Ganesha
A simple RDF/XML editor

Project Report

by:
James Nash

supervised
by:
Dr. Stephen Jarvis

year:
2003 / 2004

Ganesha – a simple RDF/XML editor
Project Report

This document is also available online at:
http://event-horizon.kicks-ass.net/uni/ganesha/docs/report/

Keywords

Keywords relating to the content of this report:

XML, RDF, meta-data, editor, Java, GUI, Jena, "Semantic Web"

Abstract

The Resource Description Framework (RDF) is a format for describing anything from web pages to people as well as the relations between them in a way that computers can easily process. A key area of interest for this technology is the World Wide Web, where it would allow the vast amount of information that is available online to be used far more effectively than has has so far been the case. This report highlights aspects of RDF that make it too complex to appeal to most web-designers and thus hinder its widespread adoption. An account is given of the design and implementation of an application, named Ganesha, that addresses these issues and simplifies the creation and publication of RDF data.

Table of Contents

Keywords 3

Abstract 3

Table of Contents 4

Chapter 1. - Introduction 7

1.1 Background 7

1.1.1 The Resource Description Framework 7

1.1.2 The Semantic Web 10

1.2 Project Motivation 11

1.3 Aims and Objectives 12

1.4 Outline of Report 13

Chapter 2. - Background Research 15

2.1 Difficulties of RDF/XML 15

2.1.1 The RDF model 15

2.1.2 Serialisation 16

2.2 RDF Uses 19

2.3 Other editors 21

2.3.1 Overview 21

2.3.2 FOAF-o-matic 21

2.3.3 RDF Editor 22

2.3.4 IsaViz 23

2.3.5 Suitability 24

2.4 Choice of language and libraries 25

Chapter 3. - Design 27

3.1 Overview 27

3.2 GUI Design 27

3.2.1 Choice of features 27

3.2.2 Basic concepts 27

3.2.3 Layout 30

3.2.4 Example walk-through 32

3.3 Program Design.....37
3.3.1 Structure.....37
3.3.2 Interface design.....37
3.3.3 The back-end design.....40
3.3.4 GUI code design.....40

Chapter 4. - Development & Testing.....43
4.1 Development Methodology.....43
4.2 Implementation.....43
4.3 Code testing.....45
4.3.1 Overview.....45
4.3.2 Back-end.....45
4.3.3 GUI code.....46
4.4 GUI Usability testing.....48

Chapter 5. - Conclusions.....49
5.1 Summary.....49
5.2 Self Assessment.....50
5.2.1 What is the technical contribution of this project?.....50
5.2.2 Why should this contribution be considered important to computer science?.....50
5.2.3 How can others make use of this project?.....50
5.2.4 Why should this project be considered an achievement?.....50
5.2.5 What are the weaknesses of this project?.....50
5.3 Future developments.....51

Acknowledgements.....53

Bibliography.....55

Appendix I: Further reading.....57

Appendix II: User Manual.....59

Appendix III: Source Code (CD-ROM).....71

Chapter 1. - Introduction

1.1 Background

1.1.1 The Resource Description Framework

Many file formats have provisions to embed descriptive data about their content, so-called meta-data, within them. HTML web pages have meta-tags, MP3 music has ID3 tags, JPEG pictures have comment fields to name a few. This meta-data allows machines to gain some (limited) understanding of what a file contains so that they may, for example, classify it or search for it in a sensible way. Unfortunately all these format specific meta-data schemes vary widely in their abilities, are mutually incompatible and are generally not extensible. This limits the potential uses of such meta-data. Given an MP3 file a program can currently tell the user who the artist of that song is and what album it belongs to, but it would not be able to find out what that artist's website is or what other albums he or she may have released. The Resource Description Framework (RDF) is an attempt to alleviate these problems by defining a standardised meta-data framework.

Essentially RDF is a model for representing information intended for machine consumption. RDF data can be serialised in a variety of formats, the most widely used of which is as an XML application (known as RDF/XML). The RDF model, its semantics and the syntax of the various serialisations have been standardised by the World Wide Web Consortium (W3C, 2004c). It was first proposed in 1997, although the first recommended RDF specification was not released until 1999. Since then the specifications have been repeatedly updated and the latest revisions at the time of writing were released in February 2004.

RDF expresses statements about resources. A resource is anything that can be uniquely identified by a Uniform Resource Indicator (URI) such as a web page which is identified by its address (URL) or a book which is identified by its ISBN number. Resources are then described by any number of properties, where a property consists of a property type and its value. Values can either be literal values, such as a string or a number, or they can themselves be resources. Thus each RDF statement is a triple consisting of a resource URI, a property type and a value.

