On representing, linking and distributing food metadata: a framework for distributing descriptive data about food

Final Year Project (2005)

Author: Thomas Betts Supervised by: Vladimiro Sassone

Computer Science and Artificial Intelligence,
Department of Informatics,
University of Sussex

Candidate number: 41139

Statement of originality

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Acknowledgements

I would like to extend sincere thanks to Vladimiro Sassone for his guidance, support and enthusiasm for this project.

Abstract

> **'Over 90% of consumers want to know where the meat in their pork pie or sausages comes from'** - Cabinet Office, 2002 [37]

This report outlines a centralised and self-moderated database system for the distribution of descriptive information about food. Designed as a server, the product combines principles of web services with the Semantic Web to provide a facility for the accurate, machine-processable and extensible description of food. To facilitate the population of such a vast shared database, an open-access approach is proposed for both reading and writing.

While applications are suggested, the database is constructed with no specific application in mind and no usage restrictions. The type of data to be stored in the database and the use of that data is a matter for software developers using the database. However, disparate use of the system is encouraged to provide a broad basis of information.

The report focuses on four key areas:

- the role of users in a shared database system, including discussions on authorisation, accountability and participation.
- the representation of food item description using a serialisable and extensible vocabulary language
- the management of data including methods to maintain data integrity and reduce vandalism and data error in an open-access database
- the interface to the database as a web service

Contents

1 Introduction **9**

1.1 Motivation 9

1.1.1 Applications 10

1.1.2 Terms of reference 10

1.2 Relevance 11

1.3 Approach 11

1.4 Structure of this document 12

2 Professional considerations **14**

3 Requirements analysis and problem specification **15**

3.1 Problem specification 15

3.2 Existing systems 15

3.2.1 Systems with similar domain-specific aims 16

3.2.2 Systems with similar technical aims 16

3.3 Users and participation 18

3.3.1 Capturing the data 18

3.3.2 Participants 18

3.4 Data representation 20

3.4.1 Metadata model 20

3.4.2 Choosing a data model 21
3.5 Data management 23
3.5.1 Data integrity 23
3.5.2 Storage 24
3.6 Delivery mechanism 25
3.6.1 Web Services 25
3.6.2 DOAFI and Web Services 25
3.7 Testing 26
3.8 System overview 26

4 Design **27**

4.1 Users and participation 27
4.1.1 Authentication 27
4.1.2 Authorisation and integrity 27
4.1.3 Storing user details 28
4.2 Data representation 28
4.2.1 Relational modeling 28
4.2.2 Prototyping the standard vocabulary 28
4.3 Data management 31
4.3.1 Uniform Resource Identifiers 31
4.3.2 Rolling back changes 32
4.3.3 Transactions 33
4.4 Delivery mechanism 33
4.4.1 Query protocol 34
4.4.2 Server administration module 36
4.5 Test specification 36
4.5.1 Server 36

4.5.2 Vocabulary 37
4.6 System overview 38
4.6.1 REST 38
4.6.2 Components in place 38

5 Implementation **40**
5.1 System structure 40
5.2 Users and participation 40
5.3 Data representation 41
5.3.1 Query structure 41
5.3.2 Query result 43
5.3.3 Standard vocabulary 44
5.3.4 Recording changes 45
5.4 Data management 45
5.4.1 Rollback implementation 45
5.4.2 Transactions 46
5.5 Delivery mechanism 46
5.5.1 Server configuration 46
5.5.2 Database 47
5.5.3 Receiving queries 47
5.5.4 Interpreting queries 47
5.5.5 Administration module 47

6 Testing **49**
6.1 Server 49
6.1.1 Execution of regular operations 49
6.1.2 Concurrency 51

6.1.3 User access 52
6.2 Vocabulary 53

7 Conclusions **54**

7.1 Project evaluation 54
7.2 System criticism 55
7.2.1 Data representation and semantics 55
7.2.2 Users and access 56
7.2.3 Efficiency 56
7.3 Deployment requirements 57
7.3.1 System integration 57
7.3.2 Users and contribution 57
7.4 Extensions and future work 57
7.4.1 Accuracy 58
7.4.2 Collaborative categorisation 58
7.4.3 Social software 59
7.4.4 Identity and trust 59

Bibliography **64**

Appendices **64**

A Report appendices **65**

A.1 Requirements Analysis 65
A.2 Design 67
A.2.1 RDF instantiation 67
A.2.2 Formalising the vocabularies with RDFS 70
A.3 Implementation 77

A.3.1 Code walkthrough 77
A.4 Testing 80
A.4.1 GetQuery 80
A.4.2 SetQuery 80
A.4.3 User access 81

B Source code **86**

B.1 DOAFI package 86
B.1.1 QueryProcessor 86
B.1.2 QueryBase 90
B.1.3 QueryInterface 92
B.1.4 GetQuery 92
B.1.5 SetQuery 95
B.1.6 QueryWrapper 107
B.2 Vocabulary package 108
B.2.1 DOAFI 108
B.2.2 QUAFI 111
B.3 Administration module package 112
B.3.1 AddUser 112
B.3.2 EditUser 114
B.3.3 DeleteUser 116

C Administration **118**

C.1 Project schedule 118
C.2 Project phases 119
C.3 Progress report 121

List of Figures

3.1 Example assertions made by different organisations or food certifying bodies 20

3.2 Tree structure of classes as found in a taxonomy 22

3.3 A visual overview of the proposed DOAFI system structure . . . 26

4.1 Worcestershire Sauce RDF/XML containing base classes 30

4.2 A RDF/XML Biography class 30

4.3 Simplified graph-based view of difference between original database and edit 32

4.4 Sequence of database item edits 33

4.5 Database query example combining the DOAFI and QUAFI vocabularies 35

4.6 Database response example combining the DOAFI and QUAFI vocabularies 36

4.7 Component structure of the DOAFI server 39

5.1 An example SetQuery involving modification to title field of a FoodItem 41

5.2 GetQuery request to a depth of 2 42

5.3 GetQuery response to a depth of 2 43

5.4 Graph representation of a server response 44

6.1 Testing concurrent requests 52

6.2 Server response to failed authentication 52

6.3 Example data model illustrating combined vocabularies 53

A.1 DOAFI server class diagram 78

A.2 Example of a rollback query 80

A.3 Response to query by properties 81

A.4 Response to query by URI 81

A.5 Response to query for non-existent item 82

A.6 SetQuery operation to delete a property 82

A.7 SetQuery operation to rollback changes to the data model 83

A.8 The FoodItem modelHistory after compound changes 84

A.9 Base example of authentication query 85

List of Tables

4.1 Domain elements and their properties 29

4.2 Query vocabulary classes 34

4.3 Query vocabulary properties 35

4.4 Server query execution test specification 37

5.1 Misinterpretation of a historical version of an item 45

6.1 Results of execution of GetQuery operations 50

6.2 Results of execution of SetQuery operations 51

Chapter 1

Introduction

A shared database of food metadata could transform our ability to make justified decisions about food purchases. Description of a Food Item (DOAFI) aims to achieve exactly this - providing a centralised and self-moderated database of machine understandable food description. A generalised framework and web service interface allows any party to read and write data using standard or proprietary vocabularies. The ability to generate a vocabulary for describing a specific interest is open to all.

1.1 Motivation

> **'Over 90% of consumers want to know where the meat in their pork pie or sausages comes from' - Cabinet Office, 2002 [37]**

Traceability of ingredients and food production processes are fundamental to food quality, but this information is missing from current food labeling due to the physical constraints of providing such data. In this project I propose and construct an open-access database server with a web service interface for the collection and distribution of food metadata. This facilitates the recording of comprehensive and extensible description by corporations, individuals and independent and disparate organisations about a food item and each of it's constituent ingredients. The aim is to encourage the use of the database in software development and ultimately increase both producer and consumer participation for reading and contributing.

At present, EU law dictates that it is not obligatory for manufacturers to list the ingredients of compound ingredients if they form less than 25% of the total product weight [28].Various reasons are cited for this including lack of space and ease of description. Furthermore, where ingredients are listed, they are usually non-specific. For example, one might see 'beef mince' in an ingredients list -

but what part of the cow was used to produce this product, who produced it and where was it produced? There is little reason why this information cannot be provided - if the consumer demands it. It is likely that somebody, somewhere has the knowledge to describe the ingredients or production processes, whether they are the producer, a wholesaler, an independent organisation or an individual. Rheingold [39] suggests that using an open-access, shared database has the possibility to 'fundamentally transform knowledge-sharing by drastically lowering the transaction cost of matching questions and answers'. DOAFI may indeed open up a potentially rich (in volume and semantics) knowledge-base to developers to use freely in their applications. I indicate a variety of feasible applications in the following subsection.

The availability of comprehensive food item description, especially when augmented with guidance or knowledge from food authorities, would remove the market distortions we currently experience as a consequence of inconsistent and inadequate food labeling. The use of DOAFI could enable both human and automated product comparison, involving a better heuristic than price. Independent food bodies could describe accurately, in terms of manufacturing processes, ingredients or economic impact, the differences between a 'regular' product and one which is free-range, organic or fair trade. This combination of information could inform users (consumers or producers sourcing ingredients), guiding them to make decisions which they are unable to make with currently available information, based upon their own interests, ethics and priorities.

DOAFI attempts to provide a real basis for comparing and profiling food.

1.1.1 Applications

The possibility for applications which implement the DOAFI knowledge-base are unbounded: from food 'recommendation' systems which infer dietary preference (for example, you may prefer locally produced food, items comprising organic ingredients, or animals which have been fed a specific diet) to the automated profiling of nutrients and vitamins.

Use of the database could be coupled with hardware or software systems, for example a handheld device helping a consumer to make decisions, based on their preferences, when purchasing food in a shop. It could be combined with Internet food shopping facilities for the same purpose. The data could also be used in a system which compares recipes for freshly prepared food with the corresponding processed equivalent. These systems could use expert systems and reasoning in order to make recommendations or infer likely answers to consumer problems.

1.1.2 Terms of reference

The terms of reference for this project were to:

1. Define the markup required to describe food and design a metadata model capable of expressing food item relationships.
2. Determine a common language and a protocol for client/server interaction.
3. Determine a method for serialising the data using XML and create an open access interface using a web services approach.
4. Implement some persistent storage device for the data, relationships and attributes.
5. Provide mechanisms to reduce data duplication, inaccuracies and abuse.
6. Produce sample instantiations using the metadata model.
7. Develop a client for querying the server for the purpose of illustration.
8. Discover external reaction to the system from independent bodies in order to evaluate the system.

An implementation of the system should facilitate access to food item information, for example: the producer, production methods and ingredients - each of which could be further described by database entries and specialised vocabularies. Allowing open access to the data as a web service facilitates ubiquity and interoperability between clients and systems.

1.2 Relevance

This project is concerned with creating a framework which will allow the unified storage and delivery of descriptive data about food. I consider this project to be Semantic Web [34] enabling - providing machine understandable meaning to food description, and a process for collection and delivery of this data.

The creation of such a knowledge base has potential use in the areas of logic programming, knowledge representation and machine learning, as well as more general applications involving a public, shared database. During the course of this project I have drawn influence from database theory and application, distributed systems, standards for information interchange, internet technologies including web services, protocol design and XML and general principles from software engineering.

1.3 Approach

In this project, I combine technologies, theories and concepts that I have previously studied independently in order to solve a real-world problem, to which there is not yet a well-known solution. Due to the open-endedness of the specification and the relatively underdeveloped vision for the Semantic Web, I opted

to develop the primary themes of the project using an iterative and incremental approach.

This development approach involved development in 2 phases, requiring complete cycles of requirements analysis, design, implementation and evaluation. The first phase involved developing a requirements analysis for the project, followed by the design and implementation of a subset of the project requirements. During the second phase, revisions were made based on learning from developing and evaluating the first phase. The remaining subset of requirements were also implemented during the second phase.

Development could continue to follow this format after an initial deployment.

Further detail of the content of each project phase is contained in Appendix C. A Gannt chart for the project illustrating estimated and actual completion dates for the project phases is included in Appendix C.3.

1.4 Structure of this document

Throughout this report I focus on 4 primary themes for the development of the project - users and participation, data representation, data management and delivery of the data. Each of these themes are addressed and developed in the requirements analysis, design and implementation sections.

Chapter 2 briefly addresses ethical and professional considerations with respect to the storage and dissemination of DOAFI data. I discuss the self-moderated and open-access nature of the database in relation to libel, copyright infringement and potential economic impact.

Chapter 3 includes a requirements analysis and problem specification. I evaluate similar existing systems and consider the needs and role of users in the success of a shared database. I explore the requirements for metadata storage and assess technologies for representing data. I begin to present ideas about data integrity, including a hierarchy for database keys and facilities for storage. Finally, I consider the interface to the system as a web service.

Chapter 4 explains the translation of the requirements analysis into a project design. I formalise and illustrate the evolution of the vocabularies for food description and server interaction. I determine a policy for access control and authentication and discuss data accuracy and integrity in terms of keys, database transactions and the ability to rollback changes. I also explain the design for a delivery mechanism and give an overview of the required server components.

Chapter 5 details the implementation of the server and vocabulary elements of the project. I provide a walk-through of the server source code, and detail the implementation of transactions and database rollbacks. I explain the interaction between the components of the server and describe it in terms of a delivery mechanism for a web service. I also explain the representation of queries, re-

sponses and database entries in RDF/XML and describe user interaction, in particular in relation to authentication.

Chapter 6 illustrates the testing which has been undertaken on the system and vocabularies. It includes execution of 'regular' operations, and specific testing with respect to user access and concurrent database access. I also discuss the extent to which the vocabulary meets the system requirements.

Chapter 7 provides closing remarks and a critical review of the project implementation. I evaluate the project according to the terms of reference, provide criticism of the implementation and detail conditions to be met prior to deployment of the server. I also outline areas of future work to improve or build on the work of this project.

Chapter 2

Professional considerations

As with any shared database, to be delivered over the internet and with no explicit moderation in place, one should be aware of the problems which accompany that. Data may be entered into the system which are factually incorrect, opinionated, could be libelous or subject to copyright or similar restrictions. For a maintainer of a database, at best these problems cause inaccuracies in the data, at worst they break the law.

I have therefore concentrated effort in the combined areas of data integrity and user authorisation to minimise the effect of information imperfections. The nature of structured metadata and the reduction of free-text description should further lead the description of food items to be fact-based.

It is important that it is made clear to any users of the system that no guarantees are provided about the quality or accuracy of the data. Where producers are entering their own data into the system, they should be aware that they are describing their product and any misinformation provided could be covered by the relevant food labeling legislation.

A successful implementation of a system such as this, if widely used, could alter food consumption behaviour. By facilitating access to perfect or near perfect information about food would allow direct comparison of products on a like-for-like basis, allowing consumers and producers to evaluate products more accurately and with a better heuristic than price alone. At present there is no way for consumers to access such information, and as such the effect that it could have on purchasing patterns is therefore unknown.

Chapter 3

Requirements analysis and problem specification

In this chapter I provide a problem specification and an analysis of the area. In the first section I quantify the problem. In the second section I evaluate existing systems. I consider the role of users in the success of a shared database in the third section. In the fourth section I evaluate XML-based solutions to document structure. In the fifth section I outline ideas about data management, specifically data integrity. I discuss web services and delivery in the sixth section. The chapter ends with brief comments on structure and testing.

3.1 Problem specification

The requirement is to construct a shared database for the storage of food metadata. A vocabulary should describe characteristic and physical properties of a food item for storage in the database, and the relationship between food items. Assumptions should not be made about the structure of the database, such that users will be free to combine their own vocabularies with those provided.

The database is to be used for the distribution of food description and should therefore support some method of collecting and delivering that data. A web service interface is to be provided for this purpose.

3.2 Existing systems

In this section I examine existing solutions to similar problems. In the first subsection I analyse domain-specific applications for data delivery. In the second subsection I assess shared database solutions.

3.2.1 Systems with similar domain-specific aims

Linking Environment and Farming (LEAF) is a charitable organisation aimed at bringing farmers and consumers together. Their service, **LEAF Tracks** [29], allows consumers to find out who produced their food. This is achieved by a lookup of a producer number (attached to the product with a sticker), using a web-based interface. The scheme has been supported by some major UK supermarket chains.

Specific limitations of LEAF Tracks include:

- Provision of information is restricted to generic producer details
- Intended for a human audience only: the majority of the data is supplied in prose - rather than structured metadata.
- Content delivery is via a web interface and is not specifically machine readable.
- The system does not offer any facility for the description of food products comprising more than 1 ingredient, it is concerned only with farmed produce.
- The sole shared aspect to the database design is the ability for any user to read. Database modification is reserved for LEAF accredited producers, leading to a limited system applicability.

Increasing availability of products using the LEAF Tracks system illustrates the demand for greater access to information about food production. However, it supplies a restricted subset of the information proposed by DOAFI.

I have been unable to locate any other consumer-oriented systems for the dissemination of descriptive data about food.

3.2.2 Systems with similar technical aims

Music databases

One of the most frequently referenced shared databases syndicated over the internet is **CDDB** [21], a centralised database storing CD track titles, the progenitor of the various audio CD metadata projects.

Audio playing software can be enabled, via the use of an API, to use the database to determine CD track titles. An important feature of CDDB is that the database is populated by users. If a user plays a CD not present in the database, rather than return the titles, the API will ask for the user to enter them. Subsequent requests for the metadata for that particular CD will then be successful.

As a consequence of the sale of CDDB to Gracenote in 2001 and perceived and actual removal of 'freedom' from the system, a number of further projects materialised. One such project is **FreeDB** [19], effectively a clone of CDDB, with an additional interface operating over HTTP to avoid firewall restrictions encountered using CDDB (which uses port 8880 for communication).

MusicBrainz [35] expanded the CDDB model, creating a framework for a music encyclopedia. Starting with the CD / track database, further information can be associated such as artists, CDs, tracks, publishers, performers and record labels.

This less-simplistic metadata model is described using the Resource Description Framework (RDF), which I discuss further in Section 3.4.2. XML is employed for serialising the data, meaning both the syntax and semantics for information interchange are common, open standards. Query and response involves posting RDF/XML using HTTP to a query processor. Developers who are not concerned with the detail of RDF may use an API to wrap and unwrap their queries.

Wikipedia

Wikipedia [48], 'the free encyclopedia', is currently one of the most popular open access, shared database projects. It differs somewhat from the proposed DOAFI model as it does not take a web services approach and editing is achieved via a manual web interface. Little metadata is stored against each article - with the main bulk of the information content being prose - Wikipedia is aimed very much at a human audience.

The project has frequently received criticism [45] about the likelihood that published information will be inaccurate, biased, of poor quality and subject to vandalism due to the extreme open access 'anybody can edit' policy. Many people assert that the project cannot scale, the belief being that as it increases in popularity, so should the level of misuse of the resource. To date, this has not been the case. According to Alexa [1], Wikipedia is the 170th most visited web site on the Internet, and for the most part, the content remains factually reliable. Those analysing Wikipedia suggest that as the site increased in popularity, so did the number of participants willing to oppose abuse.

Empirical analysis of these shared databases suggests that systems with open-access writing policies *can* scale, and abuses will be ironed out by genuine participants in the long term. Bricklin [12] describes this phenomena as the 'cornucopia of the commons', suggesting that contrary to being detrimental to the system, 'use brings overflowing abundance'.

3.3 Users and participation

In this section I examine the role of users in populating a shared database and user needs. In the first subsection I detail successful shared databases and users. In the second subsection I indicated participant types and their potential roles.

3.3.1 Capturing the data

Traditionally a large database would be populated using an organised manual approach, where authorised bodies are specifically organised to enter a large quantity of data. A preferable alternative is to automate the process - an organised mechanical approach, such as an automatic indexing algorithm.

The popularity of the Internet as a communication medium has given rise to another method whereby users volunteer to manually enter entries themselves. This option is useful for populating exceptionally large databases, where even a substantial organisation does not have the resources to enter the data, and where an automated system is not viable. The volunteer manual approach could allow a database to potentially be filled with semantically rich data, very fast; if a motivation for users to enter their own data can be found. This is where 'CDDB succeeded ... by harnessing the energy of its users' [13]; the motivation being that the users wanted to see track information about CDs in their collection when playing them. They participated for their own benefit, but simultaneously benefitted a community of users.

3.3.2 Participants

The scope of this project is to design and deploy a data store and facilitate access to that store at a low level. Ultimately access to and use of the data is aimed at end users, however the interface that I am concerned with is for software developers in the first instance, with consideration as to their likely use of the data. In the following sections I discuss participant interaction, a detailed account of user requirements is contained in Appendix A.1.

Producers

Producers are in the best position to collect and administer (and mechanise the collection of) their data; large producers are most likely to have the infrastructure to be able to do this. When looking at the creation of metadata for electronic documents, Greenberg et al. [22] found that creators are intimate with their work, they are familiar with their audience and are likely to want their work to be consulted. Their study found that the original creators, while not expert in the field library science, were able to produce very high quality

metadata. This suggests that food producers would be equally suited to the task of metadata generation.

The question remains as to why a producer would want to contribute to a shared database. Thomas and Griffin [23] suggest that where the burden of metadata creation falls upon producers, a financial incentive is required to encourage them to participate. It may also be the case that food producers want to actively disguise exactly the sort of factual information DOAFI aims to reveal, in which case, presumably an even greater financial incentive is required to secure their participation.

Responsibility for metadata creation should therefore be delegated elsewhere during the infancy of the system. In the first instance, it is possible that smaller producers may be willing to participate. While this administrative overhead may be large as a proportion of their workload, many such companies may consider it in their interest to participate in the system. Their participation could provide a basis for their products to be accurately compared to the competition.

Developers

Developers are the bridge between DOAFI and the end users. They will not be directly involved in contributing to the database, but will dictate the interactions between end users and the database, and will be responsible for making use of the data.

End users

It seems likely that individual contributors will be instrumental in populating the database, in addition to producers, especially during the infancy of the project. They may fill in gaps left by producers or supplement existing information. While contributions may only be beneficial as a group, as in the cornucopia of the commons [12], contributors will want to see an individual reward for entering data into the system. Such a reward could be the ability to use 'their' data in some DOAFI-enabled application.

I also include organisational bodies, not involved in the production of food, in this category. Figure 3.1 illustrates an informal model of plausible assertions which could be made by organisations about farmed fish. Assertions about food do not necessarily reflect complimentary interests and the incentive for food bodies to participate is to allow their assertions to form part of the model for a specific food item.

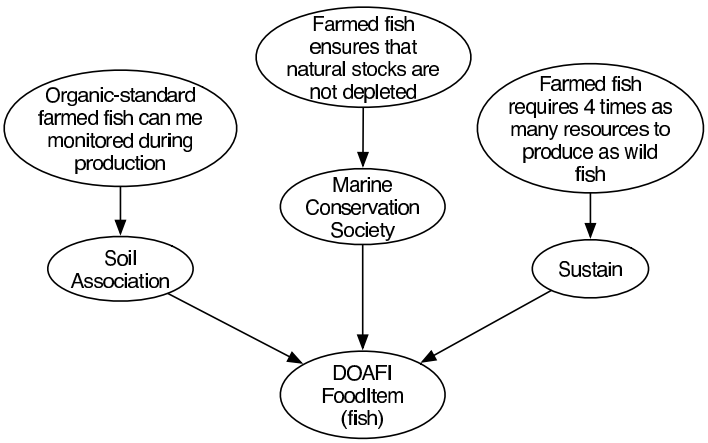

Figure 3.1: Example (fictitious) assertions made by different organisations or food certifying bodies about farmed fish

3.4 Data representation

In this section I discuss ontologies, taxonomies and their integration in XML. I provide an introduction in the first subsection. In the second subsection I compare and contrast DTDs, XML Schema and RDF as viable solutions.

