is an invaluable publication which provides the essential elements of basic intelligence on all areas of the world. There will always be a continuing requirement for keeping the Survey up-to-date." The Factbook was created as an annual summary and update to the encyclopedic NIS studies. The first classified Factbook was published in August 1962, and the first unclassified version was published in June 1971. The NIS program was terminated in 1973 except for the Factbook, map, and gazetteer components. The 1975 Factbook was the first to be made available to the public with sales through the US Government Printing Office (GPO). The Factbook was first made available on the Internet in June 1997. The year 2003 marks the 56th anniversary of the establishment of the Central Intelligence Agency and the 60th year of continuous basic intelligence support to the US Government by The World Factbook and its two predecessor programs [CIA, 2003].

3.2.2 Parsing

The process followed for the parsing of the CIA World Factbook was to first examine the web documents to ensure that the required data fields could be extracted, and to formulate a basic strategy for this extraction. As with a lot of data sources on the world wide web, the data was laid out in a tabular format, which allowed for an intelligent script to parse the data using standard methods [Dixon, 1997].

Once parsed and formed into RDF, using the field names as predicates, the data was ready to be asserted into a triplestore.

3.3 UI Design

The starting point for any UI design is to understand the data being modelled. In this case the data is in the form of RDF triples (see 3.1 for specifics on RDF). There exists no precursor example or requirements specification of how to display triples and concept relation to the user. Development of the display to the user has been very much research-based and incremental.

It has been discovered that the modelling of this data into the user interface should be in the form of lists of subjects, then predicates and then the resulting objects, as in Figure 3.3.

Using this layout, it is possible for the user to specify the subject they require from a list, taken from the result of a query made to a triplestore (See Figure 3.1). Upon choosing a subject, another query is made to the triplestore returning and displaying (in a new list) the available predicates that describe the relationships between the subject they have chosen and resulting objects (See Figure 3.2). As with the subjects, the user selects a predicate, a query is generated and sent to the server, and the objects described by the chosen subject/predicate relationship are displayed (See Figure 3.3).

Once the resultant objects are shown, the user can then select one and since an object is also a subject, the result is treated as a new subject and a query to result in a list of predicates is generated and executed (See Figure 3.4).

Utilising this mechanism, it is possible for a user to quickly browse or "surf" through the contents of a triplestore indefinately (See Figure 3.5).

3.4 Multiple Triplestores

Using the browsing metaphor of a step by step branching choice, it can[should?] be possible to choose, at any branch, the triplestore to which the resultant query is to be sent for execution. Preferably in the form of a drop down combobox [ref], pre-populated with known triplestores. These would be saved from the users previous queries. An option to clear the saved servers should be available.

3.5 Namespace Caching

In RDF, every non-literal resource and object must be referenced by a properly formatted URI. In order that the interface is not "clogged up" with these (often very long) URIs, ontoBrowse stores a cache of aliases. Once an alias is set, the user never needs to see the original URI again, for example, the aktors.org triplestore may reference a person as:

$http://www.ecs.soton.ac.uk/info/\#person - 02686$

An alias can be set for the namespace $http://www.ecs.soton.ac.uk/info/$ as ecs so that the above URI would therefore show as:

$ecs : person - 02686$

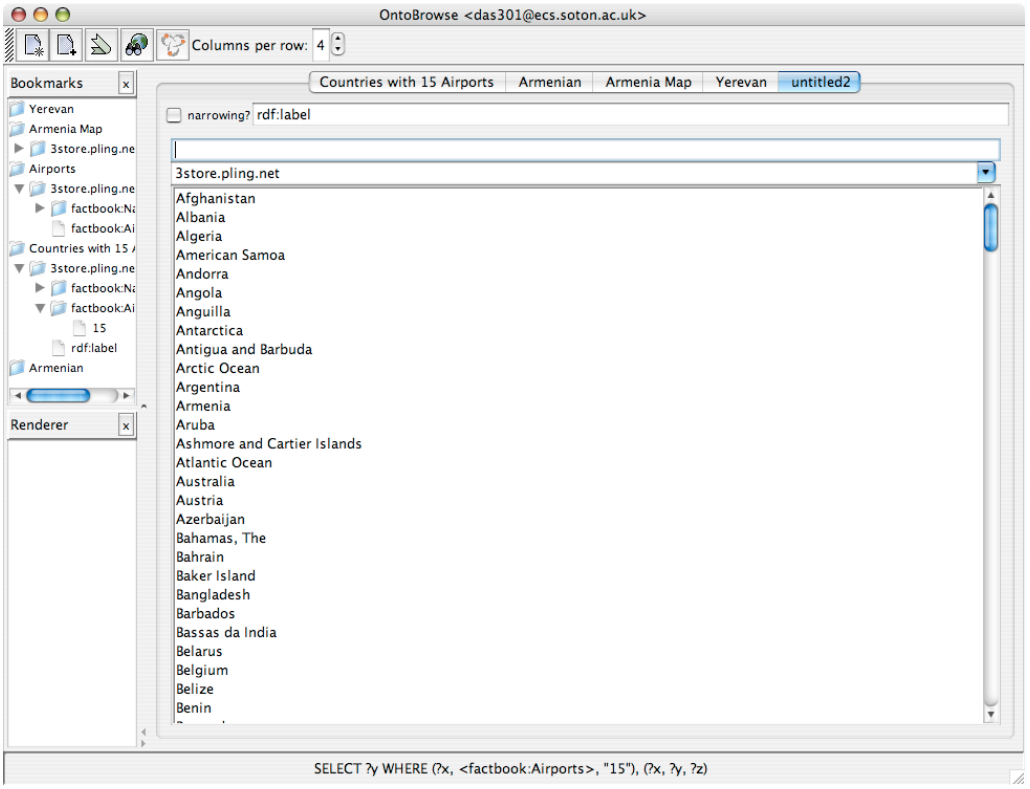

FIGURE 3.1: Single Column Screenshot

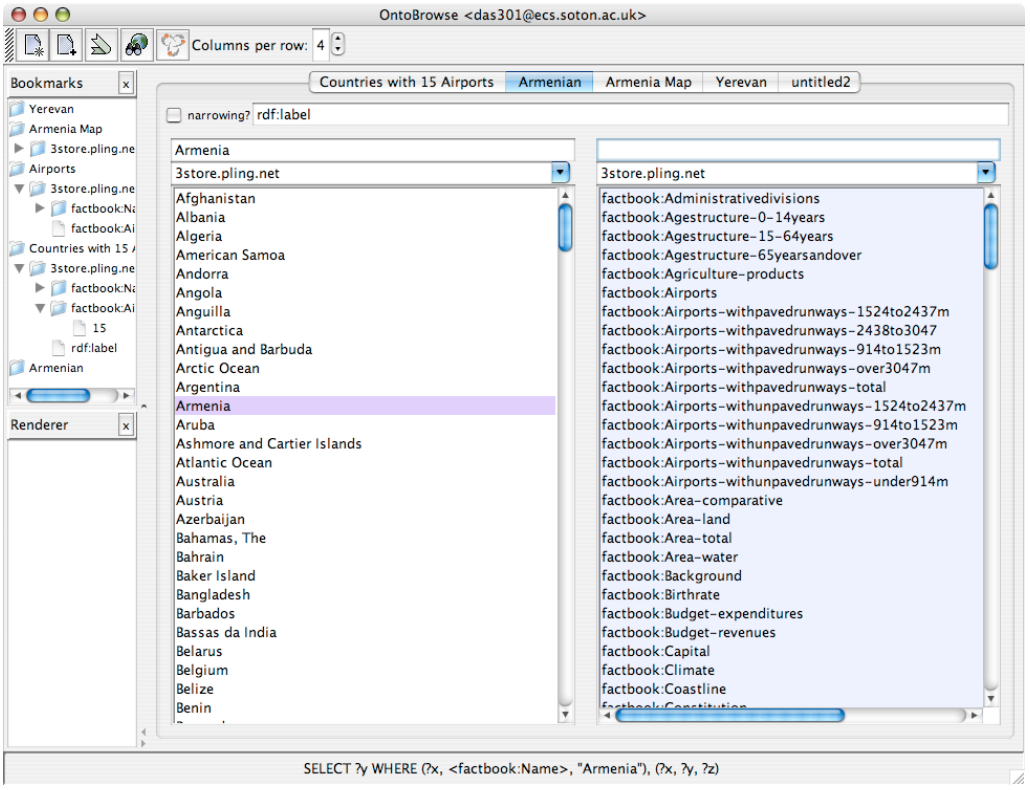

FIGURE 3.2: Two Column Screenshot

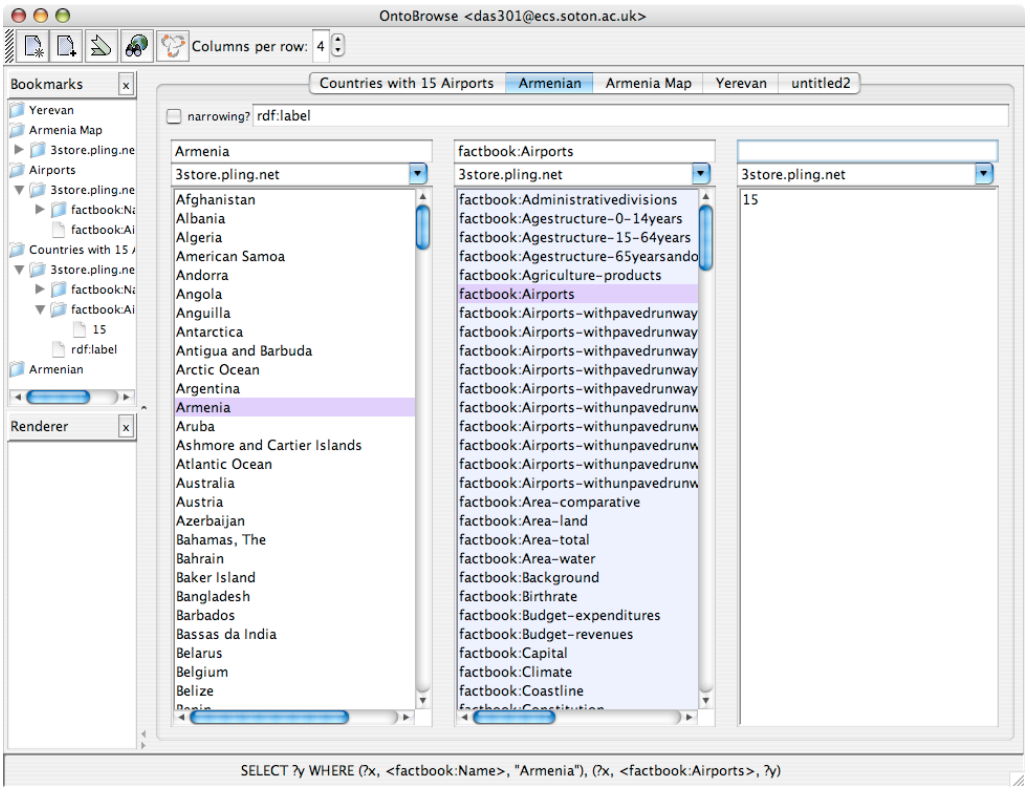

FIGURE 3.3: Three Column Screenshot

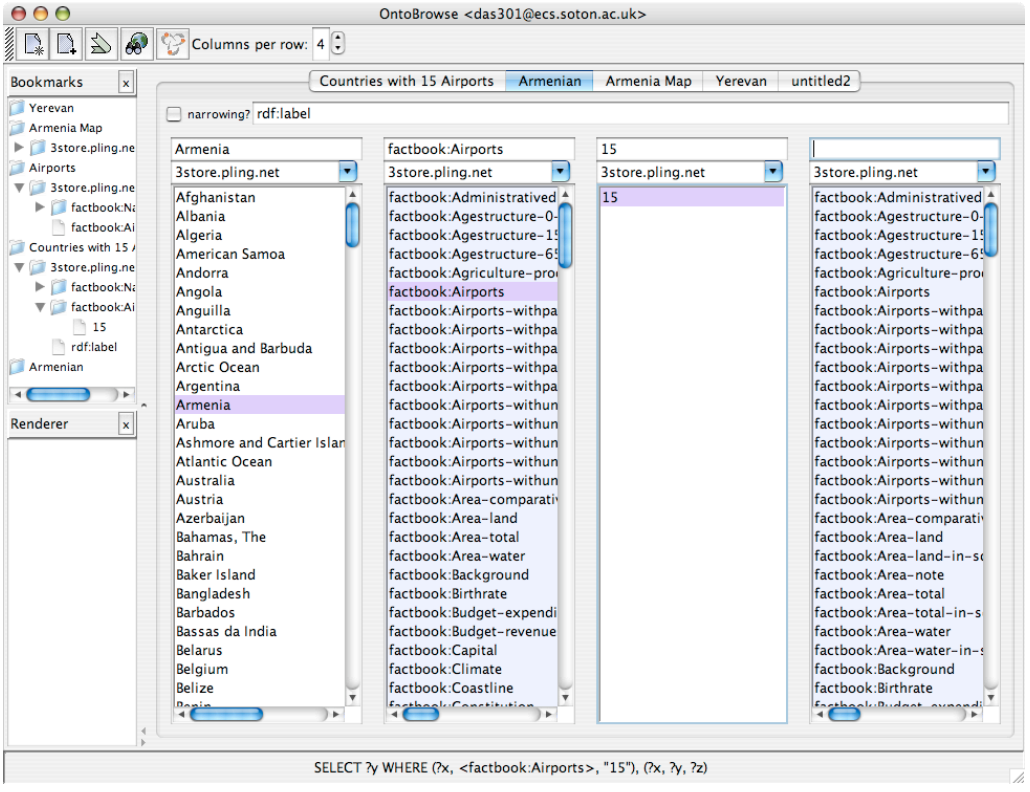

FIGURE 3.4: Four Column Screenshot

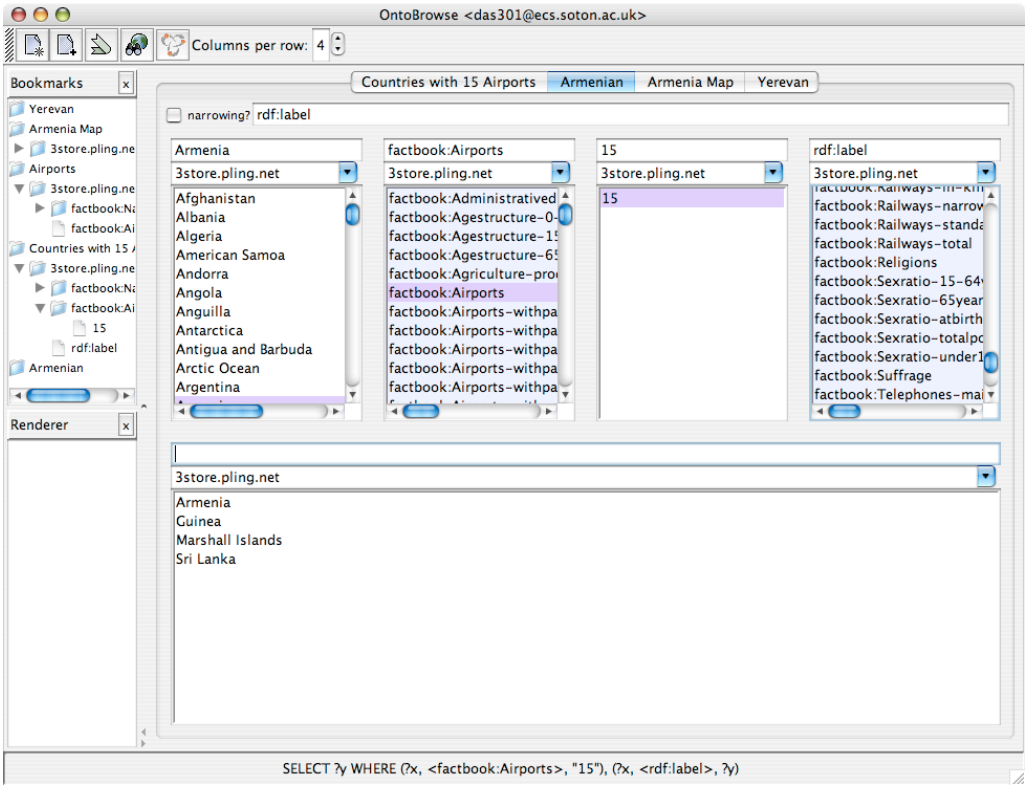

FIGURE 3.5: Five Column Screenshot

Note that when an alias is used, the octothorpe (#) symbol is changed to a colon (:), this is the standard and is required in this case so that the RDQL queries are aware this is an alias. The RDQL operator *USING* is used to state aliases, and this is automatically appended to the query where necessary.

3.6 Query Generation

One of the main abilities of ontoBrowse is to automatically generate and execute RDQL [Seaborne, 2004] queries on behalf of the user. There is, of course, no need for the user to even know what RDQL is.

The very first query is generated from an entry-point predicate chosen by the user. All objects referenced by this predicate are listed in a new column.

The initial query is generated as thus:

```
SELECT ?x WHERE (?z, <rdf:label>, ?x) USING rdf FOR
<http://www.w3.org/1999/02/22-rdf-syntax-ns>
```

This is one of the simplest forms of query, with only one triple, and indeed, only one values. The triplestore will return an XML-formatted table of results containing the value of "rdf:label" for all objects which are referenced by this predicate. In this example, the "pretty names" i.e. *rdf labels* of all objects that have them, are displayed.

The second query is generated when the user selects one of these objects from the list. The algorithm used, is designed to then show the user all predicates which apply, from the selected object, to other objects.