RDF data can be visualised as a graph. By convention resources are depicted as round nodes, literal values as square nodes and property types as directed edges between resources and values. For example, if we wanted to express the statement "*The author of http://www.twiddles.com/ is James Nash*" the corresponding RDF graph might look like this:

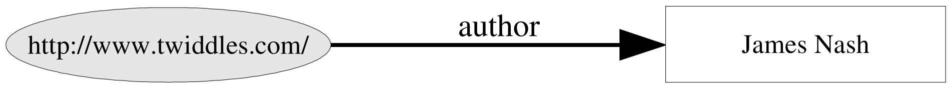


Figure 1-1: A simple RDF graph

Resources may have any number of properties so more complex graphs are possible:

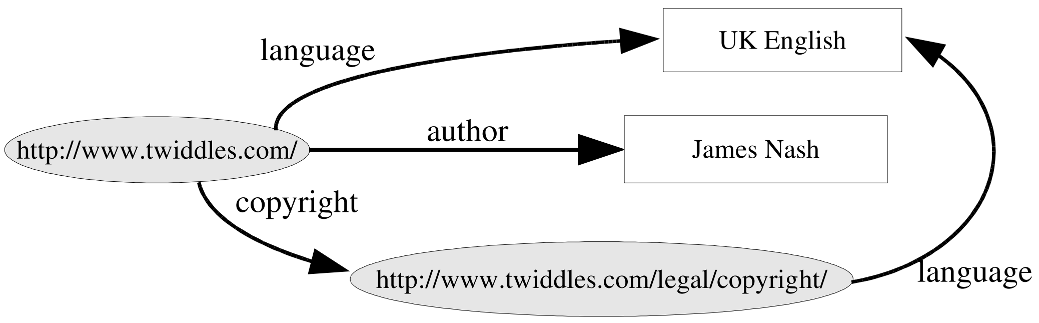


Figure 1-2: A larger RDF graph

RDF also includes the concept of anonymous resources. An anonymous resource is a resource that does not have a URI. This can be useful when there is a need to describe something has not yet been assigned a URI like a book that has not yet been published. In this case an anonymous resource may be used as a temporary place holder until it can be replaced with a URI. A further use of anonymous resources is when describing things that do not have any unique identifier but are instead identified in terms of their properties. A good example for this is a person. Neither a person's name, email address or telephone number are necessarily unique to that person alone, however the combination of all those properties and perhaps others may be sufficient to identify a person uniquely.

In the above examples we have stated that "*The author of http://www.twiddles.com/ is James Nash*" but there are probably many different people named James Nash in the world and so far a machine attempting to evaluate this data has no way of knowing which one is meant. If the example was refined using an anonymous resource and some extra properties the author in question may be reasonably well identified.

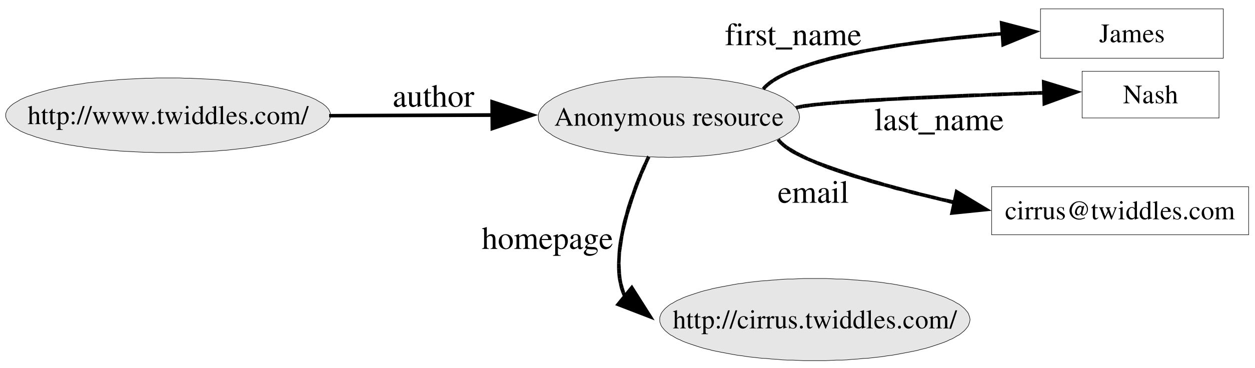


Figure 1-3: Anonymous node example

This RDF graph could be expressed in plain English as *"The author of http://www.twiddles.com/ is the resource whose first name is James, last name is Nash, email address is cirrus@twiddles.com and homepage is the resource identified by http://cirrus.twiddles.com/"*. Given this data a machine would now be a position to differentiate between the author of *http://www.twiddles.com/* and any other person named James Nash that it might encounter.

The RDF standard itself does not define any property types. Instead various interest groups define standard sets of properties and their intended semantics, known as meta-data vocabularies, for their particular domain of interest. By using related standards such as the RDF Schema language (RDFS) and the Web Ontology Language (OWL) these vocabularies and relations or restrictions between their properties can be defined in a machine-readable format that RDF applications may then use. The details of these standards is beyond the scope of this document and interested readers are referred to Appendix I for sources of information on this topic.

1.1.2 The Semantic Web

The Semantic Web is an effort to supplement the current Web with machine-readable meta-data using RDF. The hope is that this will allow content on the Web, Web-services and applications to be combined in much more powerful and useful ways than is currently possible. Giving machines not only information about resources themselves, but also knowledge *about* them in a standardised and distributed way is key to enabling this. By using RDF's model of simple statements about resources and some logic computers can begin to reason about information on the Web without requiring complex natural language processing or artificial intelligence (Berners-Lee et al, 2001).