3.4.1 Metadata model

It is necessary to create a universal and extensible domain-specific ontology for food item description. Such an ontology must allow the combination of simple, factual attributes for example weight, product name and producer with more complex (and perhaps subjective) properties, such as the assertions in Figure 3.1.

Simple attributes should not allow for duplication, however there may be multiple properties, such as certificates held from food organisations or conflicting assertions about a food item. In particular, the semantics of these properties should be broad such that they can be independently formalised by domain specialists. Such formalisation involves combination with other ontologies. Presently available relevant ontologies include the Food & Agriculture Organisation of the United Nations: Agricultural Ontology Service Project [44] and ontologies dealing with other factors such as geospacial data [10].

3.4.2 Choosing a data model

It is necessary to determine some model or framework for the description, storage and serialisation of the relevant data.

As a well recognised, widely used and supported method for data interchange on the Internet, XML should be used in some form for serialising the data. However, it is not clear how the data should be structured, and how structure should be imposed on the model used for describing and relating data.

Possible options are discussed in the following sections and include DTD and XML Schema, or the more radical approach of RDF serialised using RDF/XML and formalised using RDF Schema.

DTD

Document Type Declarations (DTDs) were included in the first release of XML [9] to apply structure to an XML document. This includes dictating each allowable element in the document, possible attributes and values. DTDs are also concerned with the occurrence and nesting of elements and are widely used for the syntactic validation of XML.

DTDs are widely supported, due to early adoption and being relatively intuitive for human readers to understand. Criticisms of DTDs include that they are not based on XML syntax and so cannot be processed by the same processing engine used for XML documents. Additionally, they do not fully support namespaces and hence make the combination of vocabularies difficult.

More fundamentally, DTDs are only capable of defining how vocabulary items relate to one another at an element level in an XML document. This model is concerned only with XML document structure and not necessarily with the more general sense of how objects relate in the real world.

XML Schema

A more sophisticated solution to the validation of XML documents is defined by XML Schema [6, 8]. These are more closely related to the definition of relational tables and the 'class' structure of object-oriented design. XML Schemas are themselves defined using XML, and are therefore much more easily machine readable. Similar to DTDs, they define how elements relate to each other, however they provide functionality to record data types for elements and attributes, rather than just 'character data' as supported by DTDs.

XML documents, as defined by DTDs or XML Schema, form tree structures at the element level, as illustrated in Figure 3.2. Using XML documents for DOAFI would be suitable for the description of items which can be described using a taxonomy - where elements form a hierarchy.

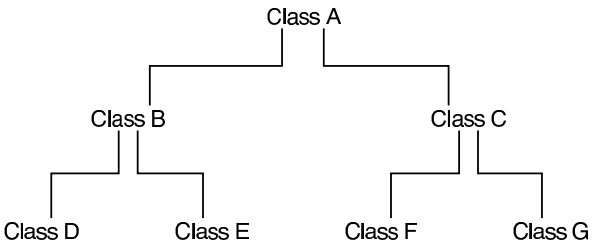

Figure 3.2: Tree structure of classes as found in a taxonomy

XML Schemas support namespaces and by using multiple namespaces in an XML document, one can combine elements from different XML Schemas, leading to the possibility of combing vocabularies.

RDF and RDF Schema

The Resource Description Framework (RDF) [5] is a W3C specification for a metadata model which does not explicitly and necessarily make use of XML, and their recommendation for the Semantic Web [34]. It can be serialised using a dialect of XML, specifically RDF/XML. The model is based around the idea of making statements about objects in the real world. These are in the form subject (the resource or object being described), predicate (the aspect or characteristic being described), object (the value of that characteristic). This is known as the RDF 'triple' – everything being described by RDF is in this triple form.

However, RDF alone provides no mechanism for defining properties such as relationships between resources or attributes of resources. RDF Schema [11], the vocabulary description language, makes it possible to describe classes of data and their properties. RDF Schemas are themselves written in RDF. As all RDF documents can be serialised using RDF/XML, it is possible to create RDF Schemas in well-formed XML syntax, for flexibility and compatibility.

RDF Schema and the RDF model offer most of the benefits of XML Schema and XML more generally. While RDF/XML is well-formed XML, the semantics of RDF/XML and non-RDF XML are considerably different. XML documents naturally form a tree structure, whereas in RDF everything is expressed in the form subject-predicate-object - each triple could visually be viewed as a node-arc-node graph representation. The potential use of URIs for every element in an RDF model breaks the forced hierarchy of XML - each element in the model can be uniquely identified by a URI and the relationship of elements is no longer dictated by their ordering in a tree. The graphical representation of aggregated RDF triples is a directed labeled graph.

XML and RDF compared

A detailed discussion of XML and RDF as suitable technologies for DOAFI is contained in Appendix A.1. The choice between 'regular' XML or RDF comes down to the underlying structure of data being described. If that data can be described entirely using a taxonomy, then an approach using XML Schema or DTD would be appropriate. However, using such a hierarchical structure makes it difficult to describe relationships such as 'part of', which will be important when describing food items, for example when expressing a food item as an ingredient of another.

It is straightforward to represent hierarchical structures and more complex relationships using directed labeled graphs, and hence RDF.

3.5 Data management

In this section I discuss keys and uniqueness for data integrity and database storage. The first subsection details mechanisms for maintaining data integrity in an open-access database. The second subsection introduces storage in RDF.

3.5.1 Data integrity

Shared resources which can be freely modified will inevitably suffer from abuse or misuse. Within a CDDB-like system, misuse is limited to incorrectly modifying track titles; easily reverted by the active members of the community.

Possible reasons for lack of accuracy in database entries may include disagreement on item content, intentional vandalism or the inclusion of factually incorrect information.

Uniqueness

CDDB benefits from the ability to create almost-unique keys for database entries using a hashing algorithm on the actual structure of the CD. This helps protect against data duplication, however in the case of food, it is not possible to automatically generate some unique and derivable key.

Some other mechanism is required to reduce duplication. Possible ideas include:

- *Self-moderation via the input process.* When a user enters a new record, before storing it, a number of similar records could be returned, requesting the user to verify the uniqueness of their entry. A data mining algorithm such as k-nearest neighbours [43] could be used to determine this similarity,

based upon fixed, physical food properties. This may not be a trivial problem, as the system would need to deal with incomplete data and encoding issues.

- *Allow anyone to enter anything.* Wikipedia does exactly this. The community of Wikipedia users also police it; inappropriate, incorrect, badly structured or duplicate material is quickly rolled back, removed or merged. The system relies on volunteer labour, rather than computation.
- *Restrict the creation of entries to trusted and authenticated users.* This would add an element of 'domain specialism', for example food producers could be responsible for manually entering their own produce and they would be directly responsible for avoiding duplication. Users should still be able to modify the database with further descriptive data. This would necessarily divide each entry in 2, a base entry (supplied by the trusted party) with optional additional information.

In this version of DOAFI, I propose the use of the 2nd option, augmented with some authentication. Self-moderating behaviour could be added as a module at a later date.

Open access

Based on the discussion in the previous sections, and especially the work of Thomas and Griffin [23] - who suggest that producers will not participate until they experience a financial incentive (ie the system is popular and participation is a marketing exercise), I propose that the database be open to modification and access by anybody. If producers add their produce and include factual properties about the item, users should be able to add to these, and even modify them.

It would be useful to attach some user information against every modification made, to facilitate user accountability, with the aim of reducing abuse (for example using account blocking) and ultimately increasing data accuracy. This might also create an opportunity for applications to make use of a 'web of trust' type approach to decide which database information to believe. This requires user authentication.

Reading should not require authentication.

3.5.2 Storage

The data must be centrally stored in an efficient manner; that is efficient with respect to the time taken to query the data store, the memory required to perform a lookup, and the overall storage required. RDF data can be stored using a regular relational database using an RDF processing package, or via a custom written database.

Guha's rdfDB [25] is a custom database written in Perl, however it is no longer being developed and is not widely supported. Jena [27] and Sesame [41] are Java-based tools which allow processing and storing of RDF data in a relational database. Both are actively being developed and are suitable for the development of a servlet for the delivery mechanism, as detailed in the next section.

3.6 Delivery mechanism

In this section I discuss the database interface as a web service. Initially I introduce web services and follow this by combining it with DOAFI.

3.6.1 Web Services

As with all web services, delivery should be over HTTP using port 80. The ubiquity of HTTP for delivery means that it is unlikely to be blocked and is platform independent.

Traditionally, web service requests and responses are wrapped in Simple Object Access Protocol (SOAP) [24] packets. These enable the exchange of messages between applications, primarily over a network. Use of SOAP requires knowledge of request and response message formats. This can be communicated using the Web Services Description Language (WDSL) [14], an XML-based mark-up describing web service communication.

3.6.2 DOAFI and Web Services

It is necessary to define a language for communication between client and server. For purposes of consistency, I propose the use of RDF as a query language as well as data model. This will ensure that queries are native to the database content and will allow queries to easily be made on partial data.

Due to RDF being a relatively young standard, there is not yet a well-known method to best deliver RDF data or integrate it as a web service. It is possible to wrap RDF into a SOAP packet, although this use is questionable [36]. MusicBrainz [31] has opted not to use SOAP, and are processing and delivering RDF/XML alone, separating the details of the query using XML namespaces.

RDF Schema can facilitate the automatic discovery as to how a data model can be queried (based upon the semantics of the data), and this WSDL-like property is cited as one of the reasons for RDF being so popular amongst the W3C.

I therefore propose to simply receive queries in RDF/XML using HTTP post, and return a response, also in RDF/XML, in the HTTP packet. RDF Schemas will be made available.

3.7 Testing

Instantiating the RDF Schemas, and attempting to add, edit and view data from the system will be a fundamental test for the system. Tests should be applied to evaluate the performance of methods implemented to avoid duplicate or bogus data in the system, and to test the support for users.

Testing will be continuous and will cover both the system and vocabularies.

3.8 System overview

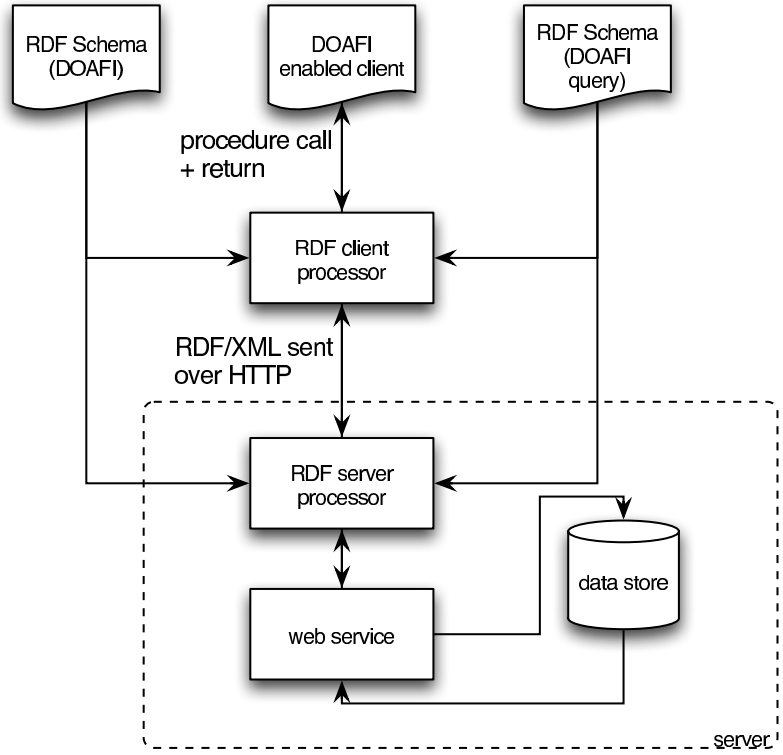

Figure 3.3: A visual overview of the proposed DOAFI system structure

The structure of the system (as pictured in Figure 3.3) is quite elementary, using a simple client-sever model. The client is interchangeable and only requires the ability to process RDF. Both the client and server share a common vocabulary whose semantics are defined by 1 or more RDF Schema. The server is responsible for processing queries and executing the features of the web service, including database interaction and generating responses.

Chapter 4

Design

In this chapter I develop the requirements analysis into a design, at the level of concepts and modules to implement. In the first section I discuss the implementation of authorisation of users. In the second subsection I use relational modeling to design the vocabularies. I discuss tools for the reduction of database error including URIs, rollbacks and transactions in the third section. In the fourth section I outline the query vocabulary as a protocol for communication. The chapter ends with some detail about testing and an overview of the server structure.

4.1 Users and participation

In this section I discuss the implementation of user authentication as discussed in Section 3.3. This includes subsections on authentication, authorisation and implementation detail.

4.1.1 Authentication

There are to be no restrictions placed on reading from the database; no user should need to be identified for read access. In order for users to modify the database, it would be useful to bind identity to their modifications, as discussed in Section 3.5.1.

4.1.2 Authorisation and integrity

Based on the assumption that producers and 'trusted' users would make few errors when entering data, restricting database edits to only these users would

remove the need to provide a facility for automatically reverting changes. However, as stated in Section 3.5.1, it would be ideal for reading and writing to be carried out universally.

Providing identity can be verified, all users should be able to create and edit items in the database. The problem arises that inexperienced users may not know how to correctly classify items, however increased participation would allow these errors to be rectified by active participants. An example of this approach working is Wikipedia [46].

4.1.3 Storing user details

This method of authorisation requires authentication and relies on a common access list dictating the level of access. This information can be recorded as a simple database table containing a username, a password which has been subject to some hashing function (such as MD5 [40]) and an access level.

4.2 Data representation

Berners-Lee [7] cites the relational data model as a principal point of reference for the Resource Description Framework. While not commutable, these models share similarities. I therefore use relational modeling techniques to develop the RDF vocabulary in this section. In the second section I provide some brief prototype examples, a complete version is contained in Appendix A.2.1

4.2.1 Relational modeling

Crucially, the system must allow the retrieval of food item metadata. Preferably this should be implemented independently of a specific vocabulary, such that the system is not bound by a single vocabulary. However to some extent it is necessary to define domain-specific and intrinsic properties of food items to be referenced within the system.

Table 4.1 shows some loosely defined domain elements identified and their properties which form the basis of the vocabulary.

4.2.2 Prototyping the standard vocabulary

For the purpose of this and subsequent prototyping and discussion, I will use Lea & Perrins Worcestershire Sauce as a sample food item. A complete model for this product is contained in Appendix A.2.1. An introduction to the model is included in Figures 4.1 and 4.2.

Class	Property	Description
FoodItem	Unique food ID	to idenfity a FoodItem
	Title	full name
	Ingredients	list of ingredients
	Biography	biographical information about FoodItem
	Produced	details of production methods and process
	Certification	properties asserted about the food item
	History	stores Movement and Model changes
	Alternatives	list of alternatives for this product
Ingredient	Amount	how much of the ingredient is in FoodItem
	Amount unit	measurement of amount
	Ingredient item	URI of ingredient
	Reason	reason for inclusion in FoodItem
Biography	Use before	consumption best before date
	Amount	how much of the item there is
	Amount unit	unit of measurement of the amount
Produced	Producer	URI of Producer
	Produced timestamp	date of production
	Geographic	location of this production
	Process	how item was made at this stage
	Used items	list non-food items items used in production
	Address	full address of producer
Producer	Producer ID	unique identifier for a Producer
	Title	full name of producer
	Web address	primary web address
	Geographic	physical location of producer, eg. factory
Used Item	Used item	URI of the item being used
	Amount	amount of item used
	Amount unit	unit of amount
	Used timestamp	date of use
	Reason	reason for use
Certification	Type	URI type of certificate issued
	Certifying body	certificate issuer
	Reason	explanation for why it is certified this way
	Timestamp	date of certification
	Expiry date	certificate valid until expiry
Certificate	Certificate ID	unique identifier for a certificate type
	Title	title of certificate
	Web address	primary web address for human readers
MovementHistory	Geographic	geo-spatial location at end of movement
	Timestamp	date of movement
	Reason	reason for movement
ModelHistory	Changes	differences in model caused by change
	Timestamp	time and date of change
	User	user responsible for change
	Reason	string literal or reference to comment
Alternative	Alternative item	URI of alternative FoodItem
	Reason	reason for relationship between the items

Table 4.1: Domain elements and their properties

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
  xml:base="http://doafi.4angle.com/FoodItem/"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">

  <doafi:FoodItem rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml/">
    <doafi:title>Worcestershire Sauce</doafi:title>

    <doafi:bio rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
    <doafi:produced rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced" />
    <doafi:history rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/History" />
    <doafi:alternatives rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Alternatives" />
    <doafi:ingredientList rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Ingredients" />
    <doafi:certification rdf:resource=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Certification" />
  </doafi:FoodItem>
</rdf:RDF>
```

Figure 4.1: Worcestershire Sauce RDF/XML containing base classes. The model provides references to each class of additional information.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
  xml:base="http://doafi.4angle.com/FoodItem/"
  xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">

  <doafi:Biography rdf:about=
      "Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
    <doafi:useBefore>2005-03-01T00:00:00-00:00</doafi:useBefore>
    <doafi:amount>150</doafi:amount>
    <doafi:amountUnit>ml</doafi:amountUnit>
  </doafi:Biography>
</rdf:RDF>
```

Figure 4.2: A RDF/XML `Biography` class, as referenced by Figure 4.1

I provide a formalisation of the vocabulary using RDF Schema in A.2.1.

4.3 Data management

In this section I discuss features for the reduction of data error. This includes an overview of URIs as database keys in the first subsection. I outline the use of rollbacks to easily revert vandalism or error in the second subsection. I also introduce the reduction of failed writes by using database transactions in the third subsection.

4.3.1 Uniform Resource Identifiers

Items within the database need to be uniquely identified according to the available classes of metadata. Using RDF, this is best done with URIs to identify the metadata classes. This will facilitate the reuse and accurate reference to statements in the model. The base of such a URI should be within some unique domain, to ensure avoid duplication. The present such base is `http://doafi.4angle.com/`

The total URI for a given item should consist of the base concatenated with the relative property being expressed.

The following proposed structure for URIs is aimed at reducing accidental duplication in the database by guiding users to categorise items according to a hierarchy. The URI design offers no protection against physical data duplication.

Objects requiring identification

- FoodItem (eg. `FoodItem/Lea_and_Perrins/Worcestershire_Sauce/150ml/`)
- NonFoodItem (eg. `NonFoodItem/Monsanto/AntibioticX/`)
- Producer (eg. `Producer/Lea_and_Perrins/`)
- Non-specific FoodItem (eg. `FoodItem/Lea_and_Perrins/Worcestershire_Sauce/` - this would be a way of describing this class of item, generally, rather than a specific instance of the object in the database)
- Certification (eg. `Certificate/HM_the_Queen/`)

Object hierarchy

Many of the classes being described form a natural hierarchy, for instance the `Ingredients` class of a FoodItem 'Worcestershire Sauce' produced by 'Lea and Perrins' could be described by the relative URI `FoodItem/Lea_and_Perrins/Worcestershire_Sauce/150ml/Ingredients`. The same is format follows for other FoodItem classes including `Biography`, `Produced`, `UsedItems`, `History`, and `Alternatives`.

Where a producer is not known for a FoodItem, perhaps when referenced in an ingredients list, it can be referenced using a 'Generic' producer, hence a URI such as `FoodItem/Generic/Sea_Salt`.

4.3.2 Rolling back changes

As detailed in Section 3.5.1, accuracy of entries in a shared database is non-trivial, especially in the situation that anybody is able to modify or add data. I propose the implementation of a rollback system for reverting changes consisting of an operation to return the database to some previous state. This is addressed in the following subsections.

Calculating and storing changes

The Jena API provides functionality for determining the difference between 2 models. To illustrate this, consider an entry about a food product in the database, and a new entry which is a copy of the original with minor modifications. Each time a modification is made, the old entry is overwritten by the new. A function is available to calculate the difference between 2 models, and this difference should be stored after each modification. An example is pictured in Figure 4.3.

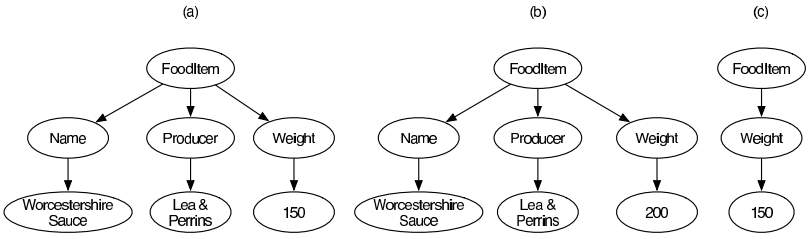

Figure 4.3: Simplified graph-based view of difference between original database and edit. State (a) shows an original model, state (b) shows the model after modification of the weight property, and state (c) shows the computed difference between the 2 models

With this difference, it is possible to return to the previous state (hence reverting changes) by treating the difference as if it were a user requested modification.

Accessing and reverting changes

Each change should have the `username` of the user making the change and a `timestamp` associated with it. It would also be useful to add a further property to the query vocabulary, `editComment`, a field for users to write a brief com-

ment justifying their edit. For frequently occurring types of edit, this could be replaced by a URI reference to a class describing the edit.

It should be possible for each uniquely identifiable database entry to have modifications associated with it. Using RDF to model these changes, one could use the built-in `RDF:Seq` notation, which stores sequential data. The model differences, combined with the additional fields as above, could be stored in such a construct. Each change would be made with respect to a given FoodItem, giving rise to the `ModelHistory` class of the FoodItem `History` class.

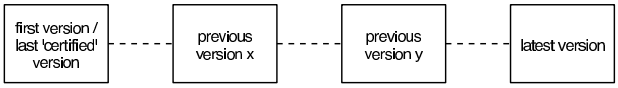

Figure 4.4: Sequence of database item edits

If Figure 4.4 were to represent the history of a database entry, then to revert the database state to x would require reverting the database first to y, then to x. Reverting in this way results in the deletion of all edits succeeding the state reverted to.

Unfortunately, should an error have been made in x which needs to be corrected, using this approach, the modifications made in version y will be removed when reverting to x. The user responsible for the rollback is free to manually re-enter any of the changes which will otherwise be lost.

4.3.3 Transactions

Transactions are a well-known solution to problems of concurrent access and failed writes in databases. In order to facilitate consistency, at the beginning of some write operation, a transaction is started. Should an element in the write fail, the transaction will not commit - all of the changes, leading to the failure, would not be stored. On success, the transaction can commit, writing the changes to the database and storing the difference.

Transactions are support natively in the Jena API.

4.4 Delivery mechanism

In this section I consider the interface to the web service. This includes a discussion, design and prototype for a query protocol in the first subsection and the administration of users via a separate module in the second subsection.

4.4.1 Query protocol

For every class of data being stored, a mechanism must be implemented to query the server with respect to that class. This includes the addition, modification and removal of metadata from the database.

As it is not possible to determine what vocabularies will be used with the system, query and response frameworks should be sufficiently general to contain all forms of food metadata. The data itself will be described by the vocabulary in use.