A query in the form of the following is generated:

```
SELECT ?x WHERE (?z, <rdf:label>, "choice"), (?z, ?x, ?y) USING
      rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns>
```

Thus, for each object which has an "rdf:label" of "choice", all predicates (designated by ?x) are returned and displayed in a list.

The remaining queries are generated in much the same way, meaning that all even columns are predicates, and all odd columns are subjects. In order to help the user distinguish them, the background colour of the predicate columns is slighty tinted.

The RDF Data Query Language (RDQL) was derived from SquishQL, an RDF query language based upon the Structured Query Language (SQL) used for querying databases.

See Appendix B for the complete RDQL grammar in Backus-Naur form (BNF).

3.7 Bookmarks

One of the most used features of a traditional (web) browsing environment is the use of bookmarks to save, return to, and share, areas of interest [Tauscher, 1996]. Given the pairing of predicates to subjects in the browsing of ontologies, the bookmarks system in ontoBrowse stores these pairings, visualising them to the user as a tree, with first order "parent" nodes representing predicates, and second order "child" nodes representing subject literals (see Figure 3.6).

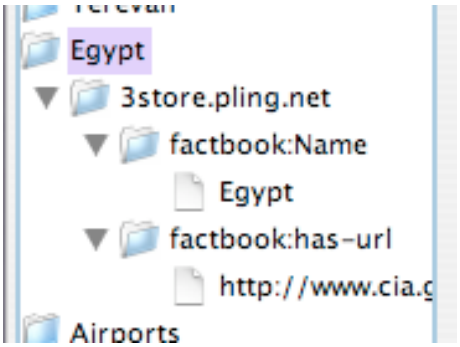

FIGURE 3.6: Bookmarks are shown as a Tree in OntoBrowse

When saved, the bookmarks are stored permanently using the Java Preferences subsystem, which provides a tree of nodes, with key/value pairs similar to that of the Microsoft Windows registry, although the actual implementation differs on each platform.