The benefits of this approach will be immense. Web searches would improve since they will be able to index non-textual data such as pictures, music or video more effectively and not get confused by searches for words which can have several different meanings. For example, a search for a person named “Cook” could avoid returning results about cooks or the Cook islands. Web browsers could aid users researching a subject by finding pages related to the subject or finding a page's author, his biography, contact details and other works. Personal assistant style programs could automatically organise meetings by finding times and locations that suit all attendees, booking the tickets and setting reminders in everyone's calendars.

Some of these applications may already be possible by other means but they would almost certainly have to depend on proprietary services and standards. This limits their uses as well as their audience because they will always be locked into whatever service providers they happen to use and platforms or data formats they happen to support. Since knowledge on the Semantic Web can be distributed across many different sources just as information on the Web is spread across many different sites and pages, applications can be free from these limitations. A personal assistant program, for instance, would not need to be restricted to a particular ticket booking service, it could use any services it finds, including new ones as and when they appear.

1.2 Project Motivation

The concept of a Semantic Web has great potential to make our lives easier and more efficient and will undoubtedly play a big role in the future of computing. However, in order for people to begin creating applications that exploit this potential there needs to be a significant amount of RDF data out on the Web. Although RDF has now existed for about five years it does not appear to have reached this critical mass yet. Publishing some forms of RDF such as RSS news feeds[1] have begun to gain momentum, but the vast majority of websites still do not contain any RDF data.

Large websites tend to use custom content management systems, small and medium-sized websites are generally created with the aid of WYSIWYG[2] web design tools or word processors. These approaches require very little or no understanding of the underlying HTML code which greatly simplifies the process. Thus the vast amount of information that has been published on the Web is surely due in part to the ease with which this can be done. It follows therefore that in order for large amounts of RDF data to be published there needs to be a method for doing so that is simple enough to appeal to the majority of people publishing material on the web.

In the case of content management systems, especially custom-made ones, there is little choice but to update those systems and add RDF functionality. For WYSIWYG editors there is some scope to add RDF features into them, but because the uses of RDF go far beyond just describing web pages it makes more sense to have a separate tool to create RDF files. A few dedicated RDF editors do already exist (and are discussed in Chapter 2.3) but none of them fully address the needs of an average computer user who wants a simple way to publish content and meta-data but has no interest in learning the specifics of RDF or the syntax of its serialisations.

1 RSS (RDF Site Summary): A vocabulary for publishing news-feeds. See http://web.resource.org/rss/1.0/spec
2 WYSIWYG: What you see is what you get

1.3 Aims and Objectives

This project has attempted to address the issues raised in the previous section by producing a simple, easy-to-use RDF editor named Ganesha[3]. It is a graphical user interface (GUI) based application capable of creating and editing RDF files. Its target audience (referred to hereafter as 'average user(s)' or 'target user(s)') are individuals publishing information on the Web. They are expected to be amateur or semi-professional web designers familiar with WYSIWYG tools such as Macromedia Dreamweaver, Adobe GoLive or Microsoft Frontpage. No knowledge of the RDF model or HTML and XML syntax is assumed.

In order to improve timetabling and organisation the development of Ganesha was broken down into a series of objectives. Each objective depended on the completion of the previous objective. The following list represents the steps of this project in chronological order:

- Objective 1: Research
    - Obj. 1.1: Evaluate RDF from the perspective of the target user
        - Obj. 1.1.1: Determine potential difficulties when working with RDF
        - Obj. 1.1.2: Determine users' uses of RDF
    - Obj. 1.2: Evaluate other RDF editors
    - Obj. 1.3: Evaluate languages and tools for creating Ganesha
- Objective 2: Design
    - Obj. 2.1: GUI design
    - Obj. 2.2: Program design
- Objective 3: Implementation
    - Obj. 3.1: Choose development methodology
    - Obj. 3.2: Build
- Objective 4: Testing
    - Obj. 4.1: Code testing
    - Obj. 4.2: Usability testing

3 Named after the Hindu god of knowledge and destroyer of obstacles, Ganesha. (Sometimes also referred to as Ganesh)

1.4 Outline of Report

The following chapters of this report detail how each of the project objectives was approached and completed:

- Chapter 2 (Page 15) outlines the research undertaken to fulfil Objective 1.
- Chapter 3 (Page 27) covers the design steps of Objective 2.
- Chapter 4 (Page 43) describes the implementation and testing procedures of Objectives 3 and 4.

The report is concluded in Chapter 5 with a critical assessment of the project's achievements. Supplemental information and documents such as source code, further reading and a user manual are included in the appendices at the end of the report.

Chapter 2. - Background Research

2.1 Difficulties of RDF/XML

2.1.1 The RDF model

The underlying concept of the RDF model, as described in chapter 1.1.1, is a fairly simple one. The corresponding graphs with resources and literals connected by property types are also easily understandable. However, there are other concepts in RDF which are more complex. While it is true that values of properties can only be literals or resources, both literals and resources can be typed (similar to the data-types found in most programming languages). Furthermore there is the concept of reification whereby the value of a statement can itself be a statement (which is achieved by defining the statement that is the value as a special kind of resource).