As queries will be posted to the server as RDF, an additional vocabulary must be established, specifically relating to the requirements of queries, which can be used in combination with the DOAFI standard vocabulary.

All queries should result in a response as per the HTTP protocol. RDF/XML responses to queries should be contained in the body of a response unless an unexpected server error has occurred, in which case regular HTTP status codes will be returned in order to facilitate error detection.

Required interactions

A query to the server will require the combination of the standard vocabulary, to describe the specific properties being requested, and a specific query vocabulary.

Queries for getting and setting data of a specific class must exist, as described in Table 4.2.

Query class	Description
`GetQuery`	reading from database query
`SetQuery`	query for adding to or editing model
`Result`	wrapper for the query response

Table 4.2: Query vocabulary classes

The result of a `GetQuery` will contain references to all of the database items matching the query, if there are any, up to a limit, `maxItems`. These references will be contained in an `RDF:Bag` and wrapped in an `ItemList`.

A `SetQuery` should be used to add, edit or delete all of part of an existing database item. The behaviour of the query with respect to adding or editing is determined by the server, for example - if the item does not exist, it should be added, or else it should be edited.

Additional properties are required in order to restrict the size of the tree of results returned, as described by Table 4.3. A facility should be provided to restrict the depth and the initial branching factor of the tree.

Query property	Description
GetQuery properties	
`depth`	a depth limit, indicating the level of information to be returned
`maxItems`	the maximum number of items (at depth 1) matching this query
SetQuery properties	
`username`	required for modifying database entries, stored against changes
`password`	a hashed password must be presented to authenticate user
`reason`	either a string literal or reference to a modification type
`deleteItems`	a flag to determine whether a set operation involves deleting the specified items or modifying/adding to the database
`rollbackTo`	facilitates the return to some previous state as indicated by the `modelHistory`
Result properties	
`status`	an indicator of whether the query was successful - a property of Result
`itemList`	URI container of items matching a query

Table 4.3: Query vocabulary properties

Prototyping the query vocabulary

As with the regular vocabulary, the query vocabulary should be prototyped. The combination of standard and query vocabularies can be accomplished using XML namespaces. Examples of a query and response are illustrated in Figures 4.5 and 4.6 respectively.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/">

  <quafi:GetQuery>
    <doafi:title>Worcestershire Sauce</doafi:title>
    <quafi:depth>1</quafi:depth>
    <quafi:maxItems>1</quafi:maxItems>
  </quafi:GetQuery>
</rdf:RDF>
```

Figure 4.5: Database query example combining the DOAFI and QUAFI vocabularies

The result should return the solution using an `rdf:Bag`. This construct is a

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/"
  xml:base="http://doafi.4angle.com/">

  <quafi:Result>
    <quafi:status>OK</quafi:status>
    <quafi:ItemList>
      <rdf:Bag>
        <rdf:li
          rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml" />
      </rdf:Bag>
    </quafi:ItemList>
  </quafi:Result>
</rdf:RDF>
```

Figure 4.6: Database response example combining the DOAFI and QUAFI vocabularies

container for holding multiple unordered properties in RDF, although in Figure 4.6 it contains only one food item. The number of items to return (at the first depth level) is explicitly stated in `maxItems`. Had a `depth` greater than 1 been specified, the server would continue to expand the model, following URIs to expand the graph until the depth limit is reached.

4.4.2 Server administration module

This servlet provides a web interface for user account administration..

Subsequent versions of the system should allow these functions to be performed using regular queries, as discussed in Section 7.2.2.

4.5 Test specification

In the following two subsections I discuss testing strategies for the server and vocabularies. The results of these tests are included in 6.

4.5.1 Server

The only public interaction with the server during the development phase is via the RDF interface. All testing will involve RDF queries, where the test results

are the responses, or if applicable, the modification to the underlying database structure.

	Test	**Anticipated result**
1	Add a FoodItem	FoodItem is added to database with associated properties Query response is 'OK'
2	Remove a FoodItem	FoodItem and associated properties are removed from database Query response is 'OK'
3	Edit a FoodItem	FoodItem entry in database is altered to reflect specified modifications Query response is 'OK'
4	Retrieve a FoodItem	FoodItem is returned to stated `depth` and `maxItems` Query response is 'OK'
5	Add a pre-existing item	Database is not modified Query response is 'Error' with a relevant message
6	Edit a non-existent item	
7	Remove a non-existent item	
8	Semantically incorrect query	
9	Syntactically incorrect query	

Table 4.4: Server query execution test specification

Queries and responses in RDF/XML are discussed in Sections 5.3.1 and 5.3.2.

4.5.2 Vocabulary

There is no general-purpose procedure for 'debugging' a vocabulary. Relevant aspects include the 'accuracy' of the domain-specific nature of the ontology, including the definition of classes and properties. RDF specific aspects relate to the graph structure underlying the data model and how this affects the access and reference to specific elements.

On designing vocabularies, Powers [38] suggests an iterative and incremental approach to testing (verification and validation) which involves prototyping and using the vocabulary as defined. This use will naturally highlight errors, omissions and optimisations.

I therefore propose no formal testing for the content of the vocabulary at this stage, but will continue to test all RDF for semantic and syntactic validity using the W3 RDF Validator [47] and the ICS-FORTH Validating RDF Parser [26]. Vocabulary modifications are discussed further in Section 5.3.3.

4.6 System overview

The area of information delivery via the Internet is well explored, and an initial aim of this project was to deliver the required information as a web service. It seems inevitable that the server should follow an approach like Fielding's Representational State Transfer (REST) [16], which loosely fits that of the web today and is discussed in the next section. The interface of the web service is to be made possible using HTTP requests (GET and POST) and RDF/XML for the description of queries and responses.

4.6.1 REST

The primary behaviour will be implemented by posting a query to the server, using RDF/XML for the accurate description of such a query. Assuming there are no server errors, it will then reply with the result of the query, also in RDF/XML.

Many queries may be straightforward, but will still require the overhead of encoding in RDF/XML, and it may be suggested that this approach is too heavyweight. In order to facilitate interoperability and compatibility, I propose that the data model and query language be unified, where possible. This was an important factor in selecting RDF over the combination of XML Schema, WSDL and UDDI for the provision of the web service, as discussed in Section 3.4.2.

It could also be possible to provide a 'short cut' facility for retrieval of simple data classes using GET. For example, if requesting the URI of a FoodItem in a web browser, such as `http://doafi.4angle.com/FoodItem/Lea_and_Perrins/Worcestershire_Sauce/150ml/`, the server could respond with a response to a predefined query, such as URIs of the components of the FoodItem.

Currently existing systems, including MusicBrainz [31], allow shortcuts such as this. While this may simplify access to the database in pre-specified ways, this may be of limited use, given that client-server interaction should be a machine to machine process. This data store is not intended to be operated on directly by human users.

4.6.2 Components in place

The web service will execute on an Apache HTTP server [3]. In order to use Java as the server programming language, Tomcat [4] will be required to augment Apache.

Jena [27] is a Java-based RDF database engine. While Jena can be queried directly over HTTP, DOAFI adds a layer of abstraction using a Java servlet which will read queries from the web server, perform the necessary database

administration via Jena, and then return the corresponding RDF response.

Jena supports a number of methods for storing RDF data. For persistent storage of RDF data it is able to use a relational database, via JDBC interface. In this case I use a MySQL database, as it is freely available, well supported and easily configurable. Figure 4.7 illustrates the proposed server components in place.

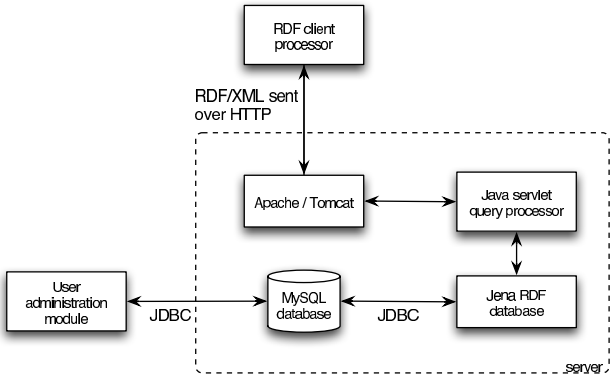

Figure 4.7: Component structure of the DOAFI server

Chapter 5

Implementation

In this chapter I explain the detail of implementing the design from the previous chapter and continue to discuss this according to the primary themes of users and participation, data representation and management and system delivery. A walkthrough of the code is provided in Appendix A.3.1 and is briefly discussed in the first section. This is followed by a section on authorisation and authentication. In the third section I explain the representation of queries and database entries in RDF/XML. Following this, I describe mechanisms to protect and manage the database content. In the final section I explain delivery of the shared database as a web service - from server configuration to user interface.

5.1 System structure

The server comprises a single package, `ds`. This is to be used in combination with a `vocabulary` package and the `admin` package for user administration. I discuss the general structure of the system in the following sections. A walkthrough and class diagram of the `ds` code is included in Appendix A.3.1.

5.2 Users and participation

As discussed in Section 4.1.2, a single level of authorisation is implemented for making modifications to the database. This level is dictated by an integer which would allow differing levels of authorisation, should they become a future requirement. Users are authenticated using a username and a password which must be provided with the query. Authentication is not required when using a `GetQuery`. An example `SetQuery` is given in Figure 5.1.

Passwords must be supplied hashed using MD5 [40] and are compared with those supplied in the 'users' table (as described in Section 5.5.2). If authentication

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/FoodItem/">

  <quafi:SetQuery
    rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml/">
    <doafi:biography
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
    <quafi:username>tom</quafi:username>
    <quafi:password>5f4dcc3b5aa765d61d8327deb882cf99</quafi:password>
    <quafi:editComment>changed the name of the product</quafi:editComment>
  </quafi:SetQuery>
  <doafi:Biography
    rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
    <doafi:title>Worcestershire Sorce</doafi:title>
  </doafi:Biography>
</rdf:RDF>
```

Figure 5.1: An example SetQuery involving modification to title field of a FoodItem

fails, the requested modification will not be made.

In this implementation, user accounts are managed using a server administration module, discussed in Section 5.5.5.

5.3 Data representation

In this section I explain the structure of RDF/XML as implemented in the query and standard vocabularies. In the first subsection I provide detail about how to structure queries. The second subsection explains the response a user would expect to receive from their query. Use of the standard vocabulary is described in the third subsection, followed by a discussion of a data inconsistency difficulty I overcame when storing the history of the metadata model in the final subsection.

Formalisation of the query and standard vocabularies using RDF Schema is provided with comments in Appendix A.2.2.

5.3.1 Query structure

Queries are structured in RDF/XML and take 2 primary formats, `SetQuery` and `GetQuery`, as illustrated in Figures 5.1 and 5.2 respectively. Both have required

and optional properties, as previously defined in Section 4.4.1.

The `depth` and `maxItems` properties of a `GetQuery` determine the amount of information returned. The query in Figure 5.2 requests a maximum of 10 items which match the desired properties (those indicated by the `doafi` namespace). Each of the returned items is explored by following URIs to the constituent parts of the model. The `depth` indicates how many levels of the graph, from the base, to explore.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi = "http://doafi.4angle.com/elements/q-0.1/">

  <quafi:GetQuery>
    <quafi:depth>2</quafi:depth>
    <quafi:maxItems>10</quafi:maxItems>
    <doafi:biography rdf:parseType="Resource">
      <doafi:title>Worcestershire Sauce</doafi:title>
      <doafi:useBefore>2005-03-01T00:00:00-00:00</doafi:useBefore>
    </doafi:biography>
  </quafi:GetQuery>
</rdf:RDF>
```

Figure 5.2: GetQuery request to a depth of 2, returning a maximum of 10 items at the first level. The request is for a FoodItem with title 'Worcestershire Sauce' and a use before date of 01 March 2005

Queries must be made in context. In the pictured `GetQuery` example, the `useBefore` and `title` properties are contained within a `biography` property. This part of the query dictates that these properties are of the class associated with the `biography` property. This context must be given to enforce the use of these properties in the model, to ensure accuracy and specifically to avoid misinterpretation of a generic property such as `title`.

Context is equally important when issuing a `SetQuery`. The URI of the FoodItem being modified in Figure 5.1 is included in the `SetQuery` statement, using `rdf:about`. This provides the context for the `biography` property, and hence the modification to the correct `Biography` class.

Scrutiny of Figures 5.1 and 5.2 reveals a difference in the way details associated with the `biography` property are included in Get and Set queries. When modifying a FoodItem, the concrete URI of a class being modified must be given - here the URI of `Biography` is given using `rdf:resource`. When executing a `GetQuery` it is not necessary to know the concrete URI of classes. In the given example, the URI of the `Biography` class is not given. This facilitates queries where concrete URIs are unknown, and the desired behaviour is to match *some* FoodItems with `biography` properties corresponding to those given.

5.3.2 Query result

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/FoodItem/">

  <quafi:Result>
    <quafi:status>OK</quafi:status>
    <quafi:itemList>
      <rdf:Bag>
        <rdf:li
          rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml" />
      </rdf:Bag>
    </quafi:itemList>
  </quafi:Result>

  <doafi:FoodItem rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <doafi:biography
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
    <doafi:produced
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced" />
    <doafi:history
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/History" />
    <doafi:alternatives
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Alternatives" />
    <doafi:ingredients
      rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Ingredients" />
    <doafi:certification
      rdf:resource="Lea_and_Perrins/Worcestershire_Sauce/150ml/Certification" />
  </doafi:FoodItem>
</rdf:RDF>
```

Figure 5.3: GetQuery response to the query in Figure 5.2

Figure 5.3 illustrates a response to the `GetQuery` in Figure 5.2. The response is wrapped in a `Result` class, and at the base of the graph is an `rdf:Bag` containing all of the matching FoodItems. This base of the tree is considered to be 'level 1' according to the `depth` property in the query. The number of items in the `rdf:Bag` has an upper limit, as indicated in `maxItems` in the query.

As the original query indicated a `depth` of 2, each URI in the `rdf:Bag` (in this example, there is only `http://doafi.4angle.com/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml`) is expanded to include all properties directly related to this item (the 2nd level in the graph). This expansion includes the addition of Biography, Produced, History, Alternatives, Ingredients and Certification classes.

A graph representation of this result model, with an indication of depth, is included in Figure 5.4.

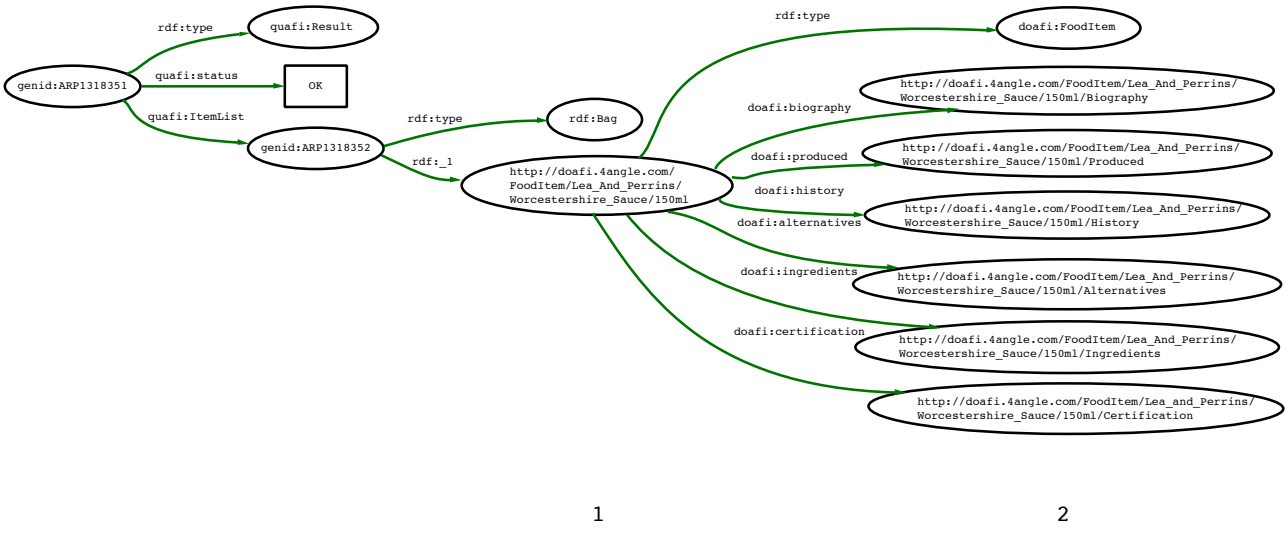

Figure 5.4: Graph representation of the model in Figure 5.3, with a depth limit of 2. Depths are indicated at the base. The items up to and including those at the 1st level would be returned if a depth limit '1' had been specified.

5.3.3 Standard vocabulary

During the development of the project, the vocabulary has been modified to enforce strict classification of properties within their respective classes. The FoodItem root is implemented as a wrapper for each of these classes. A user of the vocabulary can make use of the pre-defined classes as they are, or in a modified form. They may use the vocabulary as a part of the shared-database system or for a stand-alone metadata model. By using a unique namespace, users are able to create their own classes, subclass existing classes and add or modify the intended properties.

The vocabulary provides a loose framework for describing FoodItems, however there is no requirement to use any part of it. When a FoodItem is entered into the database, there are no restrictions placed on the amount of information entered - any or all of the classes and properties can be specified, as can use-defined properties. Interested parties are able to create their own complementary vocabularies and description involving multiple vocabularies allows software developers to include or ignore any specific vocabulary they wish.

5.3.4 Recording changes

The use of URIs and a structure as simple as subject-predicate-object used in RDF makes the storage of a history of changes made to the database non-trivial. As a URI references *unique* items, storing previous versions of an item using the original URI, even in a specific 'changes' construct, would lead to the belief that previous versions were still current when searching the database. An example of such a mis-representation is given in Table 5.1 for clarity.

In order to rectify this, I implemented functions to modify all of the URIs stored in the `modelHistory` of an item in order to make them unique, but consistently so. This allows the storage of previous versions, without interfering with the remainder of the model.

Subject	**Predicate**	**Object**
`<blankNode>`	`doafi:change`	`<http://doafi.4angle.com/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml>`
`<http://doafi.4angle.com/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml>`	`doafi:title`	'Worcestershire Sorce'

Table 5.1: Misinterpretation of a historical version of an item. While these 2 valid RDF triples are intended to indicate that this change was a previous modification, taken out of context - which in a database consisting entirely of triples is unavoidable - the second triple implies that this `doafi:title` is current.

Each URI is modified to be identified as a 'Change', and then appended to this is a counter of the actual sequential change to this model. Thus, a URI such as `http://doafi.4angle.com/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml`, modified for the first time, would become `http://doafi.4angle.com/Change/1/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml`.

5.4 Data management

In this section I described the mechanisms implemented to protect and manage the content of the database, as discussed in Section 4.3. In the first subsection I explain the process by which users may efficiently revert incorrect changes to the database. The second subsection describes a mechanism to prevent database inconsistency arising from concurrent or failed writes.

5.4.1 Rollback implementation

Despite a native implementation of transactions in the Jena API, there is currently no support for rolling back changes (any further than the current transaction) in any RDF-specific database solution.

I implemented a simple mechanism to roll back FoodItem changes to any previous state. The `modelHistory`, associated with every `FoodItem` as part of the `History` class, stores modifications to the model. The process of modifying a FoodItem (add, edit, delete or rollback) involves recording the difference between the new (post-modification) state and the previous.

In a `SetQuery`, the property `rollbackTo` indicates the edit sequence number to revert to. Reverting changes involves sequentially using statements from the `modelHistory` as if they were requested query modifications, from the most recent modification, to the value of `rollbackTo`. During this process, it is also necessary to revert the URI modifications indicated in the previous section.

5.4.2 Transactions

Database transaction methods - `begin`, `commit` and `abort` - are used in order to ensure integrity of the database when dealing with concurrent access and failure. These ensure atomicity of operations such that, if necessary, operations take place in isolation and for interactions involving modification of the database, an 'all or nothing' approach is taken - either all of the modifications are made or, in the case of some failure, none are.

Locks are implemented on the database to avoid modification conflicts, in addition to synchronisation of the Java servlets. For example, if a write, w_1, is executing, and a second write, w_2 begins executing, w_1 will complete before any modification by w_2 takes place.

Section 6.1.2 details multi-threaded testing of concurrent access to the database.

5.5 Delivery mechanism

In this section I address delivery of the web service, the interaction of components and the user interface provided. The first and second subsections detail configuration of the host machine and deployment of the DOAFI server. In the third and fourth sections I explain how the user interface interacts with the other server modules. Finally, in the fifth section I describe how user accounts can be administered in this implementation.

5.5.1 Server configuration

As indicated in Section 4.6.2, the DOAFI server is built in Java and executes as a servlet using a combination of Tomcat and the Apache HTTP server. The servlet provides the web service interface and uses the Jena API to construct read and write queries to the underlying database of RDF entries. For storage, the system uses a single MySQL database and is accessed using JDBC.

I used Apache Ant [2] to build and deploy the servlet. Both the server and administration module are packaged as WAR (Web Application Archive) files suitable for deployment using Tomcat, combined with Jena and a configured database.

5.5.2 Database

A single MySQL database is used for storing RDF data, as accessed and configured by Jena. In addition, a 'users' table was added to this database for storing usernames and associated hashed passwords with authorisation levels.

5.5.3 Receiving queries

The server receives RDF/XML queries using HTTP post. Queries comprising a combination of query and DOAFI or other vocabularies must be posted alone in the body of an HTTP packet.

The server is not capable of handling multiple queries in one packet and no additional data should be posted to the server.

5.5.4 Interpreting queries

Upon receipt of a query, the servlet executes and begins by parsing the query and constructing an in-memory representation of it. It is then determined whether the query is of type `SetQuery` or `GetQuery`. A corresponding operation is initialised by instantiating a relevant class to process the request, passing on the remainder of the query, now in an in-memory model format.

The result of the query execution is a further in-memory model which represents an answer to the query. This answer contains query status (either success or failure), and in the case of a `GetQuery`, the answer to the corresponding query. This is serialised using RDF/XML and returned to the user in a return HTTP packet.

5.5.5 Administration module

A trusted individual or party is required in this first implementation to create and administer user accounts. The administration module facilitates this, allowing the creation, deletion and modification of users. It consists of 3 XHTML interfaces which are processed using a servlet for each respective operation.

Development further than the test-phase of the server would require a more universal interface. At present, the creation is restricted to a single party. Ideally

an interface should be provided to allow users to administer themselves, with an additional interface such as this for the purpose of system administration.

Source for the module is provided in Appendix B.3.

Chapter 6

Testing

In this chapter I outline the testing which has been undertaken on both the server and vocabularies. The first section is based upon the receipt of responses to the execution of queries on the server. In the second section I evaluate the vocabularies according to the desired characteristics discussed in Section 3.4.

6.1 Server

In this section I detail testing the functional behaviour of the server by executing queries. All queries are written in RDF/XML and wrapped in an HTTP packet. The process of testing involves posting the packet to the DOAFI server using GNU Netcat [20], followed by observation of the responses to those queries and modifications to the underlying database structure.

The following subsections provide brief observations regarding the state and response of the server to queries. The first subsection is concerned with single queries to see how the server performs in normal and exceptional circumstances. The following subsection illustrates the results of testing concurrent access to the database. The third subsection briefly describes the results of testing authentication for making modifications.

6.1.1 Execution of regular operations

Tables 6.1 and 6.2 contain a summary of the results to the execution of these queries.

GetQuery operations

`GetQuery` operations perform as specified, and as indicated in Table 6.1. Actual responses to the classes of query are contained in Appendix A.4.1.

The only anomalous query is an exceptional one, where a query is made for a non-existent item. In this case, rather than explicitly specifying that no item matches the query, it returns an empty `rdf:Bag`.

	Test	**Result**
1	Direct request for a FoodItem by URI	FoodItem is returned, followed by the amount of detail specified in `depth` Response `status` is 'OK'
2	Request FoodItems which match specified properties	`rdf:Bag` of matching results are returned, followed by the amount of detail specified in `depth` Response `status` is 'OK'
3	Direct or indirect request for a FoodItem not present	empty `rdf:Bag` is returned Response `status` is 'OK'

Table 6.1: Generalised results to execution of `GetQuery` operations, organised by class of query

SetQuery operations

The 'regular' `SetQuery` operations (in Table 6.2, tests 1-4) provide output and modifications to the database as desired. Test classes 5 and 6 do not perform as originally intended. Rather than provide an error to these queries, the DOAFI server recognises them as different queries. An attempt to add an item which already exists is the same as editing that item, if there is nothing new in this 'addition', the database is not modified. The opposite is true for editing an item which does not exist, this is treated as an adding operation.

The remaining queries execute as desired, except that test 7 returns a generic error message.

In Appendix A.4.2 I provide detail of a sequence of database queries. This demonstrates a `modelHistory` after the execution of an addition, an edit, a deletion and a rollback. The database properties were as anticipated after completion of these operations, and the `modelHistory` was as illustrated in Figure A.8.

	Test	Result
1	Add a FoodItem Response	FoodItem is added to database with associated properties `status` is 'OK'
2	Remove a FoodItem	FoodItem is removed from database, along with associated properties Response `status` is 'OK'
3	Edit a FoodItem	FoodItem entry in database is altered to reflect specified property modifications, previous properties are stored in `modelHistory` Response `status` is 'OK'
4	Rollback changes	FoodItem entry in database is returned to previous state and changed properties are stored in `modelHistory` Response `status` is 'OK'
5	Add a pre-existing item	Treated like an Edit query, FoodItem is modified (if different) Response `status` is 'OK'
6	Edit a non-existent item	Treated as an Add query, FoodItem is added to database Response `status` is 'OK'
7	Remove a non-existent item	Database is not modified Response `status` is 'Error' with a generic message
8	Query is semantically incorrect	Database is not modified Response `status` is 'Error' with parsing error message
9	Query is syntactically incorrect	Database is not modified Response `status` is 'Error' with parsing error message

Table 6.2: Generalised results to execution of `SetQuery` operations, organised by class of query

6.1.2 Concurrency

Testing concurrent reading and writing involved performing simultaneous queries. Queries were either dispatched at the same instance, or one timed briefly after the other. Queries are processed in the order they are received, but due to processing and network delays, it is not possible to say which will arrive first.

The implementation of `SingleThreadModel` in the `QueryProcessor` servlet restricted the execution of queries to be performed in serial. This was shown to be the case in testing, and the result of these tests is depicted graphically in Figure 6.1.

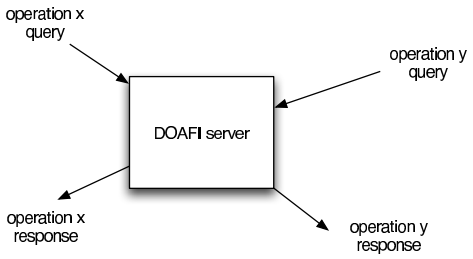

Figure 6.1: Testing concurrent requests. The first query to arrive at the server is the first to be served. During this time, the second query will wait. All concurrent requests are handled in this manner.

6.1.3 User access

All authentication testing was concerned with failure - successful authentication is extensively visible in the `SetQuery` tests. Testing was based upon a common modification query, with modified properties for the respective tests. The base query is pictured in Figure A.9 in Appendix A.4.3. Tests included incorrect `username`, incorrect `password`, missing `username` or `password` property and an authorisation level which was less than 1 (the level currently required to make database modifications).