There are several different choices of bookmarking the current session available to the user. Upon right-clicking any of the results columns, the user has the choice of either bookmarking all columns, bookmarking all columns from the current column, or bookmarking all columns up to the current column. This flexibility allows the user more control over what they save, which is was found in usability testing to be especially useful when a lot of columns were shown and the user wished to backtrack without losing the current browsing point.

All saved bookmarks are shown in the bookmark panel, and a right-click context menu gives the user the following options:

- Open in New Window
- Open in New Tab
- Remove Entire Bookmark
- Remove This Pair

The above options allow the user to utilise the powerful multiple window and multiple tab interface, as well as easy manage their bookmarks, removing bookmarks that are no longer required as well as removing pairs that are no longer of interest.

3.8 Rendezvous Sharing of Bookmarks

OntoBrowse incorporates an implementation of the ZeroConf "multicast DNS" system (also known as Apple Rendezvous). When enabled (by simply toggling the toolbar button, shown in Figure 3.7), if not already saved, the user is asked for their name. The Rendezvous system then advertises that an ontoBrowse service is running at this machine's IP.

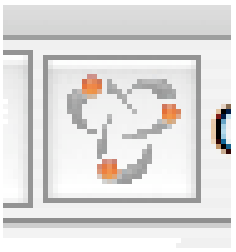

FIGURE 3.7: Rendezvous Toolbar toogle button

Other users running ontoBrowse on the same network segment will have a node added to their bookmark panel such as "Daniel Smith's bookmarks" (see Figure 3.8), which they can choose to traverse to access other users bookmarks. Right-clicking on this node offers a "Refresh" option to update the bookmarks. Network users cannot alter other users' bookmarks.

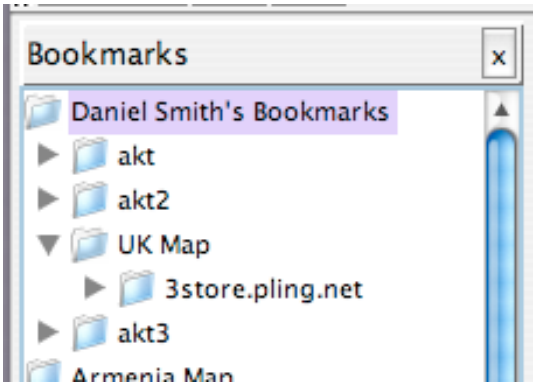

FIGURE 3.8: Another user's bookmarks are shown

To the user, this process is extremely simple, however the system behind this functionality is much less so. When the user chooses to share their bookmarks by pressed the aforementioned toolbar button, the system "announces" via Rendezvous, that a new ontoBrowse service is available at the user's IP and on a specific port, it then starts a TCP server listening on that port. Upon receiving a notification of a new ontoBrowse service being offered (either at initialisation of the Rendezvous layer, or during running), the system connects to the specified IP and

port. When a connection is made, the serving program sends an XML-formatted stream of all bookmarks as such:

```
<messagestream>
        <bookmark>
                <name>Armenia Map</name>
                <server>3store.pling.net</server>
                <type>1</type>
                <entry type="uri">factbook:Name</entry>
                <entry type="literal">Armenia</entry>
                <entry type="uri">factbook:has-map</entry>
        </bookmark>
        <bookmark>
                <name>Armenian</name>
                <server>3store.pling.net</server>
                <type>1</type>
                <entry type="uri">factbook:Name</entry>
                <entry type="literal">Armenia</entry>
        </bookmark>
        ...
</messagestream>
```

The connecting client decodes this stream, and displays it to the user via the bookmark panel. If the user decides to stop sharing bookmarks or when the program is closed, the Rendezvous service notification is expired and removed.

3.9 Renderer Panel

When results are shown, the rendering panel parses them for any literal URIs that reference any document types that that renderer knows how to display, which is currently PNG, JPEG and GIF images. The renderer will scale them (retaining correct aspect ratio) to the width of the panel, stacking them vertically, with a scroll facility when required, a feature which found particular favour in usability testing.

This feature is particularly useful when the user is browsing for an item, and there is a predicate such as (in the case of the CIA World Factbook) “has-map” which

returns a the literal URI where a map image can be downloaded. The user clicks on "has-map" and as the result is shown, the map is also automatically shown in the renderer (see Figure 3.9).

FIGURE 3.9: A map of Iraq has been rendered

3.10 Docked Panels and State

The bookmarks and renderer panels are implemented using a generic panel architecture, which means that not only is maintenance of the source code a lot simpler, but also that the preference of the user is maintained quite gracefully. OntoBrowse remembers which panels the user had open, and opens only those as opened before, retaining almost all state as when it was last closed. If fully stateful operation was required, this could be implemented quite easily by creating temporary bookmarks of all open windows and tabs.

3.11 URIs and Literals

Data returned by the triplestore in response to a query is formatted as an XML-formatted table. Passing the query

```
SELECT ?x WHERE (?y,
<http://www.ecs.soton.ac.uk/~das301/factbook/#Name>, ?x)
```

to the CIA World Factbook triplestore, will yield the following response:

```
<?xml version='1.0' encoding='UTF-8'?>
<table>
<row>
<column name="x" type="literal">Cape Verde</column>
</row>
<row>
<column name="x" type="literal">Peru</column>
</row>
<row>
<column name="x" type="literal">Virgin Islands</column>
</row>
...
</table>
```

Since only one variable ("x") is requested, there is only one column, always named "x" containing the results. In this case, since the names of the countries are requested, the results are all literals. This indicates that the data is not a reference, it is simply textual data provided as-is.

Sometimes however the type is returned as "uri". This indicates that the data returned for that particular entry is in fact a reference.

This occurs when we pass a query such as

```
SELECT ?y WHERE (?y,
<http://www.ecs.soton.ac.uk/~das301/factbook/#Name>, ?x)
```

to the same triplestore as above, resulting in:

```
<table>
<row>
<column name="y" type="uri">
http://www.cia.gov/cia/publications/factbook/print/cv.html
```

```
</column>
</row>
<row>
<column name="y" type="uri">
http://www.cia.gov/cia/publications/factbook/print/pe.html
</column>
</row>
<row>
<column name="y" type="uri">
http://www.cia.gov/cia/publications/factbook/print/vq.html
</column>
</row>
...
</table>
```

The data is clearly in the form of a URI, and as the "type" attribute is set as "uri" this is enforced as a reference, in this case, the countries are being referenced by the URI of the CIA World Factbook pages.

The main reason that these must be carefully handled by ontoBrowse is that when performing an RDQL query, any URI must be surrounded with "greater than"/"less than" brackets as in the triple:

```
<http://www.cia.gov/cia/publications/factbook/print/vq.html>, ?x, ?z
```

OntoBrowse handles these properly, with the type always being attached to the data through the use of a particular java class. This is maintained throughout saving bookmarks, sharing bookmarks and all queries.