The RDF specification (W3C, 2004b) itself defines some basic resource types, but most are defined as part of vocabularies just as most property types are. It is worth noting at this point that all property types and and resource types are identified by URIs. Often a set of types that belong to the same vocabulary will a share a common base URI and differ only in the fragment part of the URI. By convention the common part of the URIs (which also serves as an XML namespace when serialised as RDF/XML) points to a document that defines those types.

Three important resource types from the RDF specification are bags, sequences and alternatives. These are intended as containers for when a property of a resource has several values. Bags simply imply a set of values, alternatives are bags except that one value is defined as the default choice and sequences imply that the values are in a particular order. Since these containers are only meant to hold values but in fact consist of property types and values they use a dummy property type called 'li'[4].

4 The 'li' property type is named after the list item tag <li> in HTML.

Extending the example from Figure 1-3 to use resource types illustrates the complexity that these new concepts can add to the otherwise simple model:

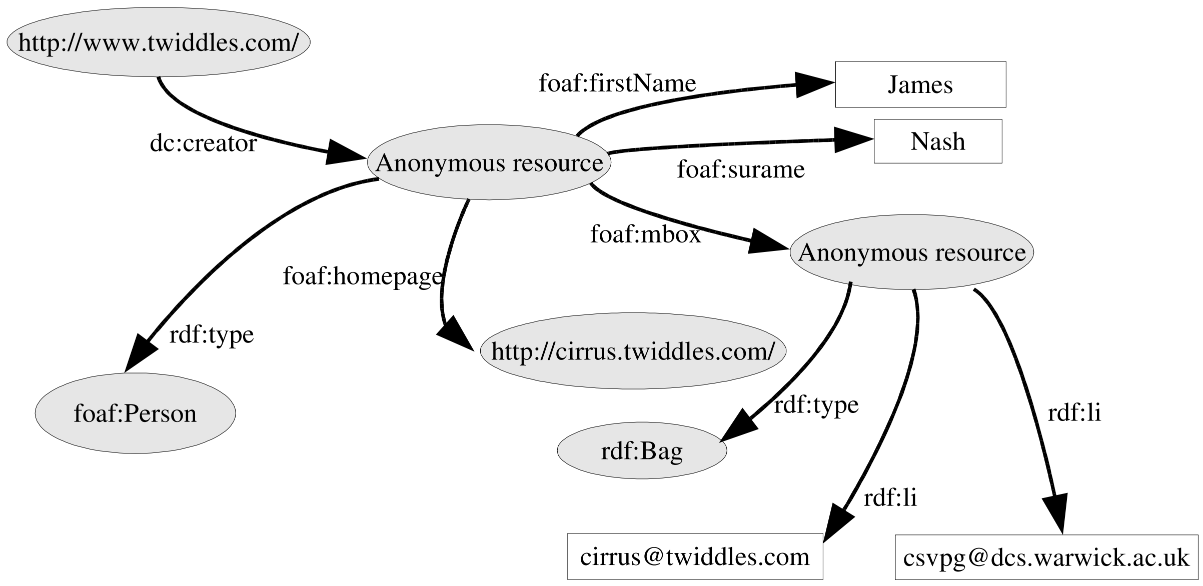


Figure 2-1: RDF graph with resource types

Compared to the original example in chapter 1.1.1 the property types have been replaced with URIs (using their common namespace abbreviations for better legibility), the anonymous resource describing James Nash has been given the type Person and a bag has been used for the email property to indicate that two email addresses exist for this person[5].

There are several aspects in this example that may appear confusing to people not familiar with the details of RDF. For example the fact that in order to assign two email addresses to a person an anonymous resource with three properties was used. While the discussed RDF concepts are both important and useful it is clear that they also steepen the learning curve thus making RDF less attractive to many users. With respect to objective 1.1.1, Ganesha's design needed to take the difficulties of resource and literal types and reification for average users into account.

2.1.2 Serialisation

When storing RDF data to a file or exchanging it between applications it needs to be serialised. The RDF specifications define two formats for doing so: N-Triples and RDF/XML. RDF data serialised as one format can be converted to the other without any loss of information. The N-Triples format lends itself well to some forms of automated processing and validation but for the purposes of publishing or exchanging data RDF/XML is more flexible. Since RDF/XML is an XML application it

5 This example is for illustrative purposes only: While it is valid RDF, some of the properties have not been used as intended by their respective vocabularies.

can be embedded within other XML documents and processed by non RDF-aware parsers or tools. For example RDF/XML data could be inserted into an SVG image. SVG viewers could ignore the RDF data, whereas RDF applications could extract it and gather meta-data about the image.