```
<rdf:RDF
    xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/" >
  <rdf:Description rdf:nodeID="A0">
    <rdf:type
      rdf:resource="http://doafi.4angle.com/elements/q-0.1/Result"/>
    <quafi:status>
      Error: You are not authorised to make this modification
    </quafi:status>
  </rdf:Description>
</rdf:RDF>
```

Figure 6.2: Server response to failed authentication

All of the tests returned a common response, pictured in Figure 6.2. In this example, it would be possible to associate specific RDF resources with the `status` property for machine-processable error reading and recovery. Advanced error reporting is a possible area for future work.

6.2 Vocabulary

The requirements for a vocabulary were determined in Section 3.4. There is no formal process for determining whether a vocabulary suits a purpose, however the examples have been tested for semantic (using the RDF Schema in Appendix A.2.2) and XML syntactic validity. Use of the vocabulary, through the process of testing - instantiating queries and records, has illustrated that the model is capable of representing food items.

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xml:base = "http://doafi.4angle.com/FoodItem/"
  xmlns:geo = "http://www.w3.org/2003/01/geo/wgs84_pos#">

  <doafi:Produced
    rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced">
    <doafi:producer rdf:resource="Producer/Lea_And_Perrins" />
    <doafi:timestamp>2004-01-01T00:00:00-00:00</doafi:timestamp>
    <doafi:location rdf:parseType="Resource">
      <geo:lat>52.1903</geo:lat>
      <geo:long>-2.2086</geo:long>
    </doafi:location>
  </doafi:Produced>
</rdf:RDF>
```

Figure 6.3: Example data model illustrating combined vocabularies. DOAFI is combined with Geo, for representing geospacial data

The vocabulary may be combined with other vocabularies, as illustrated in Figure 6.3, where it is combined with a vocabulary for representing geospacial data. Using RDF Schema, it is possible to subclass DOAFI properties.

A primary motivation was to produce a truly machine-processable description of food items. The current vocabulary is limited, for example where the property `unitAmount` is currently used, it stores literal values, such as 'ml' to denote a measurement. Ideally, the vocabulary should be combined with a weights and measures vocabulary for concrete definitions.

Finally, there are no constraints place upon the definition of multiple properties using the DOAFI vocabulary, where they should not be supported. For example, at present, it is possible to include 2 `title` properties for a FoodItem, at the vocabulary level. At the server level, this is not allowed, as properties will overwrite each other unless an explicit container such as an `rdf:Bag` or `rdf:Seq` is used.

Chapter 7

Conclusions

In this chapter I provide closing remarks and a critical review of the project as it has been implemented. In the first section I evaluate the project according to the terms of reference. I criticise the technical implementation of the server in the second section. In the third section I indicate some conditions to be met in order to deploy the server. In the final section I outline areas of future work which I have discovered during my work on this project.

7.1 Project evaluation

I have successfully implemented a loosely-structured shared database with a web service interface, which can be used for the storage of food metadata. I have also designed a generic and extensible vocabulary for the description of food items using RDF/XML and an interface for reading and writing information to the database. Throughout this document I have discussed the position of the project in the Semantic Web, the ability to add or modify the vocabularies with no further intervention and for disparate parties to utilise one single database for multifarious uses. With regard to the technical aspects of database implementation, I have provided solutions to problems with data modelling, data integrity, communication protocols, authorisation and authentication and concurrent access. In particular, in addition to the stated aims, I have provided a unique solution for rolling back RDF databases.

I believe that in the domain of food, this project is a stepping stone towards the proliferation of metadata, which is often regarded as a potential remedy to matching problems with solutions [23].

As it happens, the end product contains little reference to food, and as such could form a general solution for adding authorisation and data integrity to an RDF database. The additional work I chose to carry out with database rollbacks was at the expense of any illustrative stub software, which I had intended to

use as proof of concept. I feel this project should be sufficient to demonstrate the need for and use of the database.

Without a higher-level application to experiment with at this time, a food producer or related organisation is unlikely to be interested in this detail. However, I would hope that deployment of the system would create sufficient interest for them to become involved.

HP Foods and Unilever both declined to participate in an evaluation of the system and without substantial public pressure, it seems unlikely that large food manufacturers would take part in populating the database.

At this statge, it would seem that the majority of aspects relating to the success of the system are related to participation and deployment. I therefore discuss this in detail in Section 7.3.

7.2 System criticism

In this section I discuss the shortcomings of the system which may need to be addressed prior to deployment. In the first subsection I consider the semantic web applicability of the system. I discuss barriers to participation in the second subsection. Finally, I suggest speed and overhead improvements in the third section.

7.2.1 Data representation and semantics

The FoodItem and query vocabularies are designed to be generic and extensible. At present, they say little about classification or relationship of items, except that one may be an ingredient of another. To use the database as a true 'Semantic Web service' will require additional work on the part of a developer. This may include the addition of more specialised user-defined vocabularies and combination with an ontology language, such as OWL [32]. The use of OWL would allow the inclusion of more complex semantic relationships.

Furthermore, when combining the server and vocabulary, properties may take only one value. One may assert multiple properties about a FoodItem, but a specific property, may only be asserted once - any attempt to add a further value to the property will replace the previous value. This was implemented in the interest of reducing data duplication.

An alternative approach would be to store multiple values, rather then replacing existing properties. Users of the data would then be free to use the multiple assertions as they wish, for example, if 100 people assert that a title is 'Worcestershire Sauce' and 2 assert that it is 'Worcestershire Sorce', then one could conclude that the former is probably correct - popularity and reinforcement dictates which option to select. This could allow, for example, the automated

discovery of synonyms.

I discuss the idea of multiple assertions for collaborative classification in Section 7.4.2.

7.2.2 Users and access

Shirky [42] suggests that there must be some barrier to participation in an open-access project; to prevent vandalism there should be a notion of user segmentation. In the present implementation, this segmentation is total - either you can or cannot participate in writing. When deployed, users should be able to create their own accounts, and hence begin editing immediately. However, in the interest of data integrity (preventing both vandalism and genuine misunderstanding), perhaps a partial segmentation would be preferable. This would involve undertaking some initialisation process in order to be considered a 'genuine' user. This is discussed further in Section 7.4.1.

The server currently has an implementation of user access control and records details of database modifications against those changes, however there is no facility or process for specifically dealing with vandalism. An administrator could manually disable a user account, but an automated heuristic would be preferable. For example, a user account could be temporarily suspended if x changes made by a user are reverted in one 24-hour period. Any solution avoiding the need for manual intervention by an administrator would be preferable.

7.2.3 Efficiency

Using an XML-based language for communication involves sizable resources to parse, which could become particularly computationally expensive when supporting multiple concurrent users. However, standardisation and reusability are a desirable trade-off for this additional processing. In addition, the combination of queries in one packed also results in reduced network overhead.

Optimisation of the server would be useful prior to deployment; for example a queue of locking transactions may be reordered such that some operations may execute concurrently.

If load were too high on the server, it may be necessary to consider a distributed approach. This could involve complete copies of the database stored across multiple machines, or individual machines could be responsible for a section of the database, as in a distributed hash table.

7.3 Deployment requirements

In this section I explain the necessary conditions which must be met in order for the database to be used. In the first subsection I describe the integration of the server into a useful application. In the second subsection I detail a modification to aid user contribution.

7.3.1 System integration

Fundamentally, for the database to be of use, it must be integrated into an application, whether in the traditional sense or a web application. Use of the database in multiple applications would encourage varied participation.

The database and content will be in the commons, and for that reason, open-source developers may be the first to implement it. To develop the concept further, I would invite participation from independent food-related organisations, particularly those interested in the certification of food, as a means of providing this description electronically.

7.3.2 Users and contribution

One test I have been unable to perform is how users will make use of the system. With few participants, a shared database is not particularly useful - part of the motivation is that many participants will populate a database much faster than could otherwise be achieved. How data is entered, for example what vocabulary and which properties are included or ignored is unknown. This is inevitable when a primary run-time variable is a group of (as yet unknown) users - which makes it very different from a 'traditional' application where most of the runtime variables can be determined in advance. The success or failure of a shared database relies almost entirely on participation.

Prior to deployment, it is therefore important that a mechanism be introduced to permit users to create and administer their own accounts and hence their ability to modify the database. Ultimate centralised control of the issuing of user accounts will not allow the system to scale.

7.4 Extensions and future work

In this section I describe modifications and modular additions which could be included in future DOAFI releases. In the first subsection I outline further mechanisms for the reduction of erroneous data. I discuss the potential benefits of user-generated vocabularies in the second subsection. In the third subsection I suggest a means of communication for users and in the fourth section I ex-

plain the potential use and benefit of digital signatures for verifying food item properties.

7.4.1 Accuracy

In addition to the ideas in Section 7.4.4, I also propose further modification which could lead to a greater degree of database accuracy.

In Section 3.5.1 I discussed algorithmically determining the similarity of entries in the database, in order to prevent similar duplicate entries. This could be implemented as a module to be executed when adding a new item to the database. The module would determine and return to the user possible duplicate entries, based on a similarity measure. Reliance is still placed upon the judgment of the user to determine whether to finally add the entry or not, but it would reduce the accidental addition of duplicate data.

In response to my criticism of user segmentation (in Section 7.2.2), I propose that further development could facilitate partial segmentation by implementing a modification queue. For example, new users could provide edits, which would be queued and manually approved by a user with a higher level of authorisation. Their modifications would continue to be queued until some predefined number of their edits have been approved, after which they obtain a higher authorisation level.

This queuing mechanism could also be used by those with no user account. It could encourage new users to participate with little knowledge or understanding, leading to a wider applicability of the system. Both of these applications would be aimed at reducing juvenile vandalism and inaccurate modifications by new users, but without discouraging them from participating.

7.4.2 Collaborative categorisation

Classification and description is often carried out by librarians or information specialists. These professionals provide formal frameworks for object categorisation and specifications for metadata description using taxonomies or ontologies.

Mathes [30] describes a more recent development in user-generated metadata - 'folksonomy' - combining folk and taxonomy. He illustrates this example with tools such as the shared bookmarking web application Del.icio.us [15] and Flickr [17], a photo sharing service. These applications allow users to associate keywords with the respective objects they are describing. This gives rise to common (and less common) patterns of classification for web resources and photos. This form of metadata generation benefits from few 'cognitive costs' or barriers required to generate the metadata, and as explained by Merholz [33], gives rise to 'ethnoclassification' - a categorisation of objects in the world by regular people.

While there are disadvantages to this simplistic model such as a lack of hierar-

chy and flat namespaces, the simplicity encourages participation and facilitates classification in a democratic fashion. The rise in popularity of this technique led me to consider the storage of multiple values for the same FoodItem property, as detailed in Section 7.2.1. Perhaps there would also be a use here for a folksonomy classification of food.

7.4.3 Social software

Some of the most popular web applications today could be considered 'social software', including weblogs: a platform for personal publishing and individual comment, wikis: which allow any user to add or edit web content in real time, or social bookmarking projects, which allow others to view your bookmarks. The success of these tools has been a consequence of individual participation as part of a group, and importantly - a group which communicates and interacts.

With the inclusion of greater individual control for user account maintenance, as outlined in Section 7.3.2, it may be beneficial to provide an additional facility for communication between users. At present, there is no mechanism for users to discuss differences of opinion, formal or informal goals and generally self-organise as a group. For a system reliant on user collaboration for the population of the database, a communication tool may increase confidence in the data through awareness of other participants.

7.4.4 Identity and trust

In the past, trust was not the problem it is today. In the context of food production, much of the food we ate was locally produced and belief in the quality of the product was a consequence of direct trust of the producer, or trust in your immediate social network, who perhaps trusted the producer. Following the rise of global trade, it is no longer likely that you will have this direct relationship with a producer, giving rise to the need for some other mechanism. For this reason, there has been recent increase in the popularity of independent organisations involved in the certification of food.

Foster et al. [18] discuss the need for the ability to uniquely identify users on the Internet, such that the occurrence of trust in social networks can be extended electronically to the global community.

As an extension to the DOAFI server implementation, I suggest a combination of these concepts. A simple implementation of digital certificates could be used, as is used by current certification authorities such as VeriSign to provide authentication for e-commerce. Food certification bodies could then be involved in issuing electronic public key certificates which attest that specific FoodItem properties have been independently verified. This would allow applications to use this information for the purpose of the automated verification of specified properties.

A facility has been provided in the DOAFI vocabulary for the assertion of any property about a FoodItem using a `Certificate`. At present, this notion does not imply any authentication or trust as detailed in this section, however public key certificates could be included as a property of the `Certificate`.

Bibliography

[1] Alexa web search. http://www.alexa.com Traffic Rank: 3 month average - referenced 2004-12-01.

[2] Apache ant. The Apache Software Foundation. http://ant.apache.org/ referenced 2005-01-21.

[3] Apache http server project. The Apache Project. http://httpd.apache.org/ referenced 2005-01-19.

[4] Apache jakarta tomcat. The Apache Jakarta Project. http://jakarta.apache.org/tomcat/ referenced 2005-01-19.

[5] Beckett, D. (Feb 2004). Rdf/xml syntax specification (revised): W3c recommendation. Technical report, W3C. http://www.w3.org/TR/rdf-syntax-grammar/ referenced 2005-02-10.

[6] Beech, D., Maloney, M., Mendelsohn, N., and Thompson, H. S. (Oct 2004). Xml schema part 1: Structures second edition. Technical report, W3C. http://www.w3.org/TR/xmlschema-1/ referenced 2005-02-10.

[7] Berners-Lee, T. (Sep 1998). Relational databases on the semantic web. http://www.w3.org/DesignIssues/RDB-RDF.html.

[8] Biron, P. V. and Malhotra, A. (Oct 2004). Xml schema part 2: Datatypes second edition. Technical report, W3C. http://www.w3.org/TR/xmlschema-2/ referenced 2005-02-10.

[9] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (Feb 1998). Extensible markup language (xml) 1.0: W3c recommendation. Technical report, W3C. http://www.w3.org/TR/1998/REC-xml-19980210 referenced 2005-02-10.

[10] Brickley, D. Geo. http://www.w3.org/2003/01/geo/ referenced 2005-02-10.

[11] Brickley, D. and Guha, R. V. (Feb 2004). Rdf vocabulary description language 1.0: Rdf schema: W3c recommendation. Technical report, W3C. http://www.w3.org/TR/rdf-schema/ referenced 2005-02-10.

[12] Bricklin, D. (2001). The cornucopia of the commons. In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly and Associates, Inc., Sebastopol, CA, United States of America, chapter 4. pages 59–63.

[13] Bricklin, D. (2001). The cornucopia of the commons. In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* [12], chapter 4, page 62.

[14] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (Mar 2001). Web services description language (wsdl) 1.1. Technical report, W3C. `http://www.w3.org/TR/wsdl` referenced 2005-03-18.

[15] del.icio.us. `http://del.icio.us` referenced 2005-03-06.

[16] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine, United States of America. `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` referenced 2005-02-20.

[17] Flickr. `http://www.flickr.com` referenced 2005-03-06.

[18] Foster, S., Hauser, J., and Jordan, K. (Aug 2003). The augmented social network: Building identity and trust into the next-generation internet. *First Monday*, 8(8). `http://firstmonday.org/issues/issue8_8/jordan/index.html` referenced 2005-02-15.

[19] freedb.org. `http://www.freedb.org` referenced 2004-11-06.

[20] Gnu netcat 0.7.1. `http://netcat.sourceforge.net/` referenced 2005-03-28.

[21] Gracenote cddb. `http://www.cddb.com` referenced 2004-11-06.

[22] Greenberg, J., Pattuelli, M. C., Parsia, B., and Robertson, W. D. (Nov 2001). Author-generated dublin core metadata for web resources: A baseline study in an organization. *Journal of Digital Information, Volume 2 Issue 2*, 2(2)(78). `http://jodi.ecs.soton.ac.uk/Articles/v02/i02/Greenberg/` referenced 2005-03-20.

[23] Griffin, L. S. and Thomas, C. F. (Dec 1999). Who will create the metadata for the internet? *First Monday*, 3(12). `http://www.firstmonday.dk/issues/issue3_12/thomas/index.html` referenced 2005-03-25.

[24] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. F. (Jun 2003). Soap version 1.2 part 1: Messaging framework: W3c recommendation. Technical report, W3C. `http://www.w3.org/TR/soap12/` referenced 2005-03-18.

[25] Guha, R. V. rdfdb. `http://www.guha.com/rdfdb/` referenced 2004-11-05.

[26] Ics-forth validating rdf parser. Institute of Computer Science - Foundation of Research Technology Hellas, Greece. `http://athena.ics.forth.gr:8080/RDF/` referenced 2005-03-01.

[27] Jena semantic web framework. `http://jena.sourceforge.net` referenced 2004-11-05.

[28] Lawrence, F. (2004). *Not On The Label: What really goes into the food on your plate*, Penguin Books, London, United Kingdom, chapter unknown. page 204.

[29] Leaf tracks. http://www.leafuk.com/leaf/consumers/tracks.asp referenced 2004-11-06.

[30] Mathes, A. (Dec 2004). Folksonomies - cooperative classification and communication through shared metadata. http://www.adammathes.com/academic/computer-mediated-communication/folk%sonomies.html referenced 2005-03-25.

[31] Mayhem and Chaos (Feb 2003). Musicbrainz metadata initiative 2.1. http://www.musicbrainz.org/MM/ referenced 2004-11-02.

[32] McGuinness, D. L. and van Harmelen, F. (Feb 2004). Owl web ontology language. Technical report, W3C. http://www.w3.org/TR/owl-features/ referenced 2005-03-15.

[33] Merholz, P. (Oct 2004). Metadata for the masses. http://www.adaptivepath.com/publications/essays/archives/000361.php referenced 2005-03-20.

[34] Miller, E., Swick, R., Brickley, D., McBride, B., Hendler, J., Schreiber, G., and Connolly, D. (2004). W3 semantic web. http://www.w3.org/2001/sw/ referenced 2004-12-01.

[35] Musicbrainz. http://www.musicbrainz.org referenced 2004-11-06.

[36] Ogbuji, U. (Feb 2002). Using rdf with soap: Beyond remote procedure calls. *IBM*. http://www-128.ibm.com/developerworks/webservices/library/ws-soaprdf/ referenced 2004-11-29.

[37] (2002). Farming and food a sustainable future. Policy Commission on the Future of Farming and Food, Cabinet Office, London.

[38] Powers, S. (2003). *Practical RDF*, O'Reilly and Associates, Inc., Sebastopol, CA, United States of America, chapter 6. page 106.

[39] Rheingold, H. (2003). *Smart Mobs: The Next Social Revolution*, Perseus Publishing, Cambridge, MA, United States of America, chapter 5. page 116.

[40] Rivest, R. (Apr 1992). The md5 message-digest algorithm. Technical report, Network Working Group, MIT Laboratory for Computer Science. http://www.ietf.org/rfc/rfc1321.txt referenced 2005-03-30.

[41] Sesame. http://www.openrdf.org referenced 2004-11-05.

[42] Shirky, C. (Apr 2003). A group is its own worst enemy. *Networks, Economics, and Culture mailing list*. http://shirky.com/writings/group_enemy.html referenced 2005-03-05.

[43] Thornton, C. (2000). *Truth From Trash*, The MIT Press, Cambridge, Massachusetts, United States of America, chapter 2. pages 19 – 30.