3.12 UML

3.12.1 Use Case

A use case diagram showing the actors that interect with the browser are shown in Figure 3.10, namely the user and the triplestore server. The user actor selects a subject or predicate, whilst the triplestore server actor sends results back to the browser.

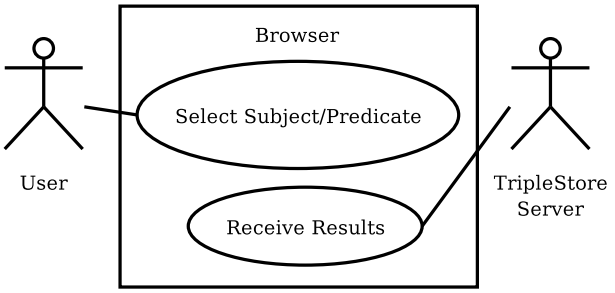

FIGURE 3.10: Use Case Diagram

3.12.2 Class Diagrams

For a breakdown of the important classes and links in the system, see Figure 3.11.

Most importantly in this diagram, is the notion that the main application class `OntoBrowse` incorporates the `ontoPanel`, which contains multiple `browsePanel`s. The other part of the program, incorporated through the `pluginPanel`, is the `rendererPanel` and `bookmarkPanel`. The `bookmarkPanel` contains multiple bookmark entries, which are implement the `unspecificBookmarkEntry` interface.

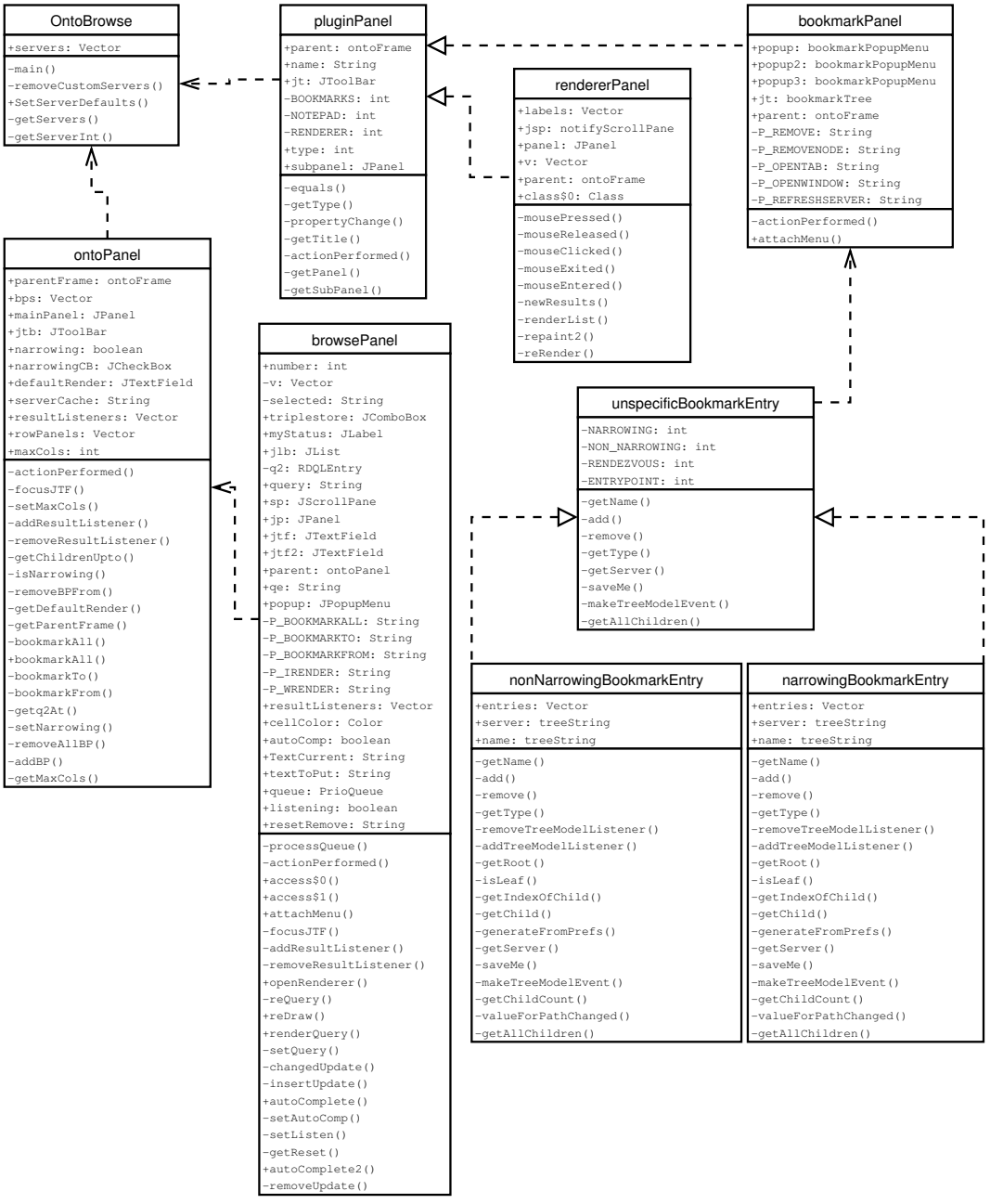

FIGURE 3.11: Class Diagram

Chapter 4

3Store

4.1 Introduction

3store is an RDF triplestore, released under the GNU General Public Licence. It is an infrastructure server for the Semantic Web.

Its purpose is to store large amounts of RDF data in a way that it can be queried quickly. It is developed in the Intelligence, Agents and Multimedia research group at Southampton University, as part of the AKT Project.

3store is a core C library that uses MySQL to store its raw RDF data and caches.

The library offers OKBC and RDQL query interfaces, over HTTP (via an Apache web server module), or directly though the C library.

4.2 Installation

Installation of 3store was performed on a server running Debian Linux (Unstable branch, 2.4 series kernel). Although initial confusion was encountered regarding the correct install procedure for the Apache web server module and some dependencies, mainly due to the installation instructions assuming a RedHat Linux system, the 3store was installed with few problems.

To assert RDF data into the 3store, one has to first create the database and import the default table structure (for the SQL, See Appendix C). Once this has been

completed, simply pass the RDF data one wishes to assert, to the `tstore_import` utility, which takes care of putting the RDF into the database.

4.3 Access

There are two methods of performing remote queries on the triplestore, Open Knowledge Base Connectivity (OKBC) and HTTP-RDQL. OKBC uses a lisp-like syntax and would take much longer to implement than the RDQL method, which is what ontoBrowse uses.