In terms of complexity N-Triples is the simpler of the two formats. Each line of an N-Triples file contains an RDF triple, whose elements are separated by whitespace, and is terminated by a full stop. URIs are enclosed in angle brackets and anonymous resource identifiers begin with “_:”. Serialising the example from Figure 2-1 into N-Triples would yield the following result[6] (wrapped lines are indicated by a “➘” symbol):

```
<http://www.twiddles.com/> <http://purl.org/dc/elements/1.1/creator> _:A0 .

_:A0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>➘
  <http://xmlns.com/foaf/0.1/Person> .
_:A0 <http://xmlns.com/foaf/0.1/firstName> "James" .
_:A0 <http://xmlns.com/foaf/0.1/surname> "Nash" .
_:A0 <http://xmlns.com/foaf/0.1/homepage> <http://cirrus.twiddles.com/> .
_:A0 <http://xmlns.com/foaf/0.1/mbox> _:A1 .

_:A1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>➘
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag> .
_:A1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#li> "csvpg@dcs.warwick.ac.uk" .
_:A1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#li> "cirrus@twiddles.com" .
```

Output 2-1: Figure 2-1 serialised as N-Triples

RDF/XML is a more complex format. The full syntax definitions are documented in the “RDF Syntax Specification” (W3C, 2004a). Key points of difficulty for Ganesha users are:

- All the XML syntax rules apply (every tag must have a closing tag, exactly one root element, attributes must be quoted etc.)
- There are two styles of RDF/XML: Normal syntax and abbreviated syntax
- The two styles may be mixed in the same document
- There are shorthand forms for some RDF concepts (such as typed resources)