[44] Various (2004). Agricultural ontology service project. *Food & Agriculture Organisation of the United Nations*. http://www.fao.org/agris/aos/ referenced 2004-11-28.

[45] Various (Dec 2004). Wikipedia:replies to common objections. http://en.wikipedia.org/wiki/Wikipedia:Replies_to_common_objections referenced 2004-12-05.

[46] Various (Feb 2005). Wikipedia:replies to common objections: Accepting edits. http://en.wikipedia.org/wiki/Replies_to_common_objections#Errors_and_om%issions referenced 2005-02-20.

[47] W3 rdf validator. W3C. http://www.w3.org/RDF/Validator/ referenced 2004-11-27.

[48] Wikipedia. http://www.wikipedia.org referenced 2004-11-06.

Appendix A

Report appendices

This chapter contains items which are referenced from the body of the report. It primarily contains detail which was too extensive for the main body of the report. The sections are structured according to the parts of the report they relate to.

A.1 Requirements Analysis

User requirements

Software developers will require a well known, cross platform interface to the database. They will need to be able to implement functionality to query the database to the fullest extent of the data contained within it. It should be possible for them to augment the data extracted from the system with their own data, should they wish to. Access to the database is covered more fully in Sections 3.5.1 and 3.6.

End users, regardless of the application they are using, will require the ability to edit and query the database.

Producers

In particular, producers following ethical or other noteworthy practices may consider it beneficial to participate and actively market the particular features of their product; whereas producers of more homogenous products may not be so willing to disclose their production practices, especially where they may be considered undesirable.

XML and RDF compared

The choice of model appears to come down to whether to use XML to model the data (either using DTDs or XML Schema) or RDF and RDF/XML, a totally different paradigm. The points can be summarised as below:

- XML documents are naturally hierarchical
  - When parsing XML, an element is not 'complete' until the end tag has been reached. For nested XML documents, many tags will need to reside in memory as the document is processed; for large documents this could be an expensive task.

    While it does not seem likely a system such as the proposed one would generate documents of sufficient size to cause problems for modern computers, it is still a consideration. It would also be preferable to not make an assumption about document size in the outset.
  - Searching non-RDF XML for a specific piece of data requires viewing the tree up until the point the data is found in order to determine that the context of the data is correct. This is not necessary in RDF/XML, where searches can be done for a specific triple, independent of the remainder of the document.

    It would be particularly useful to make searching as fast as possible. Potentially RDF triples could easily be stored in a database, which would speed up access if the search triple is known directly.
  - While fundamentally the same syntactically, XML documents using the RDF model are not as human readable as a 'regular' XML document, that is one where the tree structure is intentionally intended to reflect the structure of the data.

    The use of RDF here should not pose a problem as the documents are not meant to be read by humans, the system is concerned with information interchange using a web services approach. Interaction can be via some either via language support for RDF or some API, meaning that little RDF will be viewed by humans. Tools for viewing and validating RDF data can be used for debugging purposes.
- XML Schema 'understanding' must be complete
  - As non-RDF XML is based on a tree structure, a whole schema must be complete in order for it to be used. Using RDF this is not the case and it is possible to work with what you have.

    If it is possible to devise some way to uniquely determine food products then this feature of RDF could be useful where only partial information is available for a particular product being described. This could be particularly useful for interference engines working with only partial knowledge.
- Combining vocabularies and documents

  - Using RDF the ordering of RDF triples is unimportant, however for non-RDF XML the ordering is important - there is only one valid XML tree.

    Using RDF would ease the merging of multiple documents and vocabularies, no attention need be paid to correctly ordering the data according to a tree-like structure.

  - Using XML Schema or DTDs requires thought early on in the development process to determine where 'foreign' elements may be present, ie elements from other vocabularies. Also one must state which elements are optional, obligatory etc. RDF and RDF Schema do not require this, people using an RDF Schema are free to make use of elements from any other namespaces while describing relationships.

    This property makes RDF naturally extensible, which is particularly useful as it is beyond the scope of this project to create concrete and comprehensive vocabularies for every available form of food. Using RDF will facilitate this at a later stage by anybody interested in using the schemas.

- Communication of XML requires common syntax

  - Parties (physical or electronic) wishing to communicate must first determine a common syntax for their documents - the idea behind the Schemas. In the case of DOAFI, the syntax would be up to me as a designer. Using RDF communication could take place, independent of syntax, using the notion of equivalence.

    Using RDF, other developers would not be bound by the syntax I specify, meaning that they could work independently of any mistakes I make. By making the database open access using a web services approach, and making the schema definitions and data publicly available could aid further development of the system by others.

Essentially and in addition to the above points, the difference between approaches of non-RDF XML and RDF seems to be that XML and the available schema definition specifications are concerned with data types and simple and complex element structures to describe data for data interchange. Conversely, using pre-existing or user defined RDF vocabularies/ontologies lead to the description of the meaning of the data *in the real world*, but in a way which still facilitates data interchange.

A.2 Design

A.2.1 RDF instantiation

The following is an example instantiation of a model for Worcestershire Sauce using the DOAFI vocabulary combined with the Geo vocabulary for geospacial data.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xml:base = "http://doafi.4angle.com/"
  xmlns:geo = "http://www.w3.org/2003/01/geo/wgs84_pos#">

  <doafi:FoodItem rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <doafi:biography
    rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
    <doafi:produced
    rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced" />
    <doafi:history
    rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/History" />
    <doafi:alternatives
    rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Alternatives" />
    <doafi:ingredients
    rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Ingredients" />
    <doafi:certification
    rdf:resource="FoodItem/Lea_and_Perrins/Worcestershire_Sauce/150ml/Certification" />
  </doafi:FoodItem>

  <doafi:Ingredients
  rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Ingredients">
    <doafi:ingredientList>
      <rdf:Seq>
        <rdf:_1>
          <doafi:IngredientItem>
            <doafi:ingredient rdf:resource="FoodITem/Generic/Malt_Vinegar" />
            <doafi:amount>50</doafi:amount>
            <doafi:amountUnit>ml</doafi:amountUnit>
            <doafi:reason>it is the main flavour in Worcestershire Sauce</doafi:reason>
          </doafi:IngredientItem>
        </rdf:_1>
        <rdf:_2>
          <doafi:IngredientItem>
              <doafi:ingredient rdf:resource="FoodItem/Generic/Spirit_Vinegar" />
            <doafi:amount>20</doafi:amount>
            <doafi:amountUnit>ml</doafi:amountUnit>
            <doafi:reason>it is the main flavour in Worcestershire Sauce</doafi:reason>
          </doafi:IngredientItem>
        </rdf:_2>
        <!-- ... -->
      </rdf:Seq>
    </doafi:ingredientList>
  </doafi:Ingredients>

  <doafi:History rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/History">
    <doafi:modelHistory>
      <rdf:Seq>
        <rdf:_1>
          <doafi:ModelHistoryItem>
            <doafi:username>tom2</doafi:username>
            <doafi:editComment>added item to database</doafi:editComment>
            <doafi:timestamp>2005-03-05 19:01:06</doafi:timestamp>
            <doafi:changes>
              <rdf:Description
                rdf:about="http://doafi.4angle.com/Change/1/FoodItem/Lea_And_Perrins/
                Worcestershire_Sauce/150ml">
                <doafi:title>Worcestershire Sorcerer</doafi:title>
              </rdf:Description>
            </doafi:changes>
```

```
        </doafi:ModelHistoryItem>
      </rdf:_1>
      <rdf:_2>
        <doafi:ModelHistoryItem>
          <doafi:username>steve</doafi:username>
          <doafi:editComment>modified nothing</doafi:editComment>
          <doafi:timestamp>2005-02-01 11:21:22</doafi:timestamp>
          <doafi:changes>changing</doafi:changes>
        </doafi:ModelHistoryItem>
      </rdf:_2>
    </rdf:Seq>
  </doafi:modelHistory>
</doafi:History>

<doafi:Biography rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
  <doafi:title>Worcestershire Sauce</doafi:title>
  <doafi:useBefore>2005-03-01T00:00:00-00:00</doafi:useBefore>
  <doafi:amount>150</doafi:amount>
  <!--reference to ml definition would be preferable to literal below-->
  <doafi:amountUnit>ml</doafi:amountUnit>
</doafi:Biography>


<!-- information about the most recent stage of production -->
<doafi:Produced rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced">
  <doafi:producer rdf:resource="Producer/Lea_And_Perrins" />
  <doafi:timestamp>2004-01-01T00:00:00-00:00</doafi:timestamp>
  <doafi:location rdf:parseType="Resource">
    <geo:lat>52.1903</geo:lat>
    <geo:long>-2.2086</geo:long>
  </doafi:location>
  <doafi:process>mix all the ingredients together</doafi:process>
</doafi:Produced>

<!-- properties are asserted -->
<doafi:Certification
rdf:about="FoodItem/Lea_and_Perrins/Worcestershire_Sauce/150ml/Certification">
  <!-- certification about the properties of the food item -->
  <doafi:type rdf:resource="Certificate/HM_the_Queen" />
  <doafi:issuedBy>Royal Palace</doafi:issuedBy>
  <doafi:reason>The queen uses it</doafi:reason>
  <doafi:issueDate>2004-01-01T00:00:00-00:00</doafi:issueDate>
</doafi:Certification>

<doafi:Certificate rdf:about="Certificate/HM_the_Queen">
  <doafi:title>By appointment to Her Majesty the Queen</doafi:title>
  <doafi:webAddress>http://www.royalwarrant.org</doafi:webAddress>
</doafi:Certificate>

<doafi:Alternatives
rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Alternatives">
  <doafi:alternativeList>
    <rdf:Seq>
      <rdf:_1>
        <doafi:AlternativeItem>
          <doafi:alternative rdf:resource="FoodItem/Tiparos/Fish_Sauce/23floz" />
          <doafi:reason>provides a good seasoning flavour</doafi:reason>
        </doafi:AlternativeItem>
      </rdf:_1>
      <!-- ... -->
    </rdf:Seq>
  </doafi:alternativeList>
```

```
  </doafi:Alternatives>


  <doafi:Producer rdf:about="Producer/Lea_And_Perrins">
    <doafi:name>Lea Perrins</doafi:name>
    <doafi:webAddress>http://www.leaperrins.com</doafi:webAddress>
    <doafi:location rdf:parseType="Resource">
      <geo:lat>52.1903</geo:lat>
      <geo:long>-2.2086</geo:long>
    </doafi:location>
  </doafi:Producer>

</rdf:RDF>
```

A.2.2 Formalising the vocabularies with RDFS

The following two subsections contain the RDF Schema for the DOAFI and QUAFI (query) vocabularies. These can be used for the purpose of semantic validation, and discovery of the items and relationship of vocabulary elements.

DOAFI vocabulary

```
<?xml version="1.0"?>
<rdf:RDF xml:lang="en"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/FoodItem">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Class for food item</rdfs:label>
    <rdfs:comment xml:lang="en">Wrapper for all of the properties of a food item</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/AlternativeItem">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Alternative to this FoodItem</rdfs:label>
    <rdfs:comment xml:lang="en">A product which could be used instead of this product</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Producer">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Producer of a FoodItem</rdfs:label>
    <rdfs:comment xml:lang="en">Defines the producer of the specified FoodItem</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Ingredients">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Ingredients of a FoodItem</rdfs:label>
    <rdfs:comment xml:lang="en">Container class for list of ingredients</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Recording changes to a FoodItem</rdfs:label>
    <rdfs:comment xml:lang="en">Wrapper class for change metadata to a FoodItem in the
      database</rdfs:comment>
```

```
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Certificate">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Defines a food certificate</rdfs:label>
  <rdfs:comment xml:lang="en">A certificate is used to assert a property about a
    FoodItem</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Biography">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem biography</rdfs:label>
  <rdfs:comment xml:lang="en">Biographical detail about a FoodItem</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Alternatives">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem alternatives</rdfs:label>
  <rdfs:comment xml:lang="en">Wrapper for list of alternative FoodItems</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Certification">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem certification</rdfs:label>
  <rdfs:comment xml:lang="en">Wrapper for list of certificates associated with this
    FoodItem</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/Produced">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem production detail</rdfs:label>
  <rdfs:comment xml:lang="en">Information about the production of a FoodItem</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/IngredientItem">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem ingredient wrapper</rdfs:label>
  <rdfs:comment xml:lang="en">Wraps a FoodItem as an ingredient with ingredient
    metadata</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/History"/>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/s-0.1/DeleteOnRevert">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Delete on revert</rdfs:label>
  <rdfs:comment xml:lang="en">Was new item at this modification sequence, when reverting it
    should be deleted</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/name">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Object name</rdfs:label>
  <rdfs:comment xml:lang="en">Literal name for an object, such as a Producer,
    not a FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Producer"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/title">
  <rdfs:label xml:lang="en">Object title</rdfs:label>
  <rdfs:comment xml:lang="en">Literal title for an Object, such as FoodItem or
    Certificate</rdfs:comment>
```

```
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Biography"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certificate"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/ingredient">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem ingredient</rdfs:label>
  <rdfs:comment xml:lang="en">The object of this property is an ingredient of the parent
    FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/IngredientItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/type">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Certificate type</rdfs:label>
  <rdfs:comment xml:lang="en">Defines the type of Certificate issued in this
    Certification</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certification"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certificate"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/amountUnit">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Amount unit of FoodItem</rdfs:label>
  <rdfs:comment xml:lang="en">Defines the unit of amount of the item being described</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Biography"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/IngredientItem"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/username">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Username</rdfs:label>
  <rdfs:comment xml:lang="en">DOAFI username, as recorded when making model
    modifications</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/issuedBy">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Certificate issuing body</rdfs:label>
  <rdfs:comment xml:lang="en">Detail of who issued the Certificate</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certification"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/alternatives">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Alternatives wrapper</rdfs:label>
  <rdfs:comment xml:lang="en">This property indicates the alternatives to a FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Alternatives"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/alternativeList">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">List of alternative FoodItems</rdfs:label>
```

```
  <rdfs:comment xml:lang="en">Wrapper for bag of alternative FoodItems</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Alternatives"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/issueDate">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Certificate issue date</rdfs:label>
  <rdfs:comment xml:lang="en">Date of issue of a certificate</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certification"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/certification">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Certification pointer</rdfs:label>
  <rdfs:comment xml:lang="en">Indicates the URI of certification details</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certification"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/amount">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem amount</rdfs:label>
  <rdfs:comment xml:lang="en">Amount of a FoodItem being described as a literal
    value</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/IngredientItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Biography"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/useBefore">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Use before date</rdfs:label>
  <rdfs:comment xml:lang="en">Date indicating when FoodItem should b e consumed by</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Biography"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/history">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">History pointer</rdfs:label>
  <rdfs:comment xml:lang="en">Pointer to URI of History of this FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/History"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/produced">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Produced pointer</rdfs:label>
  <rdfs:comment xml:lang="en">Pointer to URI of Produced of this FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Produced"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/alternative">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Alternative FoodItem</rdfs:label>
  <rdfs:comment xml:lang="en">Concrete pointer to the FoodItem which is an alternative
    to this</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/AlternativeItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
```

```
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/location">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Geographic location</rdfs:label>
  <rdfs:comment xml:lang="en">Location at which something happened</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Producer"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Produced"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/biography">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Biography pointer</rdfs:label>
  <rdfs:comment xml:lang="en">Pointer to URI of Biography of this FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
  <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Biography"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/changes">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">FoodItem history of changes</rdfs:label>
  <rdfs:comment xml:lang="en">Pointer to the changes which took place at each
    modification</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/ingredientList">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Ingredient sequence pointer</rdfs:label>
  <rdfs:comment xml:lang="en">Pointer to sequence of Ingredients</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Ingredients"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/webAddress">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Web address</rdfs:label>
  <rdfs:comment xml:lang="en">Web address with more information about this entity</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Producer"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certificate"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/timestamp">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Timestamp of modification</rdfs:label>
  <rdfs:comment xml:lang="en">Timestamp of modification</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Produced"/>
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/editComment">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
  <rdfs:label xml:lang="en">Edit comment</rdfs:label>
  <rdfs:comment xml:lang="en">Comment relating to each edit of the FoodItem</rdfs:comment>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

```
  <rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/reason">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Reason</rdfs:label>
    <rdfs:comment xml:lang="en">Reason for.. relationship, modification etc. </rdfs:comment>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/AlternativeItem"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Certification"/>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/IngredientItem"/>
  </rdf:Property>

  <rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/modelHistory">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Model history pointer</rdfs:label>
    <rdfs:comment xml:lang="en">Pointer to sequence of history changes to this
      FoodItem</rdfs:comment>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/History"/>
    <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
  </rdf:Property>

  <rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/producer">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Producer pointer</rdfs:label>
    <rdfs:comment xml:lang="en">Pointer to Producer of who Produced this FoodItem</rdfs:comment>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Produced"/>
    <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Producer"/>
  </rdf:Property>

  <rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/ingredients">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Ingredient pointer</rdfs:label>
    <rdfs:comment xml:lang="en">Pointer to Ingredients of this FoodItem</rdfs:comment>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
    <rdfs:range rdf:resource="http://doafi.4angle.com/elements/s-0.1/Ingredients"/>
  </rdf:Property>

  <rdf:Property rdf:about="http://doafi.4angle.com/elements/s-0.1/process">
    <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/s-0.1/" />
    <rdfs:label xml:lang="en">Production process</rdfs:label>
    <rdfs:comment xml:lang="en">Detail about how this FoodItem was produced</rdfs:comment>
    <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/s-0.1/Produced"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  </rdf:Property>

</rdf:RDF>
```

QUAFI vocabulary

```
<?xml version="1.0"?>
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/q-0.1/GetQuery">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Class for reading</rdfs:label>
  <rdfs:comment xml:lang="en">Look up query for a FoodItem using DOAFI in the query</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/q-0.1/SetQuery">
```

```
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Class for modifying</rdfs:label>
  <rdfs:comment xml:lang="en">Like GetQuery, but used to set/delete/add items in a query instead
    of getting them</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://doafi.4angle.com/elements/q-0.1/Result">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Result response</rdfs:label>
  <rdfs:comment xml:lang="en">A wrapper for the result of a query response</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/depth">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Query depth</rdfs:label>
  <rdfs:comment xml:lang="en">a depth limit, indicating the level of information to be
    returned</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/GetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/deleteItems">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Delete or update items?</rdfs:label>
  <rdfs:comment xml:lang="en">A flag to set to true if these items should be deleted rather
    than updated</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/SetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/maxItems">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Maximum number of first-level items to return</rdfs:label>
  <rdfs:comment xml:lang="en">the maximum number of items (at depth 1) matching this
    query</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/GetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/status">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Query status</rdfs:label>
  <rdfs:comment xml:lang="en">an indicator of whether the query was successful - a property
    of Result</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/Result" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/itemList">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Item List</rdfs:label>
  <rdfs:comment xml:lang="en">List of items matching a GetQuery, stored in an RDF:bag, if
    there are any</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/Result" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/username">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">DOAFI username</rdfs:label>
  <rdfs:comment xml:lang="en">DOAFI username for making database modifications (required for
     a SetQuery)</rdfs:comment>
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/SetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/rollbackTo">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Rollback to sequence number</rdfs:label>
  <rdfs:comment xml:lang="en">Modification version state to rollback to, according to the
     items in movementHistory</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/SetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/password">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">DOAFI hashed password</rdfs:label>
  <rdfs:comment xml:lang="en">DOAFI password, hashed using md5, for making database
     modifications (required for a SetQuery)</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/SetQuery" />
</rdf:Property>

<rdf:Property rdf:about="http://doafi.4angle.com/elements/q-0.1/editComment">
  <rdfs:isDefinedBy rdf:resource="http://doafi.4angle.com/elements/q-0.1/" />
  <rdfs:label xml:lang="en">Edit comment</rdfs:label>
  <rdfs:comment xml:lang="en">Comment explaining database modifications</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="http://doafi.4angle.com/elements/q-0.1/SetQuery" />
</rdf:Property>

</rdf:RDF>
```

A.3 Implementation

A.3.1 Code walkthrough

In this section I provide an overview of the classes and objects which make up the DOAFI server, their interactions and flow of execution. The complete, commented source code is included in Appendix B.1. In the first subsection I explain how the server receives queries and dispatches them to be executed. The following subsections detail query execution.

Query execution and interface

The `QueryProcessor` class provides the web services interface to the DOAFI database. It is a subclass of `javax.servlet.http.HttpServlet` and overrides the `doGet` and `doPost` methods. It implements `SingleThreadModel` in order to synchronise access to the database. Assuming no errors occur in the reading and parsing of a query, either a `SetQuery` or `GetQuery` is instantiated to process and execute the query and return a response.

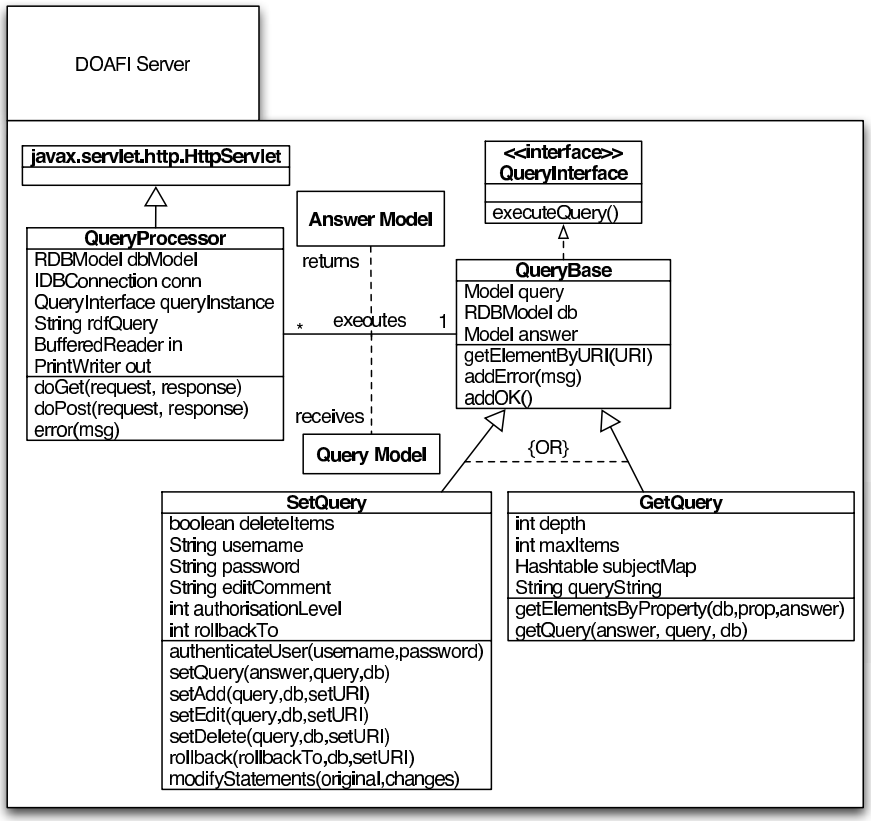

Figure A.1: DOAFI server class diagram illustrating the primary classes and methods in the `ds` package

`QueryBase` is a superclass for query execution whose constructor receives the `query` and `db Models` from `QueryProcessor` via the instantiation of the subclasses, `SetQuery` and `GetQuery`. It also creates an empty `Model - answer`.

Structure is imposed upon `SetQuery` and `GetQuery` by implementing a `QueryInterface`. In this release, this ensures the provision of an `executeQuery` method.