When the Apache module is installed, a handler is created on the web server, defaultly located at /rdql/. To send RDQL queries to the triplestore, the client sends a `GET` request to the server, with the server variable "query" holding the RDQL query. The server will then return the XML-formatted results.

4.4 Bug Fixes

While working with the triplestore server, a bug was discovered in the included Apache module, which is used by the server to allow access to the triplestore data via an HTTP-RDQL query (the method OntoBrowse uses).

The bug was concerned, specifically, with the return of headers to the client, which was affecting the correct behaviour of not only OntoBrowse, but also the default query interface that ships with triplestore.

This was debugged by myself and fixed. A patch was sent to the 3store developer Steve Harris and I was informed that this would be included in the next release of the software.

4.5 Possible Improvements

A more user-friendly way to rebuild the database could be incorporated into 3store. The method used for rebuilding the triplestore could be incorporated into the web interface via the Apache module. As described above, in order to rebuild the data, the database must be manually cleared and default table structure created, before data can be imported.

Appendix A

User Manual

A.1 The Features of OntoBrowse

A.1.1 What is OntoBrowse for?

OntoBrowse provides an easy-to-use interface for browsing large ontological databases. These databases, known as *triplestores* are asserted with RDF formatted data. The TripleStore software utilises a fast MySQL database backend to enable results of RDF Data Query Lanuage (RDQL) queries to be returned very quickly. Queries are sent to the server using the HyperText Transfer Protocol (HTTP) and eXtensible Markup Language (XML) formatted data is returned.

A.1.2 What else can OntoBrowse do?

OntoBrowse is much more than a graphical user interface (GUI) for these RDQL queries. It allows the user to migrate subsequent queries through multiple different servers. This could be used, for example, when a user wishes to find out about a subject referenced in triplestore A, they can further explore the subject in triplestore B.

A.1.3 Why is OntoBrowse easy to use?

One of the major features of OntoBrowse is the easy-to-use GUI. OntoBrowse utilises many well-known techniques to aid the cognitive use of the software. OntoBrowse features multiple windows, multiple tabs, toolbars and dock panels. This allows anybody who has used web browsing software such as *Netscape Navigator* or *Microsoft Internet Explorer* to be able to simply "jump in" and use OntoBrowse without any specific training.

A.1.4 Does OntoBrowse utilise any advanced technology?

OntoBrowse also features a complete bookmarks solution, including one-click network sharing of bookmarks using Apple Rendezvous (*ZeroConf*), See Figure A.1.

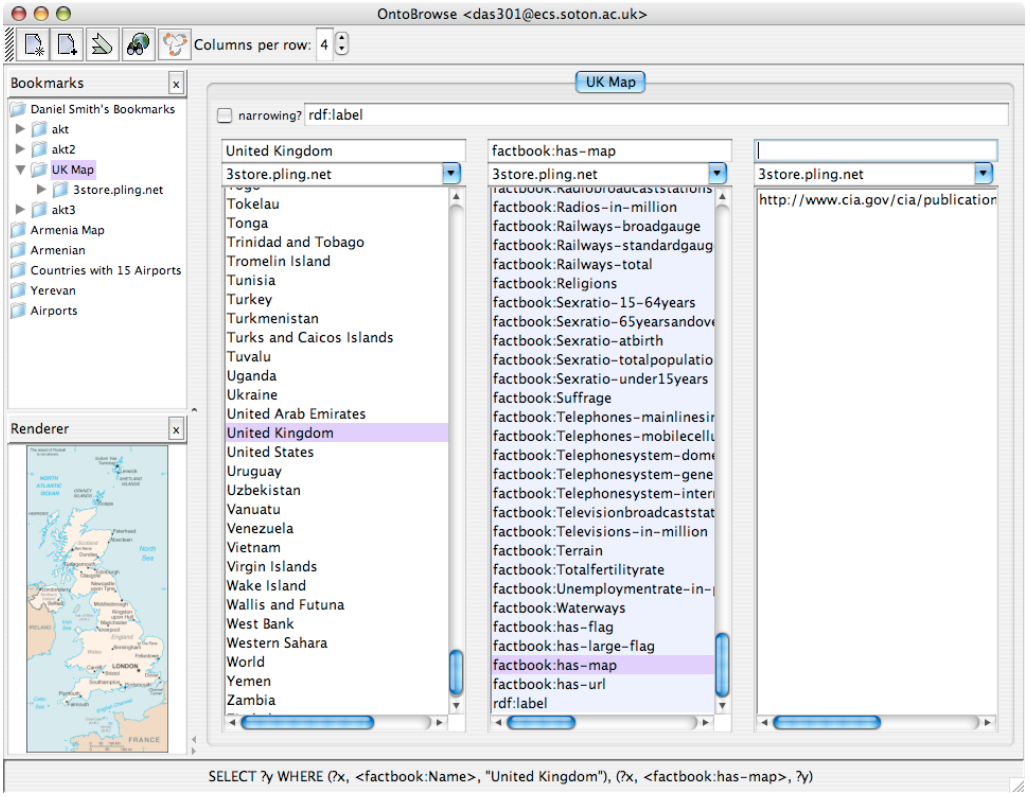

FIGURE A.1: Sharing Bookmarks with Rendezvous

A.1.5 Does OntoBrowse only show results?

Ontologies often reference external documents and media via a literal URI. OntoBrowse detects when the results of a query contain such documents and renders them automatically in the rendering panel (See Figure A.1).

A.2 Basic Usage of OntoBrowse

A.2.1 Create a New Query

To start a query, click the second button on the toolbar that looks like a document with a + symbol on it (See Figure A.2).

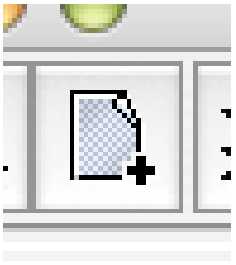

FIGURE A.2: New Tab Toolbar Button

Once this button has been pressed, a dialog will appear, prompting for a server to be chosen (See Figure A.3). Choose a server from the list, or if you wish to enter the hostname of another server, choose `-other-`, and the prompt will allow you to enter any name you wish.

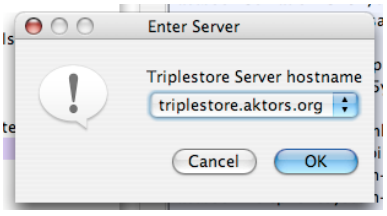

FIGURE A.3: Server Prompt Dialog

Once a server has been chosen, a dialog prompting for a start predicate will appear (See Figure A.4), you should enter a start point predicate, i.e. `rdf:label`.

The main area of the window will show a column containing the result of all objects will are referenced by this predicate, as in Figure A.5 where `factbook:Name` has been entered into the CIA World Factbook, and hence has returned all the names of countries.