6 The namespace abbreviations in Figure 2-1 have been replaced with the full URIs

As a result the same RDF data can result in many different looking, yet equally valid, RDF/XML files. The following example shows the graph in Figure 2-1 serialised as non-abbreviated RDF/XML[7]. Each set of statements about a resource is enclosed in *Description* tags, the property types appear as tags in a *Description* block, literal values are character data between property tags and resource values are attributes of property tags.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/" >
  <rdf:Description rdf:about="http://www.twiddles.com/">
    <dc:creator rdf:nodeID="A0"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A0">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <foaf:firstName>James</foaf:firstName>
    <foaf:surname>Nash</foaf:surname>
    <foaf:homepage rdf:resource="http://cirrus.twiddles.com/"/>
    <foaf:mbox rdf:nodeID="A1"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
    <rdf:li>csvpg@dcs.warwick.ac.uk</rdf:li>
    <rdf:li>cirrus@twiddles.com</rdf:li>
  </rdf:Description>
</rdf:RDF>
```

Output 2-2: Figure 2-1 serialised as RDF/XML

RDF serialisation, especially as RDF/XML, requires considerable background knowledge, both of RDF's abstract concepts and of the serialisation syntaxes. Ganesha's average users are expected to have neither. Ganesha therefore had to greatly simplify the serialisation process.

7 This example was actually output directly by Ganesha.

2.2 RDF Uses

Objective 1.1.2 required that target users' likely uses of RDF were established. As web designers and publishers their main area of interest is likely to be describing properties of their documents. HTML files can contain embedded meta tags which provide a basic mechanism for including properties with literal values. In practise only two types of document property are widely described using meta tags: keywords and description. Describing other properties of a document such as related documents, copyright information, detailed descriptions of authors etc. are equally important but cannot be achieved with HTML meta tags. Furthermore information may not only be published as HTML - various image, audio and video formats are also widely used. Such formats do not always have facilities for embedding meta data. RDF can provide a solution in these situations.

The RDF website (W3C, 2004c) contains links to various projects developing meta data vocabularies. Dublin Core, Friend of a Friend (FOAF) and Creative Commons stand out as being particularly relevant to the needs of many web designers and publishers since they can be used to address the needs described above:

- Dublin Core defines attributes common to any kind of document: title, author, rights etc.
- FOAF is a vocabulary for describing people, their interests and activities and relations between them. It also allows for the description of picture contents
- Creative Commons is intended for describing copyright restrictions of works

Although Ganesha has been made capable of handling arbitrary vocabularies it was advantageous to focus on the vocabularies above since they are expected to be the most frequently used. Besides the basics of the RDF model, these vocabularies require the use of resource types (but not necessarily typed literals).

To confirm these assumptions about likely RDF usage a variety of RDF files were downloaded from the following sources and analysed using the W3C's RDF validator[8]:

Description	*URL*
RDF file of W3C's homepage	http://www.w3.org/Overview-about.rdf
Listing Friend-of-a-Friend (FOAF) RDF files	http://rdfweb.org/topic/FOAFBulletinBoard
Creative Commons license as RDF (embedded in the HTML document)	http://creativecommons.org/licenses/nc-sa/1.0/
Google search for RDF files	http://www.google.com/search?q=rdf+filetype%3Ardf

Table 2-1: Sources of RDF data

The analysed RDF files confirmed that Dublin Core, FOAF and Creative Commons were the most widely used vocabularies. No examples of typed literals or reification were encountered.

The Google search in particular led to many RSS (RDF Site Summary) files. These are for syndicating news items and are definitely of interest to web publishers. However due to their ephemeral nature they are generally produced dynamically by server scripts rather than an application such as Ganesha which only outputs static files. Because of this creating RSS files is unlikely to be a common use of Ganesha and RSS files were omitted from the research.

8 The RDF validator is available at: http://www.w3.org/RDF/Validator/

2.3 Other editors

2.3.1 Overview

There are several ways to create RDF files. The trivial approach is to use any text editor and type RDF/XML or N-Triples data out by hand. Besides that there exist several RDF-specific editors. They exhibit different approaches for allowing users to create and interact with RDF data. The following sections cover Objective 1.2 by evaluating three popular tools for creating RDF data with respect to the needs of Ganesha's target users.

2.3.2 FOAF-o-matic

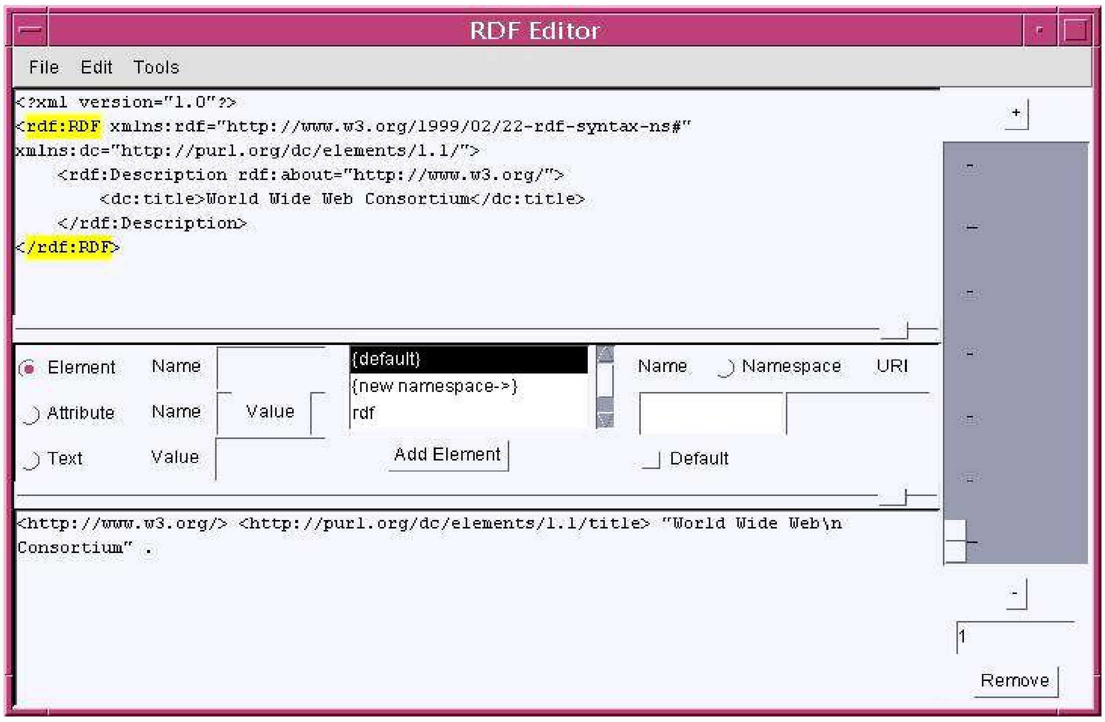
FOAF-o-matic (Dodds, 2002) is not really an RDF editor since it only permits the creation of RDF/XML containing statements using the Friend of a Friend (FOAF) vocabulary[9]. It is essentially a JavaScript embedded in a web page that takes input values from a form and outputs them enclosed in the appropriate RDF/XML tags. It is being evaluated since its form-fill interface is a very simple one that does not require users to know anything about the RDF model or RDF/XML syntax.

The main advantage of this interface is simplicity. Users have a clear starting point (the empty form fields), there is virtually no room for user errors since the form constrains possible user actions to entering value and clicking the 'submit' button and the form fields give users immediate feedback of their actions since they can observe their entries as they type them. The disadvantages are that FOAF-o-matic is limited to one particular vocabulary[10] and is only capable of producing fairly primitive RDF files.

9 The FOAF vocabulary specification is available at: http://xmlns.com/foaf/0.1/
10 In fact, it only covers a subset of the FOAF vocabulary

2.3.3 RDF Editor

RDF Editor (Punin, 2002) is a standalone application written in Java. It presents users with a view of the RDF/XML code of the current RDF data. Users can select elements of the code and modify them, remove them or add new ones using various input fields and buttons:



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.w3.org/">
        <dc:title>World Wide Web Consortium</dc:title>
    </rdf:Description>
</rdf:RDF>
```