Figure A.1 provides a graphical overview of the main features and class structure.

SetQuery

A `SetQuery` object is responsible for any query requiring modification to the database. After instantiation, execution of the query processing begins with a call to `executeQuery`. Prior to making any modifications, the `authenticateUser` method is called, which returns an authorisation level. Execution continues

by calling `setQuery`, which first begins a transaction and determines the URI (`setURI`) of the item to modify. Based on the inclusion of a `rollbackTo` query property and the existence of the FoodItem URI in the database, the method determines whether to execute `rollback`, `setEdit`, `setDelete` or `setAdd`. Upon the successful completion of any of these methods, an `answer` is returned to the `QueryProcessor` which confirms this success and the transaction is committed; otherwise it is aborted and an error returned. Each method also adds modification metadata (`username`, `editComment`, `timestamp` and the changes) to the end of the `modelHistory` sequence to facilitate rollbacks.

When adding new FoodItems, a call to `setAdd` simply inserts the new items to the database and records the changes.

If the item already exists in the database, `setEdit` is used to modify the existing entry. The new statements to be included in this model are calculated using a set `difference` operation between `query` and `db`. This is also used for calculating changes for rollbacks, as stored in the `modelHistory`. A subsequent call to `modifyStatements` determines whether statements are new or alterations to previous statements.

If the `deleteItems` flag is set, `setDelete` is called and an `intersection` is used to determine the common statements in the query and database, which are then removed. This works as `setEdit` except that dependencies must be calculated to determine which statements not to remove. As a query must be made in context, the statements relating to that context may also required for other properties (possibly not being deleted), and so these must be deleted.

The `rollback` method takes an integer parameter which determines the modification sequence number to revert to. It then iterates backwards over the changes from the most recent to the desired state, rolling back changes during each iteration. The new state is therefore reached by sequentially aggregating of all of the changes.

As indicated in Section 5.3.4, `addChangeToURI` and `removeChangeFromURI` are implemented to modify URIs in a given in-memory `Model` to contain an additional 'Change' parameter, when stored in a `modelHistory`.

GetQuery

Two primary methods are used to read from the database, `getElementsByProperty` and `getElementByURI`. In case the query simply requires the return of an item specified by a URI, the latter is called.

Alternatively, `getElementsByProperty` is called to determine matches (up to the maximum supplied in `maxItems` to a query where an explicit URI is not supplied. Both approaches return a `QueryWrapper` object, which includes a `visitQueue` of items to expand in the next level of the graph and an `answerModel` of statements to be included in an answer.

The `getQuery` method then performs a breadth-first search of the `visitQueue` by iterating over the URIs in the queue, calling `getElementByURI` on each one. Each call to `getElementByURI` returns further `visitQueue` and `answerModel` objects - the contents of which are added to their local equivalents. This iteration continues until the `depth` is reached in the graph. At this point, the answer is returned to the `QueryProcessor`.

Rollback implementation

Figure A.2 illustrates a rollback query

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/FoodItem/">

  <quafi:SetQuery
    rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml/">
    <quafi:username>tom</quafi:username>
    <quafi:password>5f4dcc3b5aa765d61d8327deb882cf99</quafi:password>
    <quafi:editComment>reverted incorrect name change</quafi:editComment>
    <quafi:rollbackTo>2</quafi:rollbackTo>
  </quafi:SetQuery>
</rdf:RDF>
```

Figure A.2: Example of a rollback query. This will return the model for the stated FoodItem to the state it was in after the 2nd edit.

A.4 Testing

A.4.1 GetQuery

Figure A.3 illustrates a sample of a response to a `GetQuery` for properties (not a concrete URI). Figure A.4 illustrates a sample response to a `GetQuery` where a URI was specified. Figure A.5 is a response to a query for an item which was not present in the database. It contains an empty `rdf:Bag` of solutions, because there are none.

A.4.2 SetQuery

Figure A.6 illustrates a query to delete the `title` from a `Biography` of a FoodItem. Figure A.7 illustrates a `SetQuery` to rollback changes to a previous version of the model.

```
<rdf:RDF
    xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xml:base="http://doafi.4angle.com/"
    xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/" >
  <rdf:Description rdf:nodeID="A0">
    <quafi:status>OK</quafi:status>
    <quafi:itemList rdf:nodeID="A1"/>
    <rdf:type rdf:resource="http://doafi.4angle.com/elements/q-0.1/Result"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
    <rdf:_1 rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml"/>
  </rdf:Description>
</rdf:RDF>
```

Figure A.3: Response to query by properties with `depth` = 1, `maxItems` = 5

```
<rdf:RDF
    xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xml:base="http://doafi.4angle.com/FoodItem/"
    xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/" >
  <rdf:Description rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <doafi:alternatives rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Alternatives"/>
    <doafi:produced rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Produced"/>
    <doafi:history rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/History"/>
    <rdf:type rdf:resource="http://doafi.4angle.com/elements/s-0.1/FoodItem"/>
    <doafi:biography rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography"/>
    <doafi:certification rdf:resource="Lea_and_Perrins/Worcestershire_Sauce/150ml/Certification"/>
    <doafi:ingredients rdf:resource="Lea_And_Perrins/Worcestershire_Sauce/150ml/Ingredients"/>
  </rdf:Description>
</rdf:RDF>
```

Figure A.4: Response to query by URI with `depth` = 1, `maxItems` = 5

Figure A.8 illustrates the `modelHistory` of a FoodItem after a number of modifications, finishing with a rollback.

A.4.3 User access

Figure A.9 illustrates the base query on which all of the authentication was based, as referenced from Section 6.1.3.

```
<rdf:RDF
    xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/" >
  <rdf:Description rdf:nodeID="A0">
    <rdf:type rdf:resource="http://doafi.4angle.com/elements/q-0.1/Result"/>
    <quafi:itemList rdf:nodeID="A1"/>
    <quafi:status>OK</quafi:status>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
  </rdf:Description>
</rdf:RDF>
```

Figure A.5: Response to query for non-existent item

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi = "http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/"
  xmlns:geo = "http://www.w3.org/2003/01/geo/wgs84_pos#">

  <quafi:SetQuery rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <quafi:username>dave1</quafi:username>
    <quafi:password>9f9d51bc70ef21ca5c14f307980a29d8</quafi:password>
    <quafi:editComment>deleted product</quafi:editComment>
    <quafi:deleteItems>1</quafi:deleteItems>
    <doafi:biography
      rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
  </quafi:SetQuery>
  <doafi:Biography rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
    <doafi:title>Worcestershire Sauca</doafi:title>
  </doafi:Biography>
</rdf:RDF>
```

Figure A.6: `SetQuery` operation to delete a property. The `title` property of the item `biography` is removed.

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi="http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/FoodItem/">

  <quafi:SetQuery
    rdf:about="Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <quafi:username>tom</quafi:username>
    <quafi:password>5f4dcc3b5aa765d61d8327deb882cf99</quafi:password>
    <quafi:editComment>reverted incorrect name change</quafi:editComment>
    <quafi:rollbackTo>1</quafi:rollbackTo>
  </quafi:SetQuery>
</rdf:RDF>
```

Figure A.7: `SetQuery` operation to rollback changes to the data model. The model is reverted to the state of the first edit.

```
<rdf:RDF
    xmlns:doafi="http://doafi.4angle.com/elements/s-0.1/"
    xml:base="http://doafi.4angle.com/Change/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<doafi:modelHistory>
  <rdf:Seq>
    <rdf:li>
      <doafi:ModelHistoryItem>
        <doafi:editComment>modified product title</doafi:editComment>
        <doafi:timestamp>2005-04-26 17:54:04</doafi:timestamp>
        <doafidoafi:username>dave1</doafi:username>
      </doafi:ModelHistoryItem>
    </rdf:li>
    <rdf:li>
      <doafi:ModelHistoryItem>
        <doafi:changes
          rdf:resource="2/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography"/>
        <doafi:editComment>modified product title</doafi:editComment>
        <doafi:timestamp>2005-04-26 17:54:10</doafi:timestamp>
        <doafi:username>dave1</doafi:username>
      </doafi:ModelHistoryItem>
    </rdf:li>
    <rdf:li>
      <doafi:ModelHistoryItem>
        <doafi:changes>
          <rdf:Description
            rdf:about="3/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
            <doafi:title>Worcestershire Sauca</doafi:title>
          </rdf:Description>
        </doafi:changes>
        <doafi:editComment>deleted product</doafi:editComment>
        <doafi:timestamp>2005-04-26 17:55:00</doafi:timestamp>
        <doafi:username>dave1</doafi:username>
      </doafi:ModelHistoryItem>
    </rdf:li>
    <rdf:li>
      <doafi:ModelHistoryItem>
        <doafi:changes
          rdf:resource="4/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography"/>
        <doafi:editComment>reverted incorrect name change</doafi:editComment>
        <doafi:timestamp>2005-04-26 17:55:17</doafi:timestamp>
        <doafi:username>tom</doafi:username>
      </doafi:ModelHistoryItem>
    </rdf:li>
  </rdf:Seq>
</doafi:modelHistory>
<rdf:Description
  rdf:about="4/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
  <doafi:title>Worcestershire Sauca</doafi:title>
</rdf:Description>
<rdf:Description
  rdf:about="2/FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
  <doafi:title>Worcestershire Sauce</doafi:title>
</rdf:Description>
</rdf:RDF>
```

Figure A.8: The FoodItem `modelHistory` after compound changes. This represents the changes made and stores the previous values of modified properties. In order (to be read top to bottom), this includes addition of this item, editing it once, a further edit, a delete (from Figure A.6) and a rollback to the first state (using query in Figure A.7.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:doafi = "http://doafi.4angle.com/elements/s-0.1/"
  xmlns:quafi = "http://doafi.4angle.com/elements/q-0.1/"
  xml:base = "http://doafi.4angle.com/"
  xmlns:geo = "http://www.w3.org/2003/01/geo/wgs84_pos#">

  <quafi:SetQuery rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml">
    <quafi:username>robert</quafi:username>
    <quafi:password>9f9d51bc70ef21ca5c14f307980a29d8</quafi:password>
    <quafi:editComment>modified product title</quafi:editComment>
    <doafi:biography
      rdf:resource="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography" />
  </quafi:SetQuery>

  <doafi:Biography
    rdf:about="FoodItem/Lea_And_Perrins/Worcestershire_Sauce/150ml/Biography">
    <doafi:title>Worcestershire Saucerer</doafi:title>
    <doafi:useBefore>2005-03-01T00:00:00-00:00</doafi:useBefore>
    <doafi:amount>150</doafi:amount>
    <doafi:amountUnit>ml</doafi:amountUnit>
  </doafi:Biography>
</rdf:RDF>
```

Figure A.9: Base example of authentication query. Properties of the `SetQuery` are modified for the various tests

Appendix B

Source code

B.1 DOAFI package

A walkthrough and class diagram for the server package are provided in Appendix A.3.1.

B.1.1 QueryProcessor

```
package ds;

import vocabulary.*;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.xalan.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.vocabulary.*;

/**
 * A servlet for processing HTTP POSTed input requests for information from the DOAFI database.
 * This class is an extension of HttpServlet and implements a SingleThreadModel to ensure that
 * execution takes place in isolation and database access is not performed concurrently.
 * As data is posted, it is processed by the doPost method, which generates an instance of the
 * relevant classto deal with the query.
 *
 * @author Tom Betts
 */

public class QueryProcessor extends HttpServlet implements SingleThreadModel {
```

```
private PrintWriter out; // for writing back to the client
private BufferedReader in; // for reading from the client
private QueryInterface queryInstance = null; // instance of GetQuery or SetQuery
private IDBConnection conn; // database connection
private ModelRDB dbModel; // database Jena model
private String DB_URL = "jdbc:mysql://localhost/doafi"; // URL of database server
private String DB_USER = "root"; // database user id
private String DB_PASSWD = ""; // database password
private String DB = "MySQL"; // database type


/* Passes any requests received via GET to POST
 * Could cause problems because data input of GET is limited */
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws IOException, ServletException {
  this.doPost(request, response);
}

/* Posted requests are processed here
 * Parsing of the input data is attempted, on success - a GetQuery or
 * SetQuery is instantiated and dispatched to carry out the requested
 * operations. */
   public void doPost(HttpServletRequest request,
              HttpServletResponse response)
  throws IOException, ServletException {

  queryInstance = null; // make sure previous query is dead

  // empty model for responses
  Model answerModel = ModelFactory.createDefaultModel();

  response.setContentType("text/xml");
  out = response.getWriter();
  try {
    in = request.getReader();
  }
  catch (Exception e) {
    System.err.println("getReader exception");
  }

  try { // open database connection to RDF store
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    // Create database connection
    conn = new DBConnection ( DB_URL, DB_USER, DB_PASSWD, DB );
    // open the default model in the database
    dbModel = ModelRDB.open ( conn );
  }
  catch (Exception e) {
    answerModel = addError(answerModel, "Unable to connect to database - "
        + e.getMessage());
  }

  // create an empty in memory model of query
  Model model = ModelFactory.createDefaultModel();
  try {
    model.read(in, QUAFI.NS);
  }
  catch (Exception e) {
    answerModel = addError(answerModel, "Unable to parse RDF/XML");
  }
```

```
// determine whether to create GetQuery or SetQuery
if (model.contains(null, RDF.type, QUAFI.GetQuery)) {
  // model will just be queried
  Resource root = null;

  // default GetQuery parameters
  int maxItems = 10;
  int depth = 2;

  ResIterator rootIterator = model.listSubjectsWithProperty(RDF.type,
              QUAFI.GetQuery);
  if (rootIterator.hasNext()) { // no need to loop, only want 1
    root = rootIterator.nextResource();
  }

  // get depth
  Statement depthStmt = model.getProperty(root, QUAFI.depth);
  if (depthStmt != null) {
    depth = depthStmt.getInt();
    // remove from query model before passing to Query handler
    model.remove(depthStmt);
  }

  // get maxItems
  Statement maxItemsStmt = model.getProperty(root, QUAFI.maxItems);
  if (maxItemsStmt != null) {
    maxItems = maxItemsStmt.getInt();
    // remove from query model before passing to Query handler
    model.remove(maxItemsStmt);
  }

  queryInstance = new GetQuery(depth, maxItems, dbModel, model);

}
else if (model.contains(null, RDF.type, QUAFI.SetQuery)) {
  // model will be written to / modified
  Resource root = null;

  // default SetQuery parameters
  boolean deleteItems = false;
  String username = null;
  String password = null;
  String editComment = null;
  int rollbackTo = -1;

  // locate the root
  ResIterator rootIterator = model.listSubjectsWithProperty(RDF.type,
      QUAFI.SetQuery);
  if (rootIterator.hasNext()) { // no need to loop, only want 1
    root = rootIterator.nextResource();
    // remove the SetQuery wrapper from query before processing
    model = model.remove(model.createStatement(root,
          RDF.type,
          QUAFI.SetQuery));
    // but add a type of FoodItem
    model = model.add(model.createStatement(root,
          RDF.type,
          DOAFI.FoodItem));
  }

  // get deleteItems
  Statement deleteStmt = model.getProperty(root, QUAFI.deleteItems);
```

```
  if (deleteStmt != null) {
    // if there is any mention of deleteItems in query,
    // then it is a delete
    deleteItems = true;
    // remove from query model before passing to Query handler
    model.remove(deleteStmt);
  }

  // get username
  Statement usernameStmt = model.getProperty(root, QUAFI.username);
  if (usernameStmt != null) {
    username = usernameStmt.getString();
    model.remove(usernameStmt);
  }

  // get password
  Statement passwordStmt = model.getProperty(root, QUAFI.password);
  if (passwordStmt != null) {
    password = passwordStmt.getString();
    model.remove(passwordStmt);
  }

  // get editComment
  Statement editCommentStmt = model.getProperty(root, QUAFI.editComment);
  if (editCommentStmt != null) {
    editComment = editCommentStmt.getString();
    model.remove(editCommentStmt);
  }

  // get rollbackTo
  Statement rollbackToStmt = model.getProperty(root, QUAFI.rollbackTo);
  if (rollbackToStmt != null) {
    rollbackTo = rollbackToStmt.getInt();
    model.remove(rollbackToStmt);
  }

  if (username != null && password != null) {
    queryInstance = new SetQuery(deleteItems, dbModel, model,
        username, password, editComment, rollbackTo);
  }
  else {
    // error: username or password was not instantiated
  }
}
else {
  // no idea what to do with this model - not get or set
  if (answerModel.size() == 0) {
    answerModel = addError(answerModel, "Unable to determine query type");
  }
}

// if the query has been correctly instantiated
if (queryInstance != null) {
  // perform query and return
  answerModel = queryInstance.executeQuery();

  // close database connection
  try {
    conn.close();
  }
  catch (Exception e) {
    System.err.println("Error closing database connection");
```

```
      }
    }
    else {
      answerModel = addError(answerModel, "Query incorrectly instantiated");
    }

    answerModel.write(out);
  }

  /* takes a query return and adds result with status error + msg */
  public Model addError(Model answerModel, String errorMsg) {
    answerModel = answerModel.remove(answerModel);
    // will throw exception ^^ ?

    Resource answerRoot = answerModel.createResource()
      .addProperty(RDF.type, QUAFI.Result)
      .addProperty(QUAFI.status, "Error: " + errorMsg);

    return answerModel;
  }
}
```

B.1.2 QueryBase

```
package ds;

import vocabulary.*;

import java.util.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.vocabulary.*;

/**
 * A base class with common functionality, present in SetQuery and GetQuery objects. Instantiated
 * via QueryProcessor, but indirectly as the specialised query will be instantiated and make use
 * of the provided common methods.
 */

public class QueryBase  {

  StmtIterator queryStmtIterator; // iterator for statements of query, deconstructed
  Model queryModel; // in memory model of the query
  ModelRDB dbModel; // model access to database
  Model answerModel; // in memory model to return answers

  public QueryBase(ModelRDB dbModel, Model queryModel) {
    this.dbModel = dbModel;
    this.queryModel = queryModel;
  }

  /* takes a query return and adds status OK to QUAFI.Result */
  public Model addOK(Model answerModel) {
    Resource answerRoot;
    ResIterator ri = answerModel.listSubjectsWithProperty(
              RDF.type,
              QUAFI.Result);
    if (ri.hasNext()) {
      answerRoot = ri.nextResource();
      answerRoot.addProperty(QUAFI.status, "OK");
```

```
    }
    else {
      // error - there was no Result class!
    }
    return answerModel;
  }

  /* takes a query return and adds result with status error + msg */
  public Model addError(Model answerModel, String errorMsg) {
    answerModel = answerModel.remove(answerModel);
    // will throw exception ^^ ?

    Resource answerRoot = answerModel.createResource()
      .addProperty(RDF.type, QUAFI.Result)
      .addProperty(QUAFI.status, "Error: " + errorMsg);
    return answerModel;
  }

  /* get item properties (direct URI and via b-node)  from database
   * must add all items linked by bnodes directly and add others
    * to visitQueue */
  public QueryWrapper getElementByURI(Resource thisURI) {
    Vector visitQueue = new Vector(); // named nodes to visit next
    StmtIterator queryStmtIterator;
    Model URIAnswerModel = ModelFactory.createDefaultModel();
    // ugly hack requires casting of nulls for method overloading
    if (dbModel.contains(thisURI, (Property)null, (RDFNode)null)) {
      // get all the properties directly linked to this subject
      // ugly hack, casting nulls to facilitate method overloading
      queryStmtIterator = dbModel.listStatements(thisURI, (Property)null, (RDFNode)null);
      // check out properties
      while (queryStmtIterator.hasNext()) {
        Statement thisQueryStatement = queryStmtIterator.nextStatement();

        try { // to access it as a resource
          Resource thisObject = thisQueryStatement.getResource();
          if (thisObject.isAnon()) { // is a bNode - explore!
            // get statements which this object as subject
            // via recursive call + add to answer
            QueryWrapper qw = getElementByURI(thisObject);
            visitQueue.addAll(qw.getVisitQueue());
            URIAnswerModel.add(qw.getAnswerModel());
          }
          // add this statement to the model
          // also add to some sort of agenda?
          URIAnswerModel.add(thisQueryStatement);
          visitQueue.add(thisObject);
        }
        // if object is a literal, add it to the answerModel
        catch (ResourceRequiredException e) { // it was a literal
          URIAnswerModel.add(thisQueryStatement);
        }
      }
    }
    else {
      // error: not in model
    }
    return new QueryWrapper(visitQueue, URIAnswerModel);
  }
}
```

B.1.3 QueryInterface

```
package ds;

import com.hp.hpl.jena.rdf.model.*;

/** An interface to use for GetQuery and SetQuery to enforce the use of executeQuery */

public interface QueryInterface {
  public Model executeQuery();
}
```