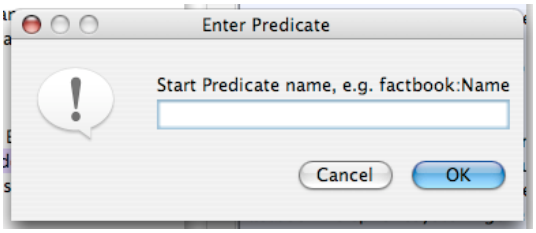

FIGURE A.4: Predicate Prompt Dialog

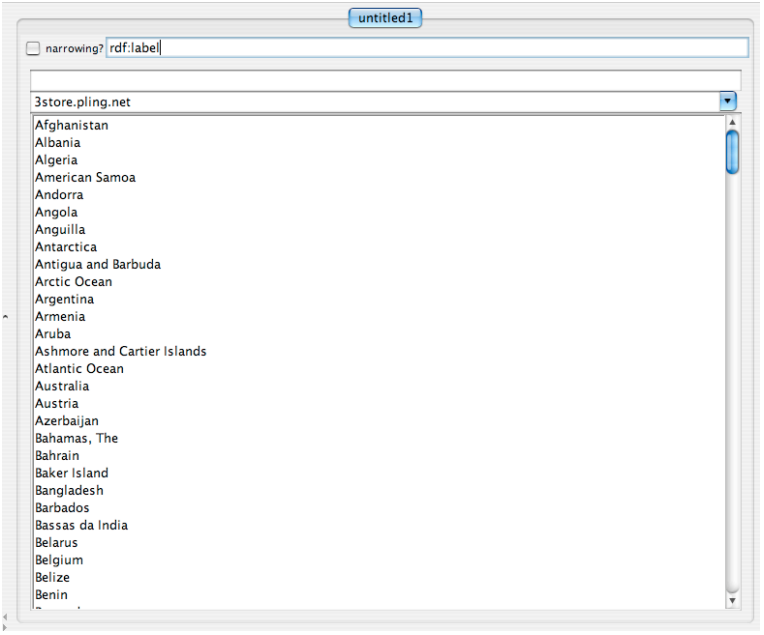

FIGURE A.5: All Country Names

A.2.2 Using Bookmarks

If there is not a panel shown entitled "Bookmarks" you will need to open it using the bookmarks toolbar button, which looks like two green bookmarks overlaid at 45 degrees (See Figure A.6). Click this icon to toggle the bookmarks panel.

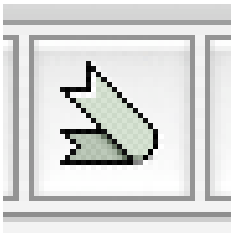

FIGURE A.6: Bookmarks Toolbar Toggle Button

To save all, or a portion of the current query as a bookmark, simply right-click on the required column and choose one of the bookmark options, as in Figure A.7.

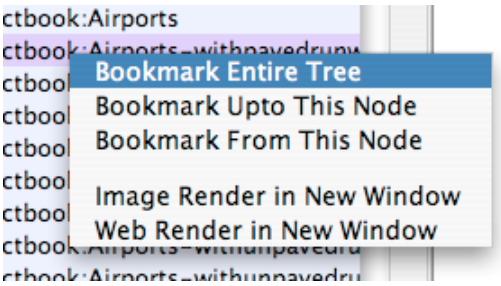

FIGURE A.7: Bookmark Options

When an option is chosen, a dialog will prompt for a name for the bookmark (See Figure A.8), enter then name you wish to use.

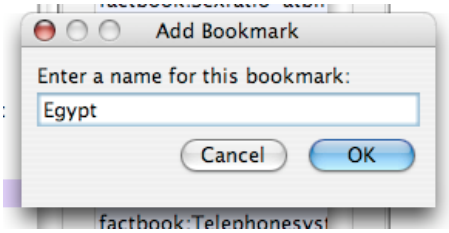

FIGURE A.8: Bookmark Name Prompt

In the bookmarks panel, the new bookmark will be shown, as in Figure A.9.

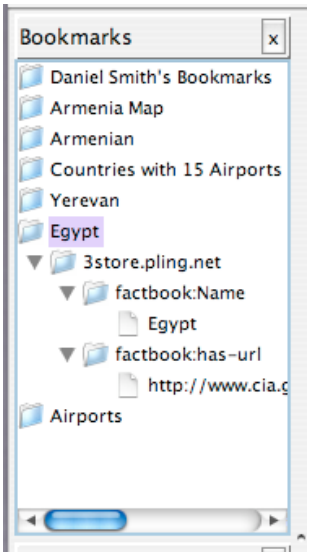

FIGURE A.9: Bookmark Panel, showing new bookmark "Egypt"

To load a bookmark, simply right-click on the bookmark and select either Open in New Tab, or Open in New Window, as in Figure A.10.

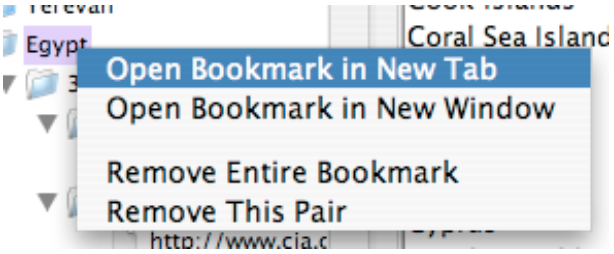

FIGURE A.10: Opening a bookmark

A.2.3 The Image Renderer

When a query results in URIs being returned in a column, such as a map or flag (see Figure A.12), the image is downloaded and shown in the Renderer Panel. To toggle the visibility of the Image Renderer, depress the renderer toolbar button, which looks like a pair of binoculars over a globe (see Figure A.11).