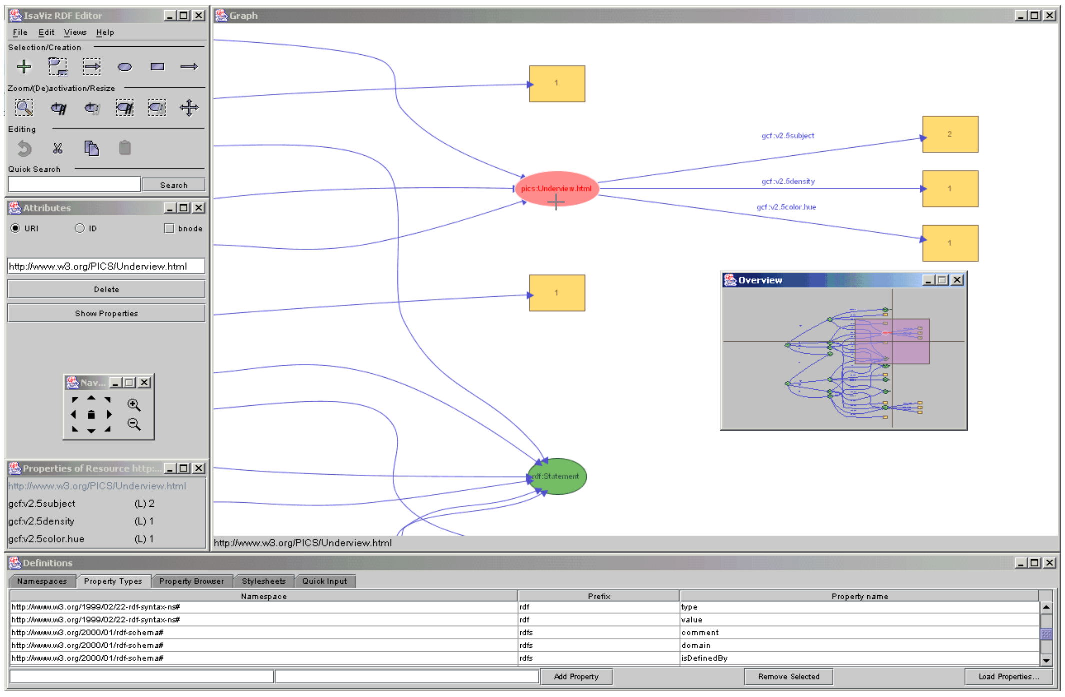
```
<http://www.w3.org/> <http://purl.org/dc/elements/1.1/title> "World Wide Web\n
Consortium" .
```

Figure 2-2: RDF Editor screenshot (Punin, 2002)

RDF Editor requires users to be familiar with RDF/XML syntax. Any RDF data could be represented using it, provided users have sufficient background knowledge. The user interface offers some advantages over a plain text editor as it reduces the margin for user errors by only allowing users to edit RDF/XML elements through a specialised form interface. However, as can be seen in the screenshot (Figure 2-2), the layout of the fields is not particularly intuitive – for example several fields share the same labels ('name' and 'value') yet perform different actions (namespace name, attribute name, element name etc.). Furthermore by requiring knowledge of RDF this editor has a steeper learning curve that prevents its use by users such as Ganesha's target users.

2.3.4 IsaViz

IsaViz (Pietriga, 2003) is an RDF editor that has been developed by members of the W3C. It visualises RDF data as a graph and allows users to edit or create RDF statements by altering the nodes and arcs of the graph:



http://www.w3.org/PICS/Underview.html

gcf:v2.5subject	(L) 2
gcf:v2.5density	(L) 1
gcf:v2.5color.hue	(L) 1

http://www.w3.org/PICS/Underview.html

Namespace	Prefix	Property name
http://www.w3.org/1999/02/22-rdf-syntax-ns#	rdf	type
http://www.w3.org/1999/02/22-rdf-syntax-ns#	rdf	value
http://www.w3.org/2000/01/rdf-schema#	rdfs	comment
http://www.w3.org/2000/01/rdf-schema#	rdfs	domain
http://www.w3.org/2000/01/rdf-schema#	rdfs	isDefinedBy

Figure 2-3: IsaViz screenshot (Pietriga, 2003)

The user interface has various aids for navigating RDF graphs such as a zoom tool and resource list, but very large RDF models can still become difficult to follow due to overlapping or closely situated arcs. Any RDF data can be represented and created in IsaViz, but an understanding of RDF, specifically its graph model, is required in order to do so. IsaViz does not let users interact with RDF/XML code directly, as RDF Editor does, so knowledge of the serialisation process is not required.

IsaViz is a powerful tool for advanced users, but its user interface is not well suited for Ganesha's target users. As can be seen on in Figure 2-3, the interface is divided into several windows and toolboxes. There is no progressive disclosure of information or controls. This allows for efficient use by experts but can be overwhelming for beginners as there is a lot to observe in one go and no clear point of entry. Furthermore the graph view can easily become cumbersome to follow and navigate on complex RDF data.

2.3.5 Suitability

Of the three editors discussed FOAF-o-matic's form fill interface is the simplest to use. RDF Editor and IsaViz' approaches are unsuitable for Ganesha's target users due to their dependency on RDF or XML background knowledge and complexity. However, FOAF-o-matic's functionality is far too simplistic. Ganesha is intended to be a general purpose RDF editor so it cannot be limited to just one vocabulary – hence it attempts to generalise FOAF-o-matic's approach so as to work with any vocabulary.