B.1.4 GetQuery

```
package ds;

import vocabulary.*;

import java.util.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.rdql.*;
import com.hp.hpl.jena.vocabulary.*;

/**
 *  Query for reading from the database. Instantiated via QueryProcessor, and requiring no authorisation.
 *  executeQuery() is called and returns an answer, either the items requested with an OK status, or an
 *  appropriate error.
 **/

public class GetQuery extends QueryBase implements QueryInterface {

  Hashtable subjectMap; // maps subject nodes to variable names for RDQL

  int depth; // query depth
  int maxItems; // query maxItems field
  String queryString; // to be built up with RDQL query

  public GetQuery(int depth, int maxItems, ModelRDB dbModel, Model queryModel) {
    super(dbModel, queryModel);
    if (depth <=6) {
      this.depth = depth;
    }
    else {
      this.depth = 6;
    }
    if (maxItems <= 10) { // set upper limit on maxItems to 10 to limit branching
      this.maxItems = maxItems;
    }
    else {
      this.maxItems = 10;
    }
    queryString = "";
  }

  /* Used to determine the answer to queries where an explict URI for an item is not
   * requested. Instead, properties are provided and the server attempts to find matches
   * to that query using RDQL to the data store. */
  public QueryWrapper getElementsByProperty(Model db, Model prop, Model answerModel) {
```

```
Vector visitQueue = new Vector(); // items to visit next, ultimately we're doing DFS
Resource answerRoot = answerModel.createResource()
  .addProperty(RDF.type, QUAFI.Result);
Bag answerBag = answerModel.createBag();
answerRoot.addProperty(QUAFI.itemList, answerBag);

// begin building up RDQL queryString
queryString = "SELECT ?subject WHERE \n" ;

StmtIterator queryToRemove = prop.listStatements((Resource)null,
          RDF.type,
          QUAFI.GetQuery);

prop = prop.remove(queryToRemove);

// build queryString from queryModel
queryStmtIterator = prop.listStatements();
subjectMap = new Hashtable();
while (queryStmtIterator.hasNext()) {
  String queryPredicate;
  String queryObject;
  Resource objectResource;
  String querySubject = "";
  Resource subjectResource;

  Statement thisQueryStatement = queryStmtIterator.nextStatement();

  // subject
  subjectResource = thisQueryStatement.getSubject();

  // if it is a blank node:
  if (subjectResource.isAnon()) {
    if (subjectMap.containsKey(subjectResource)) {
      // subject of current statement has been seen before
      querySubject = (String)subjectMap.get(subjectResource);
    }
    else { // the subject hasn't been seen before
      // check - is it the root?
      // s-p-o: if this subject is an object in this model,
      // it isn't the root
      if (prop.contains(null, null, subjectResource)) {
        // generate a name for the bNode
        // it might be messy, but get the bNode ID
        // use substring to get first 6 characters
        // include an 'a' at the beginning of variable
        // name to ensure char
        querySubject = "?a" + subjectResource.toString()
          .substring(0, 6);

      }
      else { // this is the root
        querySubject = "?subject";
      }
      subjectMap.put(subjectResource, querySubject);
    }
  }
  else { // it will be a URI - subjects are not literals
    try {
      querySubject = "<" + subjectResource.getURI() + ">";
    }
    catch (ResourceRequiredException e) {
      // error: this was not a URI as expected
```

```
      }
    }

    // predicate
    queryPredicate = "<" + thisQueryStatement.getPredicate().toString() + ">";

    // object
    try { // to see if object is a resource (URI or bNode)
      // get resource value (URI)
      objectResource = thisQueryStatement.getResource();
      if (objectResource.isAnon()) {
        // deal with a bNode
        queryObject = "?a" + objectResource.toString().substring(0, 6);
        // any need to verify from the queryMap?
      }
      else { // it is a URI
        queryObject = "<"+thisQueryStatement.getResource().getURI()+">";
      }
    }
    // object was not a resource, it was a literal
    catch (ResourceRequiredException e) {
      // get the literal value (eg String)
      queryObject = "\"" + thisQueryStatement.getString() + "\"";
    }

    // concatenate predicates/objects on to queryString
    // if this statement is at depth 1 - we are trying to find this subject
    queryString += "("+querySubject+", "+queryPredicate+", "+queryObject+") \n";
    // else if this statement is greater than 1 level deep + has bnode as subject
  }
  Query query = new Query(queryString) ;

  // Need to set the source as the RDQL query does not.
  query.setSource(dbModel);
  QueryExecution qe = new QueryEngine(query) ;

  QueryResults results = qe.exec();
  Iterator iter = results;
  // loop through statements using iterator
  // stop at maxItems
  for (int i = 0; iter.hasNext() && i < this.maxItems; i++) {
    ResultBinding res = (ResultBinding)iter.next();
    Resource querySubject = (Resource)res.get("subject");
    //queryAnswer += "querySubject = " + querySubject;
    answerBag.add(querySubject);
    visitQueue.add(querySubject);
  }
  results.close();

  return new QueryWrapper(visitQueue, answerModel);
}

/* Wrapper for the query, called by executeQuery to determine which action to take,
 * whether to search by URI or by properties */
public Model getQuery(Model answer, Model query, Model dbModel) {
  dbModel.enterCriticalSection(true); // gain read lock

  Vector visitQueue = new Vector(); // queue for URIs to visit
  QueryWrapper qw = null;

  // determine if need to call getElementsByProperty
  if (query.size() == 1 && query.contains((Resource)null, RDF.type, (RDFNode)null)) {
```

```
      // URI to get is the subject of this statement
      ResIterator r = query.listSubjects();
      Resource thisURI = r.nextResource();
      qw = getElementByURI(thisURI);
    }
    else { // item request isn't just for a URI
      qw = getElementsByProperty(dbModel, query, answer);
    }
    answer.add(qw.getAnswerModel());
    visitQueue.addAll(qw.getVisitQueue());

    // loop through queue until empty or depth limit is reached
    for (int i = 0; i < this.depth; i++) {
      int currentQueueLength = visitQueue.size();
      // call getElementByURI on each URI in queue
      for (int j = 0; i < currentQueueLength && !visitQueue.isEmpty(); j++) {
        // remove subject from front of queue (Vector(0))
        System.out.println(visitQueue.toString());
        Resource thisSubject = (Resource)visitQueue.remove(0);
        qw = getElementByURI(thisSubject);
        // add queued items from this URI to our queue
        visitQueue.addAll(qw.getVisitQueue());
        // add statements from this URI query to our model
        answer.add(qw.getAnswerModel());
      }
    }
    answer = addOK(answer);
    dbModel.leaveCriticalSection(); // get rid of db lock

    return answer;
  }

  /* Obligatory to implement this method due to QueryInterface. This begins the execution
   * process and returns an answerModel answer. */
  public Model executeQuery() {
    // create empty in memory model to build up answer to query
    answerModel = ModelFactory.createDefaultModel();
    answerModel.setNsPrefix("quafi", QUAFI.NS);
    answerModel.setNsPrefix("doafi", DOAFI.NS);

    answerModel = getQuery(answerModel, queryModel, dbModel);

    return answerModel;
  }
}
```

B.1.5 SetQuery

```
package ds;

import vocabulary.*;

import java.util.*;
import java.util.regex.*;
import java.sql.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.rdql.*;
import com.hp.hpl.jena.vocabulary.*;
```

```
/**
 *  Query for writing to the database including add, edit, delete and rollback queries. This is
 *  called by QueryProcessor and has an executeQuery method which returns the answer to
 *  QueryProcessor for return to the user.
 *  */

public class SetQuery extends QueryBase implements QueryInterface {

  StmtIterator queryStmtIterator; // iterator for statements of query, deconstructed
  boolean deleteItems; // is this a delete query? flag
  String username; // quafi:username
  String password; // quafi:password
  String editComment; // quafi:editComment
  int authorisationLevel; // from database, determines how important user is
  int rollbackTo; // quafi:rollbackTo

  String DB_URL = "jdbc:mysql://localhost/doafi";  // URL of database server
  String DB_USER = "root";      // database user id
  String DB_PASSWD = "";       // database password
  String DB = "MySQL";          // database type

  Connection conn; // database connection for user table access

  public SetQuery(boolean deleteItems, ModelRDB dbModel, Model queryModel,
      String username, String password, String editComment,
      int rollbackTo) {
    super(dbModel, queryModel);
    this.deleteItems = deleteItems;
    this.username = username;
    this.password = password;
    this.editComment = editComment;
    this.rollbackTo = rollbackTo;
  }

  /* Called to determine the authentication level of a user, if authentication fails
   * then -1 is returned. */
  public int authenticateUser(String username, String password) {
    String sql;
    try { // connect to db
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      // Create database connection
      conn = DriverManager.getConnection ( DB_URL, DB_USER, DB_PASSWD );
    }
    catch (Exception e) {
      System.err.println("authenticateUser: error connecting to db");
    }

    sql = "SELECT authorisation FROM users WHERE username = '"
      + username + "' AND password = '" + password + "'";

       try {
      java.sql.Statement s = conn.createStatement(
          java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE,
          java.sql.ResultSet.CONCUR_UPDATABLE);
      java.sql.ResultSet rs = s.executeQuery(sql);

      if (rs.first()) { // open first row in resultset
        // return value of 1st column - authorisation level
        int authorisationLevel = rs.getInt(1);
        conn.close();
        return authorisationLevel;
      }
```

```
    else {
      System.err.println("authenticateUser: no matching username"
          + " /password\n" + sql);
    }
    conn.close();
  }
  catch (Exception e) {
    System.err.println("authenticateUser: error executing query\n" +
        sql + "\n" + e.getMessage());
  }
  return -1; // fail to authenticate user

}

/* Wrapper for query operations which determines which operation to
 * perform based on the supplied parameters */
public Model setQuery(Model answer, Model queryModel, ModelRDB dbModel,
    boolean deleteItems) {
  Resource setURI = null; // base URI of item requiring modification
  boolean noErrors; // flag to indicate whether errors were found on the way
  dbModel.enterCriticalSection(false); // get write lock
  dbModel.begin(); // begin database transaction

  // get the URI
  queryStmtIterator = queryModel.listStatements();
  // continue to look through statements until setURI is instantiated
  while (queryStmtIterator.hasNext() && setURI == null) {
    Resource subjectResource;
    com.hp.hpl.jena.rdf.model.Statement thisQueryStatement =
      queryStmtIterator.nextStatement();

    subjectResource = thisQueryStatement.getSubject();
    // if this subject is not a object in the model
    // AND it is a bNode - likely to be the root
    if (!queryModel.contains(null, null, subjectResource)) {
      setURI = subjectResource; // root URI - FoodItem to modify
    }

  }

  // if a URI was actually given
  if (setURI != null) {
    // user requested a rollback
    if (rollbackTo >= 0) {
      noErrors = rollback(rollbackTo, dbModel, setURI);
    }
    // if it is already in the dbModel
    else if (dbModel.contains(setURI,
          RDF.type,
      DOAFI.FoodItem)) {

      if (deleteItems) { // query is to delete
        noErrors = setDelete(queryModel, dbModel, setURI);
      }
      else { // query is to edit
        noErrors = setEdit(queryModel, dbModel, setURI);
      }
    }
    else { // it isn't in the dbModel - add it
      noErrors = setAdd(queryModel, dbModel, setURI, deleteItems);
    }
  }
```

```
  else {
    // error: URI was null, can't do anything
    noErrors = false;
  }
  if (noErrors) {
    answer = addOK(answer);
    dbModel.commit(); // commit db transaction
  }
  else { // there was an error at some point
    answer = addError(answer, "Error using set command");
    dbModel.abort(); // abort db transaction
  }
  dbModel.leaveCriticalSection();
  return answer;
}

/* deals with additions to the database (ie new entries)
 * adds entry to database and then records entry metadata in modelHistory */
public boolean setAdd(Model queryModel, Model dbModel, Resource setURI,
    boolean deleteItems) {
  if (deleteItems) { // can't delete something which doesn't exist
    return false;
  }
  Model newerItems = ModelFactory.createDefaultModel();
  // it is new - try and add it to the dbModel
  dbModel.add(queryModel);

  // create blank node to wrap changes+parameters in
  Resource changesRoot = newerItems.createResource();

  newerItems = addChangeMetadata(newerItems, username, editComment,
      changesRoot);

  // need tomake new seq because FoodItem is new
  Seq historySeq = dbModel.createSeq();

  dbModel.add(newerItems);

  // add newItems to seq (at end)
  historySeq = (Seq)historySeq.add(changesRoot);

  // add the modelHistory to the dbModel
  // create History class

  Resource addHistory = dbModel.createResource(setURI + "/History");

  dbModel.add(dbModel.createStatement(setURI,
          DOAFI.history,
          addHistory));

  dbModel.add(dbModel.createStatement(addHistory,
          DOAFI.modelHistory,
          historySeq));

  return true;
}

/* Used to edit an existing entry in the database, makes modifications
 * and then stores a record of the previous entries in modelHistory */
public boolean setEdit(Model queryModel, Model dbModel, Resource setURI) {
  Model newItems;
  Model newerItems = ModelFactory.createDefaultModel();
```

```
  Model revertChanges = ModelFactory.createDefaultModel();
  // ADD / EDIT
  // take the diff of db and query models
  newItems = realDifference(queryModel, dbModel);

  // modify dbmodel
  // iterate through newItems
  StmtIterator newIterator = newItems.listStatements();
  while (newIterator.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement s =
      newIterator.nextStatement();
    // WHAT IF THERE IS MORE THAN ONE S-P combination?
    if (dbModel.contains(s.getSubject(), s.getPredicate())) {
      // get the old version of this statement
      com.hp.hpl.jena.rdf.model.Statement oldS =
        dbModel.getProperty(s.getSubject(),
            s.getPredicate());

      // store it in revertChanges
      revertChanges = revertChanges.add(oldS);

      // delete the old one from the dbModel
      dbModel.remove(oldS);
    }
    else { // item is new
      // add delete statement for this subject,predicate combo
       // denotes that this statement is new+should be removed
      // if rolling back to this iteration
    }
    // add the new one
    dbModel.add(s);
      }

  // get seq of modelHistory
  Seq historySeq = dbModel.getProperty(
    dbModel.getProperty(setURI, DOAFI.history).getResource(),
    DOAFI.modelHistory)
    .getSeq();

  // add Change/i/ to concrete URIs
  // +1 reflects next seq
  newerItems = addChangeToURI(revertChanges, historySeq.size()+1);

  // create blank node to wrap changes+parameters in
  Resource changesRoot = dbModel.createResource();

  newerItems = addChangeMetadata(newerItems, username, editComment,
      changesRoot);

  // add newItems to seq (at end)
  historySeq = (Seq)historySeq.add(changesRoot);

  // need to add this back into the dbModel?
  dbModel.add(newerItems);

  return true;
}

/* Used to delete specified items from the database. It is required to
* determine dependencies in order to do this because it is necessary to
* provide context in order to delete an entry, but if other entries rely
* on that context, you don't want to delete it */
```

```
public boolean setDelete(Model queryModel, Model dbModel, Resource setURI) {
  Model newItems;
  Model temp = ModelFactory.createDefaultModel();
  Model newerItems = ModelFactory.createDefaultModel();
  Model revertChanges = ModelFactory.createDefaultModel();
  // DELETE
  // take the intersection of db and query models
  // to determine which items to delete
  // ie discard request to delete items which aren't present
  newItems = queryModel.intersection(dbModel);

  // iterate through newItems to determine which not to delete
  StmtIterator newIterator = newItems.listStatements();
  while (newIterator.hasNext()) {
    temp = ModelFactory.createDefaultModel(); // empty temporary model
    com.hp.hpl.jena.rdf.model.Statement s =
      newIterator.nextStatement();

    // rdf:type nodes are shared by others
    if (s.getPredicate().equals(RDF.type)) {
      // dealing with rdf:type nodes first
      StmtIterator innerIterator =
        dbModel.listStatements(s.getSubject(),
          (Property)null, (RDFNode)null);
      temp = temp.add(innerIterator);
      if (temp.difference(newItems).size() > 0) {
        // if some statements in the dbModel share share
        // this subject as theirs, not being deleted
        // then delete from deleteItems to avoid
        // deletion from dbModel
        newIterator.remove();
      }
      else { // it is going to be deleted
        // so get old version to store in modelHistory
               // store it in revertChanges
        revertChanges = revertChanges.add(dbModel.getProperty(
            s.getSubject(),s.getPredicate()));
      }
    }
    else {
      // remove from underlying representation
      try {
        Resource r = s.getResource();
        StmtIterator innerIterator = dbModel.listStatements(r,
              (Property)null, (RDFNode)null);

        temp = temp.add(innerIterator);
        if (temp.difference(newItems).size() > 0) {
          // object is subject of some statement not
          // being deleted
          newIterator.remove();
        }
        else { // item is not subject of some statement not
          // being deleted
          revertChanges = revertChanges.add(
              dbModel.getProperty(
              s.getSubject(),
              s.getPredicate()));
        }
      }
      catch (Exception e) {
        // object was literal, fine to delete
```

```
        revertChanges = revertChanges.add(dbModel.getProperty(
            s.getSubject(),s.getPredicate()));
      }
    }
  }
  dbModel.remove(newItems);
  // get seq of modelHistory
  Seq historySeq = dbModel.getProperty(
    dbModel.getProperty(setURI, DOAFI.history).getResource(),
    DOAFI.modelHistory)
    .getSeq();

  // add Change/i/ to concrete URIs
  // +1 reflects next seq
  newerItems = addChangeToURI(revertChanges, historySeq.size()+1);


  // create blank node to wrap changes+parameters in
  Resource changesRoot = dbModel.createResource();

  newerItems = addChangeMetadata(newerItems, username, editComment, changesRoot);

  historySeq = (Seq)historySeq.add(changesRoot);

  // need to add this back into the dbModel?
  dbModel.add(newerItems);

  return true;
}

/* The difference() method provided in Jena.Model is not great, it doesn't recognise b-nodes
 * as blank nbecause they are given unique identifiers, so I implement a method to do this */
public Model realDifference(Model queryModel, Model dbModel) {
  Model newItems = queryModel.difference(dbModel);
  // don't trust the diff - bnodes don't work
  // modify bnodes to be same as in dbModel - if they are in dbModel
  ResIterator theseSubjects = newItems.listSubjects();
  while (theseSubjects.hasNext()) {
    // begin at roots of possibly mislabelled new items
    // truly new items won't be anon at base
    try {
      Resource thisSubject = theseSubjects.nextResource();
      if (!thisSubject.isAnon()) {
        newItems = modifyBNodes(thisSubject, newItems, dbModel);
      }
    }
    catch (Exception e) {
      // subject has changed / broken - continue
    }
  }
  // now with modified bNodes, take new diff
  newItems = newItems.difference(dbModel);

  return newItems;
}

/* take a model newerItems just containing changes and add metadata */
public Model addChangeMetadata(Model newerItems, String username,
    String editComment, Resource changesRoot) {

  // add changes to changesRoot
  ResIterator changes = newerItems.listSubjects();
```

```
  while (changes.hasNext()) {
    Resource r = changes.nextResource();
    // if r isn't a bnode then it needs to be indicated
    // that this subject points to a change
    if (!newerItems.contains((Resource)null, (Property)null, r)) {
      newerItems.add(newerItems.createStatement(changesRoot,
          DOAFI.changes,
          r));
    }
  }
  // add username
  newerItems = newerItems.add(newerItems.createStatement(changesRoot,
            DOAFI.username,
          newerItems.createLiteral(username)));
  // add timestamp
  java.text.SimpleDateFormat sdf = new
  java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  newerItems = newerItems.add(newerItems.createStatement(changesRoot,
          DOAFI.timestamp,
          newerItems.createLiteral(
                sdf.format(new java.util.Date(
          System.currentTimeMillis()
          )))));
         // add editComment
  newerItems = newerItems.add(newerItems.createStatement(changesRoot,
          DOAFI.editComment,
          newerItems.createLiteral(editComment)));
  // add RDF type of ModelHistoryItem
  newerItems = newerItems.add(newerItems.createStatement(changesRoot,
          RDF.type,
          DOAFI.ModelHistoryItem));
  return newerItems;
}

/* rollback changes to change indicated by i, modify db to reflect these changes
* and store the previous values in changeHistory */
public boolean rollback(int rollbackTo, ModelRDB dbModel, Resource setURI) {
  int historyLeft;
  // start a new diff model
  Model newDiffs = ModelFactory.createDefaultModel();

  //get the modelHistory of this item
  Seq historySeq = dbModel.getProperty(
    dbModel.getProperty(setURI, DOAFI.history).getResource(),
    DOAFI.modelHistory)
        .getSeq();
  // look through from historySeq.size (upper limit) down to
  // rollbackTo
  for (int i = historySeq.size(); i > rollbackTo; i--) {
    // rolling back each iteration
    Resource thisHistory = historySeq.getResource(i);
    // start a new model
    Model changesToBeMade = ModelFactory.createDefaultModel();
    // get all subjects of the changes in this seq
    StmtIterator objectIterator = dbModel.listStatements(
        thisHistory, DOAFI.changes, (RDFNode)null);

    // add all statements to be modified to changesToBeMade
    // have to do BFS of model from thisHistory to find them all
    while (objectIterator.hasNext()) {
      Vector visitQueue = new Vector(); // items to visit
      // get all statements in the model matching one subject
```

```
    // and add them to changesToBeMade
    Resource subject = null;
    try {
      subject = objectIterator.nextStatement()
        .getResource();
    }
    catch (Exception e) {
      // subject was literal-not really possible
    }
    StmtIterator si = dbModel.listStatements(subject,
        (Property)null, (RDFNode)null);
    changesToBeMade.add(si);
    while (si.hasNext()) {
      com.hp.hpl.jena.rdf.model.Statement bfsStatement =
        si.nextStatement();
      try {
        visitQueue.add(bfsStatement.getResource());
      }
      catch (Exception e) {
        // item was a literal
      }
    }
    while (!visitQueue.isEmpty()) {
      // remove last element of Vector
      Resource subjectURI =
        (Resource)visitQueue.remove(visitQueue.size());
      QueryWrapper innerQuery = getElementByURI(subjectURI);
      visitQueue.add(innerQuery.getVisitQueue());
      changesToBeMade.add(innerQuery.getAnswerModel());
    }
  }

  // reverting the URIs from Change/0/.. to /
  changesToBeMade = removeChangeFromURI(changesToBeMade, i);

  // create diff  between current dbModel and these changess
  Model thisDiff = realDifference(changesToBeMade, dbModel);

  StmtIterator newIterator = changesToBeMade.listStatements();
  while (newIterator.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement s =
      newIterator.nextStatement();
    if (dbModel.contains(s.getSubject(), s.getPredicate())) {
      // get the old version of this statement
      com.hp.hpl.jena.rdf.model.Statement oldS =
        dbModel.getProperty(s.getSubject(),
            s.getPredicate());

      // delete the old one from the orig
      dbModel.remove(oldS);
      newDiffs.add(oldS);
    }
    // add the new one
    dbModel.add(s);
    // record deletions?
  }
}
// create blank node to wrap changes+parameters in
Resource changesRoot = dbModel.createResource();

// add changes to URIs before storing here?
newDiffs = addChangeToURI(newDiffs, historySeq.size()+1);
```

```
  // add username, editComment, timestamp and newDiffs to some bnode
  newDiffs = addChangeMetadata(newDiffs, username, editComment, changesRoot);

  // add the bnode to the rollback history
  historySeq = (Seq)historySeq.add(changesRoot);

  dbModel.add(newDiffs);

  return true;
}

/* called by realDifference to replace BNodes with those in the dbModel
 * such that items can be compared using Jena set operations */
public Model modifyBNodes(Resource toReplace, Model m, Model dbModel) {
  Model newItems = ModelFactory.createDefaultModel();
  StmtIterator st = m.listStatements(toReplace, (Property)null, (RDFNode)null);
  while (st.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement thisSt = st.nextStatement();
    try {
      Resource oldObj = thisSt.getResource();
      if (oldObj.isAnon() &&
        dbModel.contains(thisSt.getSubject(),
        thisSt.getPredicate())) { // might need to be replaced

        Resource newObj = null;
        // want to find value for newObj
        StmtIterator st1 = dbModel.listStatements(
            thisSt.getSubject(),
            thisSt.getPredicate(),
            (RDFNode)null);

        if (st1.hasNext()) { // assume there is only 1
          newObj = st1.nextStatement().getResource();
          // if it literal,will throw exception - fine
        }
        if (newObj.isAnon()) {
          // if it isn't, don't our statements are new
          StmtIterator st2 = m.listStatements(
              oldObj, (Property)null,
              (RDFNode)null);

          //thisSt.changeObject(newObj);
          newItems.add(newItems.createStatement(
                thisSt.getSubject(),
                thisSt.getPredicate(),
                newObj));

          while (st2.hasNext()) {
            com.hp.hpl.jena.rdf.model.Statement innerSt =
                  st2.nextStatement();
            // add statement to model with newObj as subj
            m = m.add(m.createStatement(newObj,
                  innerSt.getPredicate(),
                  innerSt.getObject()));
            // remove old statement with oldObj as subj
            st2.remove();
          }
          // continue to replace bNodes with DFS
          m = modifyBNodes(newObj, m, dbModel);
        }
        else {
```

```
          // this statement is different from dbModel
        }

      }
      else {
        // this statement is different from dbModel
      }
    }
    catch (Exception e) {
      System.err.println(thisSt + "\n" + e.getMessage());
      // oldObj was a literal
      // or
      // newObj was a literal
    }
  }
  m = modifyStatements(m, newItems);
  return m;
}

/* Used for replacing existing statements, or adding new ones. If a statement
 * already exists in the model with this subject,predicate - this method will
 * overwrite it. */
public Model modifyStatements(Model orig, Model changes) {
  StmtIterator newIterator = changes.listStatements();
  while (newIterator.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement s =
      newIterator.nextStatement();
    if (orig.contains(s.getSubject(), s.getPredicate())) {
      // get the old version of this statement
      com.hp.hpl.jena.rdf.model.Statement oldS =
        orig.getProperty(s.getSubject(),
            s.getPredicate());

      // delete the old one from the orig
      orig.remove(oldS);
    }
    // add the new one
    orig = orig.add(s);
       }
  return orig;
}

/* Modifies URIs to contain /Change/i/ where i is the supplied integer
 * Used for storing changes in modelHistory */
public Model addChangeToURI(Model m, int i) {
  StmtIterator st = m.listStatements();
  Model newM = ModelFactory.createDefaultModel();
  Pattern doafiBase = Pattern.compile("http://doafi.4angle.com/");

  while (st.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement s =
      st.nextStatement();
    Resource newSubject = s.getSubject();
    // if the subject is concrete
    if (!newSubject.isAnon()) {
      // replace oldURI with a new one using regex
      String oldURI = newSubject.getURI();
      String newURI = doafiBase.matcher(oldURI)
        .replaceFirst("http://doafi.4angle.com/Change/" + i + "/");
      newSubject = newM.createResource(newURI);
        }
    try {
```

```
      Resource newObject = s.getResource();
      if (!newObject.isAnon()) {
        // replace oldURI with a new one using regex
        String oldURI = newObject.getURI();
        String newURI = doafiBase.matcher(oldURI)
          .replaceFirst("http://doafi.4angle.com/Change/"
              + i + "/");
        newObject = newM.createResource(newURI);

        newM.add(newM.createStatement(newSubject,
          s.getPredicate(), newObject));
      }
    }
    catch (Exception e) {
      // object was not a resource
    }
    // if the object is concrete
    newM.add(newM.createStatement(newSubject, s.getPredicate(),
          s.getObject()));
  }
  return newM;
}

/* Removes Change.. from URIs as added by addChangeToURI */
public Model removeChangeFromURI(Model m, int i) {
  StmtIterator st = m.listStatements();
  Pattern p = Pattern.compile("http://doafi.4angle.com/Change/" + i + "/");
  Model newM = ModelFactory.createDefaultModel();

  while (st.hasNext()) {
    com.hp.hpl.jena.rdf.model.Statement s =
      st.nextStatement();
    Resource newSubject = s.getSubject();
    // if the subject is concrete
    if (!newSubject.isAnon()) {
      // replace oldURI with a new one using regex
      String oldURI = newSubject.getURI();
      String newURI = p.matcher(oldURI)
        .replaceFirst("http://doafi.4angle.com/");
      newSubject = newM.createResource(newURI);
    }
    try {
      Resource newObject = s.getResource();
      // if the object is concrete
      if (!newObject.isAnon()) {
        // replace oldURI with a new one using regex
        String oldURI = newObject.getURI();
        String newURI = p.matcher(oldURI)
          .replaceFirst("http://doafi.4angle.com/");
        newObject = newM.createResource(newURI);

        newM.add(newM.createStatement(newSubject,
            s.getPredicate(), newObject));
      }
    }
    catch (Exception e) {
      // object was literal
    }
    newM.add(newM.createStatement(newSubject, s.getPredicate(),
          s.getObject()));
  }
  return newM;
```

```
  }

  /* Method begins the execution of this query, delegates responsibility and
   * ultimately returns the answerModel to the client */
  public Model executeQuery() {
    // create empty in memory model to build up answer to query
    answerModel = ModelFactory.createDefaultModel();
    answerModel.setNsPrefix("quafi", QUAFI.NS);
    answerModel.setNsPrefix("doafi", DOAFI.NS);
    Resource answerRoot = answerModel.createResource()
        .addProperty(RDF.type, QUAFI.Result);

    // check if user exists for modifying model
    authorisationLevel = authenticateUser(username, password);

    if (authorisationLevel >= 0) {
      answerModel = setQuery(answerModel, queryModel, dbModel, deleteItems);
    }
    else {
      // error: not authenticated
      System.err.println("not authenticated");
      answerModel = addError(answerModel,
        "You are not authorised to make this modification");
    }
    return answerModel;
  }
}
```