FIGURE A.11: Image Renderer Toolbar Toggle Button

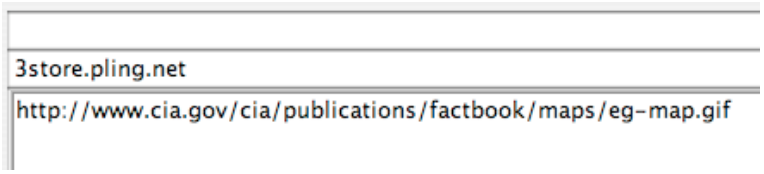

FIGURE A.12: Image URI Returned

Appendix B

RDQL Grammar

A Backus-Naur Form (BNF) of the RDQL grammar:

```
CompilationUnit ::= Query <EOF>

Query ::= SelectClause ( SourceClause )?  TriplePatternClause ( ConstraintClause )?  (
PrefixesClause )?

SelectClause ::= ( <SELECT> Var ( "," Var )* | <SELECT> "*" )

SourceClause ::= ( <SOURCE> | <FROM> ) SourceSelector

SourceSelector ::= URL

TriplePatternClause ::= <WHERE> TriplePattern ( "," TriplePattern )*

ConstraintClause ::= <SUCHTHAT> Expression ( ( "," | <SUCHTHAT> ) Expression )*

TriplePattern ::= <LPAREN> VarOrURI "," VarOrURI "," VarOrLiteral <RPAREN>

VarOrURI ::= Var | URI

VarOrLiteral ::= Var | Literal

Var ::= "?" Identifier

PrefixesClause ::= <PREFIXES> PrefixDecl ( "," PrefixDecl )*

PrefixDecl ::= Identifier <FOR> URI

Expression ::= ConditionalOrExpression

ConditionalOrExpression ::= ConditionalXorExpression ( <SC_OR> ConditionalXorExpression )*

ConditionalXorExpression ::= ConditionalAndExpression

ConditionalAndExpression ::= ValueLogical ( <SC_AND> ValueLogical )*

ValueLogical ::= StringEqualityExpression
```

```
StringEqualityExpression ::= NumericalLogical ( <STR_EQ> NumericalLogical | <STR_NE>
NumericalLogical )*

NumericalLogical ::= InclusiveOrExpression

InclusiveOrExpression ::= ExclusiveOrExpression ( <BIT_OR> ExclusiveOrExpression )*

ExclusiveOrExpression ::= AndExpression ( <BIT_XOR> AndExpression )*

AndExpression ::= ArithmeticCondition ( <BIT_AND> ArithmeticCondition )*

ArithmeticCondition ::= EqualityExpression

EqualityExpression ::= RelationalExpression ( <EQ> RelationalExpression | <NEQ>
RelationalExpression )?

RelationalExpression ::= NumericExpression ( <LT> NumericExpression | <GT> NumericExpression |
<LE> NumericExpression | <GE> NumericExpression )?

NumericExpression ::= ShiftExpression

ShiftExpression ::= AdditiveExpression ( <LSHIFT> AdditiveExpression | <RSIGNEDSHIFT>
AdditiveExpression | <RUNSIGNEDSHIFT> AdditiveExpression )*

AdditiveExpression ::= MultiplicativeExpression ( <PLUS> MultiplicativeExpression | <MINUS>
MultiplicativeExpression )*

MultiplicativeExpression ::= UnaryExpression ( <STAR> UnaryExpression | <SLASH> UnaryExpression
| <REM> UnaryExpression )*

UnaryExpression ::= UnaryExpressionNotPlusMinus | ( <PLUS> UnaryExpression | <MINUS>
UnaryExpression )

UnaryExpressionNotPlusMinus ::= ( <TILDE> | <BANG> ) UnaryExpression | PrimaryExpression

PrimaryExpression ::= Var | Literal | FunctionCall | <LPAREN> Expression <RPAREN>

FunctionCall ::= Identifier <LPAREN> ArgList <RPAREN>

ArgList ::= VarOrLiteral ( "," VarOrLiteral )*

Literal ::= URI | NumericLiteral | TextLiteral | BooleanLiteral | NullLiteral

NumericLiteral ::= ( <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> )

TextLiteral ::= <STRING_LITERAL>

BooleanLiteral ::= <BOOLEAN_LITERAL>

NullLiteral ::= <NULL_LITERAL>

URL ::= URI

URI ::= "<" <URI> ">"


Identifier ::= <IDENTIFIER>
```

Appendix C

3Store SQL

The following SQL creates properly formatted empty database tables for use with the 3store server software.

Taken from 3store release v2.2.8.

```
USE rdf;

CREATE TABLE models (
  hash bigint(20) NOT NULL default '0',
  model text NOT NULL,
  PRIMARY KEY (hash)
) TYPE=MyISAM;

CREATE TABLE literals (
  hash bigint(20) NOT NULL default '0',
  literal text NOT NULL,
  PRIMARY KEY (hash)
) TYPE=MyISAM;

CREATE TABLE resources (
  hash bigint(20) NOT NULL default '0',
  uri varchar(255) NOT NULL default '',
  PRIMARY KEY (hash),
  KEY (uri)
) TYPE=MyISAM;
```

```
CREATE TABLE ‘triples‘ (
  ‘model‘ bigint(20) NOT NULL default ’0’,
  ‘subject‘ bigint(20) NOT NULL default ’0’,
  ‘predicate‘ bigint(20) NOT NULL default ’0’,
  ‘object‘ bigint(20) NOT NULL default ’0’,
  ‘literal‘ tinyint(1) NOT NULL default ’0’,
  ‘inferred‘ tinyint(1) NOT NULL default ’0’,
  UNIQUE KEY ‘spo‘ (‘subject‘,‘predicate‘,‘object‘),
  KEY ‘o‘ (‘object‘),
  KEY ‘po‘ (‘predicate‘,‘object‘)
) TYPE=MyISAM;

CREATE TABLE ‘taxonomy‘ (
  ‘class‘ bigint(20) NOT NULL default ’0’,
  ‘superclass‘ bigint(20) NOT NULL default ’0’,
  KEY ‘class‘ (‘class‘),
  KEY ‘superclass‘ (‘superclass‘)
) TYPE=MyISAM;

CREATE TABLE ‘cache_state‘ (
  ‘taxonomy‘ int(11) NOT NULL default ’0’,
  ‘sia‘ int(11) NOT NULL default ’0’
) TYPE=MyISAM;

INSERT INTO cache_state VALUES();
```

Bibliography

D. Beckett. Rdf/xml syntax specification (revised), 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

National Technical Information Service CIA. *The CIA World Factbook*. US Government Printing Office, 2003.

M. Dixon. An overview of document mining technology, 1997. URL `citeseer.ist.psu.edu/dixon97overview.html`.

H. Glaser, H. Alani, L. Carr, S. Chapman, F. Ciravegna, A. Dingli, N. Gibbins, S. Harris, m. c. schraefel, and N. Shadbolt. Cs aktive space: Building a semantic web application. *Proceedings of European Semantic Web Symposium 2004*, 2004.

T. Leonard and H. Glaser. Large scale acquisition and maintenance from the web without source access. In *Proceedings of Workshop 4, Knowledge Markup and Semantic Annotation*, pages 97–101, 2001.

D. McGuinness and van Harmelen F. Owl web ontology language overview. *http://www.w3.org/TR/owl-features/*, 2004.

A Seaborne. Rdql - a query language for rdf, 2004. http://www.w3.org/Submission/RDQL/.

L. Tauscher. Supporting world-wide web navigation through history mechanisms. In *CHI 96 Workshop: HCI and the Web*. University of Calgary, 1996.