B.1.6 QueryWrapper

```
package ds;

import java.util.*;
import com.hp.hpl.jena.rdf.model.*;

/**
 * Used to wrap responses from some queries which require the use of a visitQueue and an answerModel
 * */

public class QueryWrapper {

  private Vector visitQueue;
  private Model answerModel;
  private Model dbModel;
  private Model oldAnswer;

  // get query wrapper
  public QueryWrapper(Vector v, Model m) {
    this.visitQueue = v;
    this.answerModel = m;
  }

  // set query wrapper
  public QueryWrapper(Model m, Model a) {
    this.answerModel = m;
    this.oldAnswer = a;
  }

  public Vector getVisitQueue() {
    return visitQueue;
```

```
  }

  public Model getAnswerModel() {
    return answerModel;
  }

  public Model getOldAnswers() {
    return oldAnswer;
  }
}
```

B.2 Vocabulary package

B.2.1 DOAFI

```
package vocabulary;

import com.hp.hpl.jena.rdf.model.*;

/**
 * Wrapper for the DOAFI vocabulary, for easy reference in the ds package.
 * Generated using an RDF Schema.
 * Vocabulary definitions from prototype6.rdfs
 * @author Auto-generated by schemagen on 25 Apr 2005 11:12
 */
public class DOAFI {
    /** <p>The RDF model that holds the vocabulary terms</p> */
    private static Model m_model = ModelFactory.createDefaultModel();

    /** <p>The namespace of the vocabalary as a string ({@value})</p> */
    public static final String NS = "http://doafi.4angle.com/elements/s-0.1/";

    /** <p>The namespace of the vocabalary as a string</p>
     *  @see #NS */
    public static String getURI() {return NS;}

    /** <p>The namespace of the vocabalary as a resource</p> */
    public static final Resource NAMESPACE = m_model.createResource( NS );

    /** <p>Reason for.. relationship, modification etc.</p> */
    public static final Property reason = m_model.createProperty(
                    "http://doafi.4angle.com/elements/s-0.1/reason" );

    /** <p>Date of issue of a certificate</p> */
    public static final Property issueDate = m_model.createProperty(
                    "http://doafi.4angle.com/elements/s-0.1/issueDate" );

    /** <p>Detail about how this FoodItem was produced</p> */
    public static final Property process = m_model.createProperty(
                    "http://doafi.4angle.com/elements/s-0.1/process" );

    /** <p>Indicates the URI of certification details</p> */
    public static final Property certification = m_model.createProperty(
                    "http://doafi.4angle.com/elements/s-0.1/certification" );

    /** <p>Pointer to URI of History of this FoodItem</p> */
    public static final Property history = m_model.createProperty(
```

```
                "http://doafi.4angle.com/elements/s-0.1/history" );

/** <p>Literal title for an Object, such as FoodItem or Certificate</p> */
public static final Property title = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/title" );

/** <p>Pointer to URI of Biography of this FoodItem</p> */
public static final Property biography = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/biography" );

/** <p>Pointer to Ingredients of this FoodItem</p> */
public static final Property ingredients = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/ingredients" );

/** <p>Literal name for an object, such as a Producer, not a FoodItem</p> */
public static final Property name = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/name" );

/** <p>Timestamp of modification</p> */
public static final Property timestamp = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/timestamp" );

/** <p>Pointer to sequence of Ingredients</p> */
public static final Property ingredientList = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/ingredientList" );

/** <p>Pointer to URI of Produced of this FoodItem</p> */
public static final Property produced = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/produced" );

/** <p>The object of this property is an ingredient of the parent FoodItem</p> */
public static final Property ingredient = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/ingredient" );

/** <p>This property indicates the alternatives to a FoodItem</p> */
public static final Property alternatives = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/alternatives" );

/** <p>Pointer to Producer of who Produced this FoodItem</p> */
public static final Property producer = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/producer" );

/** <p>Wrapper for bag of alternative FoodItems</p> */
public static final Property alternativeList = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/alternativeList" );

/** <p>Detail of who issued the Certificate</p> */
public static final Property issuedBy = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/issuedBy" );

/** <p>Defines the type of Certificate issued in this Certification</p> */
public static final Property type = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/type" );

/** <p>DOAFI username, as recorded when making model modifications</p> */
public static final Property username = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/username" );

/** <p>Web address with more information about this entity</p> */
public static final Property webAddress = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/webAddress" );
```

```
/** <p>Comment relating to each edit of the FoodItem</p> */
public static final Property editComment = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/editComment" );

/** <p>Pointer to sequence of history changes to this FoodItem</p> */
public static final Property modelHistory = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/modelHistory" );

/** <p>Location at which something happened</p> */
public static final Property location = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/location" );

/** <p>Concrete pointer to the FoodItem which is an alternative to this</p> */
public static final Property alternative = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/alternative" );

/** <p>Date indicating when FoodItem should b e consumed by</p> */
public static final Property useBefore = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/useBefore" );

/** <p>Defines the unit of amount of the item being described</p> */
public static final Property amountUnit = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/amountUnit" );

/** <p>Amount of a FoodItem being described as a literal value</p> */
public static final Property amount = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/amount" );

/** <p>Pointer to the changes which took place at each modification</p> */
public static final Property changes = m_model.createProperty(
                "http://doafi.4angle.com/elements/s-0.1/changes" );

/** <p>Defines the producer of the specified FoodItem</p> */
public static final Resource Producer = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/Producer" );

public static final Resource History = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/History" );

/** <p>Wrapper class for change metadata to a FoodItem in the database</p> */
public static final Resource ModelHistoryItem = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/ModelHistoryItem" );

/** <p>Wrapper for list of alternative FoodItems</p> */
public static final Resource Alternatives = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/Alternatives" );

/** <p>A certificate is used to assert a property about a FoodItem</p> */
public static final Resource Certificate = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/Certificate" );

/** <p>Wrapper for list of certificates associated with this FoodItem</p> */
public static final Resource Certification = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/Certification" );

/** <p>Container class for list of ingredients</p> */
public static final Resource Ingredients = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/Ingredients" );

/** <p>Wraps a FoodItem as an ingredient with ingredient metadata</p> */
public static final Resource IngredientItem = m_model.createResource(
                "http://doafi.4angle.com/elements/s-0.1/IngredientItem" );
```

```
    /** <p>A product which could be used instead of this product</p> */
    public static final Resource AlternativeItem = m_model.createResource(
                    "http://doafi.4angle.com/elements/s-0.1/AlternativeItem" );

    /** <p>Information about the production of a FoodItem</p> */
    public static final Resource Produced = m_model.createResource(
                    "http://doafi.4angle.com/elements/s-0.1/Produced" );

    /** <p>Was new item at this modification sequence, when reverting it should be deleted</p> */
    public static final Resource DeleteOnRevert = m_model.createResource(
                    "http://doafi.4angle.com/elements/s-0.1/DeleteOnRevert" );

    /** <p>Biographical detail about a FoodItem</p> */
    public static final Resource Biography = m_model.createResource(
                    "http://doafi.4angle.com/elements/s-0.1/Biography" );

    /** <p>Wrapper for all of the properties of a food item</p> */
    public static final Resource FoodItem = m_model.createResource(
                    "http://doafi.4angle.com/elements/s-0.1/FoodItem" );
}
```

B.2.2 QUAFI

```
package vocabulary;
import com.hp.hpl.jena.rdf.model.*;

/**
 * Wrapper for the DOAFI vocabulary, for easy reference in the ds package.
 * Generated using an RDF Schema.
 * @author Auto-generated by schemagen on 17 Apr 2005 17:43
 */
public class QUAFI {
    /** <p>The RDF model that holds the vocabulary terms</p> */
    private static Model m_model = ModelFactory.createDefaultModel();

    /** <p>The namespace of the vocabalary as a string ({@value})</p> */
    public static final String NS = "http://doafi.4angle.com/elements/q-0.1/";

    /** <p>The namespace of the vocabalary as a string</p>
     *  @see #NS */
    public static String getURI() {return NS;}

    /** <p>The namespace of the vocabalary as a resource</p> */
    public static final Resource NAMESPACE = m_model.createResource( NS );

    /** <p>List of items matching a GetQuery, stored in an RDF:bag, if there are any</p> */
    public static final Property itemList = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/itemList" );

    /** <p>DOAFI password, hashed using md5, for making database modifications (required
     *  for a SetQuery)</p>
     */
    public static final Property password = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/password" );

    /** <p>Comment explaining database modifications</p> */
    public static final Property editComment = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/editComment" );
```

```
    /** <p>a depth limit, indicating the level of information to be returned</p> */
    public static final Property depth = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/depth" );

    /** <p>Modification version state to rollback to, according to the items in movementHistory</p> */
    public static final Property rollbackTo = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/rollbackTo" );

    /** <p>an indicator of whether the query was successful - a property of Result</p> */
    public static final Property status = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/status" );

    /** <p>DOAFI username for making database modifications (required for a SetQuery)</p> */
    public static final Property username = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/username" );

    /** <p>A flag to set to true if these items should be deleted rather than updated</p> */
    public static final Property deleteItems = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/deleteItems" );

    /** <p>the maximum number of items (at depth 1) matching this query</p> */
    public static final Property maxItems = m_model.createProperty(
                    "http://doafi.4angle.com/elements/q-0.1/maxItems" );

    /** <p>Look up query for a FoodItem using DOAFI in the query</p> */
    public static final Resource GetQuery = m_model.createResource(
                    "http://doafi.4angle.com/elements/q-0.1/GetQuery" );

    /** <p>A wrapper for the result of a query response</p> */
    public static final Resource Result = m_model.createResource(
                    "http://doafi.4angle.com/elements/q-0.1/Result" );

    /** <p>Like GetQuery, but used to set/delete/add items in a query instead of getting
     *  them</p>
     */
    public static final Resource SetQuery = m_model.createResource(
                    "http://doafi.4angle.com/elements/q-0.1/SetQuery" );

}
```

B.3 Administration module package

B.3.1 AddUser

```
package admin;

import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.xalan.*;
import java.sql.*;
import java.security.*;
import java.io.*;
import java.math.BigInteger;

/**
 * Servlet called to add a new user to the access list, it takes the parameters username,
 * password and authorisation level. Password should be supplied plain and is hashed here.
```

```
 * */

public class AddUser extends HttpServlet {

  public Connection conn;
  public PrintWriter out;

  String DB_URL = "jdbc:mysql://localhost/doafi";  // URL of database server
  String DB_USER = "root"; // database user id
  String DB_PASSWD = ""; // database password
  String DB = "MySQL"; // database type

  public void doGet(HttpServletRequest request,
      HttpServletResponse response)
    throws IOException, ServletException {

    this.doPost(request, response);

  }

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

    String username;
    String password;
    String authorisation;
    String sql;
    int affectedRows = 0;

    response.setContentType("text/html");
    out = response.getWriter();

    try { // connect to db
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      // Create database connection
      conn = DriverManager.getConnection ( DB_URL, DB_USER, DB_PASSWD );
    }
    catch (Exception e) {}

    username = request.getParameter("username");
    password = request.getParameter("password");
    authorisation = request.getParameter("authorisation");

    // hash password with md5
    password = hashPassword(password);

    sql = "INSERT INTO users(username, password, authorisation) VALUES('";
    sql += username;
    sql += "', '";
    sql += password;
    sql += "', ";
    sql += authorisation;
    sql += ")";


    try {
      Statement s = conn.createStatement();
      affectedRows = s.executeUpdate(sql);
    }
    catch (Exception e) {}
    if (affectedRows == 1) {
```

```
      // great
      out.println("OK: ");
      out.println(password);
    }
    else {
      // bad
      out.println("not OK");
    }
    try {
      conn.close();
    }
    catch (Exception e) {
      out.println("Error closing database connection");
    }
  }

  public static String hashPassword(String password) {
    String hashword = null;
    try {
      MessageDigest md5 = MessageDigest.getInstance("MD5");
      md5.update(password.getBytes());
      BigInteger hash = new BigInteger(1, md5.digest());
      hashword = hash.toString(16);
    }
    catch (NoSuchAlgorithmException nsae) {
      // ignore
    }
    return hashword;
  }
}
```

B.3.2 EditUser

```
package admin;

import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.xalan.*;
import java.sql.*;
import java.security.*;
import java.io.*;
import java.math.BigInteger;

/**
 * Servlet called to edit a  user in the access list, it takes the parameters username,
 * password and authorisation level. Password should be supplied plain and is hashed here.
 * */


public class EditUser extends HttpServlet {

  public Connection conn;
  public PrintWriter out;
  public boolean errorBit;

  String DB_URL = "jdbc:mysql://localhost/doafi";  // URL of database server
  String DB_USER = "root";      // database user id
  String DB_PASSWD = "";       // database password
  String DB = "MySQL";          // database type
```

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws IOException, ServletException {
  this.doPost(request, response);
}

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

  String username;
  String password;
  String authorisation;
  String passwordconfirm;
  String sql;
  int affectedRows = 0;
  String errorMsg = "";
  errorBit = false; // no errors yet

  response.setContentType("text/html");
  out = response.getWriter();

  try { // connect to db
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    // Create database connection
    conn = DriverManager.getConnection ( DB_URL, DB_USER, DB_PASSWD );
  }
  catch (Exception e) {}

  username = request.getParameter("username");
  password = request.getParameter("password");
  authorisation = request.getParameter("authorisation");
  passwordconfirm = request.getParameter("passwordconfirm");

  if (username == null ||
    password == null ||
    authorisation == null ||
    passwordconfirm == null) {

    errorBit = true;
    errorMsg = "field not set in form";
  }
  else if (!password.equals(passwordconfirm)) {
    errorBit = true;
    errorMsg = "passwords do not match";
  }

  // hash password with md5
  password = hashPassword(password);

  sql = "UPDATE users set username = '";
  sql += username;
  sql += "', password = '";
  sql += password;
  sql += "', authorisation = ";
  sql += authorisation;
  sql += " WHERE username = '";
  sql += username;
  sql += "'";

  if (!errorBit) {// no errors detected so far
    try {
```

```
        Statement s = conn.createStatement();
        affectedRows = s.executeUpdate(sql);
      }
      catch (Exception e) {}
    }
    if (affectedRows == 1) { // only if the db mod took place
      // great
      out.println("OK: ");
      out.println(password);
    }
    else {
      // bad
      out.println("not OK: ");
      out.println(errorMsg);
    }
    try {
      conn.close();
    }
    catch (Exception e) {
      out.println("Error closing database connection");
    }
  }

  public static String hashPassword(String password) {
    String hashword = null;
    try {
      MessageDigest md5 = MessageDigest.getInstance("MD5");
      md5.update(password.getBytes());
      BigInteger hash = new BigInteger(1, md5.digest());
      hashword = hash.toString(16);
    }
    catch (NoSuchAlgorithmException nsae) {
      // ignore
    }
    return hashword;
  }
}
```

B.3.3 DeleteUser

```
package admin;

import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.xalan.*;
import java.sql.*;
import java.security.*;
import java.io.*;
import java.math.BigInteger;

/**
 * Servlet called to delete a user from the access list, it takes the parameter username
 * and deletes the user.
 * */

public class DeleteUser extends HttpServlet {

  public Connection conn;
  public PrintWriter out;
```

```
  String DB_URL = "jdbc:mysql://localhost/doafi";  // URL of database server
  String DB_USER = "root";      // database user id
  String DB_PASSWD = "";       // database password
  String DB = "MySQL";          // database type

  public void doGet(HttpServletRequest request,
      HttpServletResponse response)
    throws IOException, ServletException {
    this.doPost(request, response);
  }

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

    String username;
    String sql;
    int affectedRows = 0;

    response.setContentType("text/html");
    out = response.getWriter();

    try { // connect to db
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      // Create database connection
      conn = DriverManager.getConnection ( DB_URL, DB_USER, DB_PASSWD );
    }
    catch (Exception e) {}

    username = request.getParameter("username");

    sql = "DELETE FROM users WHERE username = '";
    sql += username;
    sql += "'";


    try {
      Statement s = conn.createStatement();
      affectedRows = s.executeUpdate(sql);
    }
    catch (Exception e) {}
    if (affectedRows == 1) {
      // great
      out.println("OK: deleted ");
      out.println(username);
    }
    else {
      // bad
      out.println("not OK");
    }
    try {
      conn.close();
    }
    catch (Exception e) {
      out.println("Error closing database connection");
    }
  }
}
```

Appendix C

Administration

C.1 Project schedule

There are 4 deliverables for the project. These will form the major milestones of the project.

- Project proposal: submitted 21 October 2004
- Interim report: due 09 December 2004
- Draft report: due 17 March 2005
- Final report: due 28 April 2005

The scope of this document will include up to (and including) the final report. In addition, an oral presentation must be given upon completion of the final report.

Minor milestones (as defined below) are scheduled to be completed as follows:

- Complete constituent parts for Interim Report: 06 December 2004
- Completion of Design - phase 1: 04 January 2005
- Completion of Coding / Implementation - phase 1: 25 January 2005
- Completion of Design - phase 2: 05 February 2005
- Completion of Coding / Implementation - phase 2: 28 February 2005
- Completion of Testing & Evaluation - phase 2: 09 March 2005

Progress is recorded on the attached GANTT chart.

C.2 Project phases

As the project domain is not entirely well known to me, I will pursue an iterative and incremental approach to development. In the first phase, I will design and build an advanced prototype, and in the 2nd phase I will revise this according to the overall aims of the system. This should facilitate evaluation after the first phase which can be taken into consideration for development.

Requirements analysis

- Requirements analysis sections
- Project plan
- Interim report

Design

Phase 1:

- Create a basic ontology for description of food items, considering just simple properties
- Convert ontology to extensible RDF Schema
- Create a query ontology
- Convert ontology to extensible RDF Schema
- Unique identifiers
- A format of unique IDs for products and producers to become URIs
- Low-level design for server

Phase 2:

Bug fixes and refactoring

Continuing development: (not dependent on Coding phase 1)

- Data protection / integrity: authentication and restriction system
- How to roll back changes
- Users and authentication for adding data (modifications to Schema for 'contributer' information)
- Increase information content of ontologies: more advanced properties

- Augment original Schema with properties from other ontologies (eg Dublin Core Metadata Initiative, Geo, Wine, Vegetarian)
- Low-level design for web-based client to add,view,edit
  - Create RDF queries based on user input
  - Interpret responses from server and present to user

Coding and implementation

Phase 1:

- Configure server tools required (MySQL, Jena, Apache Tomcat etc.)
- Create database structure
- Build server query processor
  - Receive queries over HTTP from client
  - Interpret RDF queries
  - Interface to database (for reading and writing)

Phase 2:

- Bug fixing
- User support system (for creation of user accounts and identity)
- Support for users with respect to RDF add/edit queries
- Authentication and restrictions
- Create web-based client interface for demonstration and testing

Testing and evaluation

Phase 1:

- Create instantiations of RDF Schema to try different domains
- View, add, edit items using RDF to query the server
- Evaluate shortcomings of the product and bugs

Phase 2:

- View, add, edit items using RDF to query the server
- Internal evaluation, based on previous evaluation and further development
- Independent evaluation: farmer / food organisation to add data to the system

Final evaluation:

Evaluation of progress, pitfalls and intractable problems

Draft report

Documentation of project phases to be continuous. The writing of the report is broken down into the various sections of the report.

- Introduction
- Requirements analysis (modifications)
- Implementation
- Testing
- Conclusions

Final report

- Revisions to Draft
- New material which didn't make it in to the Draft
- Appendices
- References

C.3 Progress report