2.4 Choice of language and libraries

Several libraries for processing RDF/XML exist. Each is written using a particular programming language, meaning applications that use the library must be written in the same language. Thus the choice of programming language and RDF library had to be made together. The following combinations are discussed in chapters 8, 9 & 11 of “Practical RDF” (Powers, 2003). Where several APIs exist for a given language the most popular or most recently updated one was taken.

- Jena & Java
- PerlRDF & Perl
- RAP (RDF API for PHP) & PHP
- RDFLib & Python
- Drive & C#

Perl & Python are scripting languages that do not have any built-in support for GUI applications. Libraries exist for both that can add such functionality (for example GTK-Perl[11]), but these would have added more requirements for users in order to run Ganesha and are generally tied to a particular GUI toolkit or platform. PHP is primarily for hypertext parsing and therefore is not suitable for a standalone GUI application.

C# on its own does not provide a GUI toolkit. Microsoft's .Net platform provides one for Windows only. The open-source Mono project[12] is working towards producing a cross-platform implementation of the .Net APIs but it is still in the early stages of development. Using Drive and C# (with .Net) would have limited Ganesha to being a Windows only application.

11 More info at: http://www.gtkperl.org/
12 More info at: http://www.go-mono.com/

The combination of the Java language and Jena RDF API was therefore the most suitable choice for Ganesha. Java provides GUI support within its core libraries which eradicates the need to use external GUI libraries. Furthermore, Java Virtual Machines are freely available for almost all operating system & hardware platform combinations meaning Ganesha's audience is not limited to users of a particular platform. Jena itself also has several advantages:

- It is a mature API that has been under development for over four years and is used by several RDF applications (including IsaViz and RDF Editor).
- It is well documented
- It has an open-source license permitting free distribution of it in source or binary form. This means it can be packaged together with Ganesha to simplify the installation for users.

The choice of Java as the programming language and Jena as the RDF API for Ganesha concluded Objective 1.3 of the project.

Chapter 3. - Design

3.1 Overview

The main focus of this project was to produce an application that makes working with RDF simple enough for average users. Therefore the user interface design was especially important and the program was designed using a top-down approach. The user interface was designed first (Objective 2.1) and the code design followed on from that (Objective 2.2).

3.2 GUI Design

3.2.1 Choice of features

As a general purpose RDF editor, Ganesha needs to be able to open and edit any RDF/XML file. Therefore the user interface needed to be designed to allow simple manipulation of the basic RDF model (covered in chapter 1.1.1). Based on the findings from Objective 1.1.2 (covered in chapter 2.2) resource types, containers and the CC, DC and FOAF vocabularies are frequently used advanced features of RDF so provision in the user interface to simplify working with them was also necessary.

One of the most basic principles in human computer interaction (HCI) is simplicity (IBM, 2004) – a user interface should not be cluttered with rarely used controls. Therefore it was decided that there would not be extra controls within the user interface for other features of RDF such as reification or typed literals. Technically however, creating or viewing RDF data that uses reification or typed literals is still possible within Ganesha (since these concepts are nothing more than resource descriptions with specific properties).

3.2.2 Basic concepts

The overall concept of Ganesha's user interface is to break the RDF data down into smaller parts rather than presenting it all at once as IsaViz or RDF Editor do (by displaying the entire RDF graph or XML code respectively). In Ganesha the user is presented with a list of all currently loaded resource descriptions. Each one can then be edited in its own window. The list also allows resource descriptions to be selected to be removed and allows new resource descriptions to be added. This hierarchical structuring and progressive disclosure of the data reduces the cognitive load on the users since there is less for them to observe and take in at any given moment.

The resource description editing windows each contain a table listing all the resource description's property type and value pairs. Any of the properties can be removed and new ones can be added. The idea is to present the user with a simple form metaphor which is intuitive to understand and interact with, similar to FOAF-o-matic's interface as discussed in chapter 2.3. In order to assign a type to a resource the editing windows provide the user with a list of preset types (from the DC, FOAF & CC vocabularies) as well as a button to add other types by their URI.

Editing the values displays a dialogue window specific to the kind of value – literal, resource or container. Dialogues for editing literals simply provide a text field for the user to input the value. Resource values have a similar dialogue window with a single line text field for inputting the resource's URI. If the resource has a description in the current RDF/XML file, then that resource's description will be displayed in a new editing window instead of the URI dialogue window. Finally, container values have their own specialised editing windows which are similar to the resource editing windows, except that they only list values[13] and do not have resource types.

Adding new properties is handled by a wizard. First it prompts the user to choose a property type from a predefined list (including properties from the DC, FOAF & CC vocabularies) or to add their own by entering the type's URI. Then the user is asked to choose a type of value: literal, resource or container. The last step prompts them to enter a value. If the value is a resource they can either enter its URI or choose any of the other resources described in their current RDF/XML file from a list. If they choose a container type they are then prompted to specify what container type (Bag, Sequence or Alternatives). The new property is then instantly displayed in the resource editing window confirming to the user that their actions were successful.

13 Sequence editing windows feature an additional column showing the indices of the sequence items.

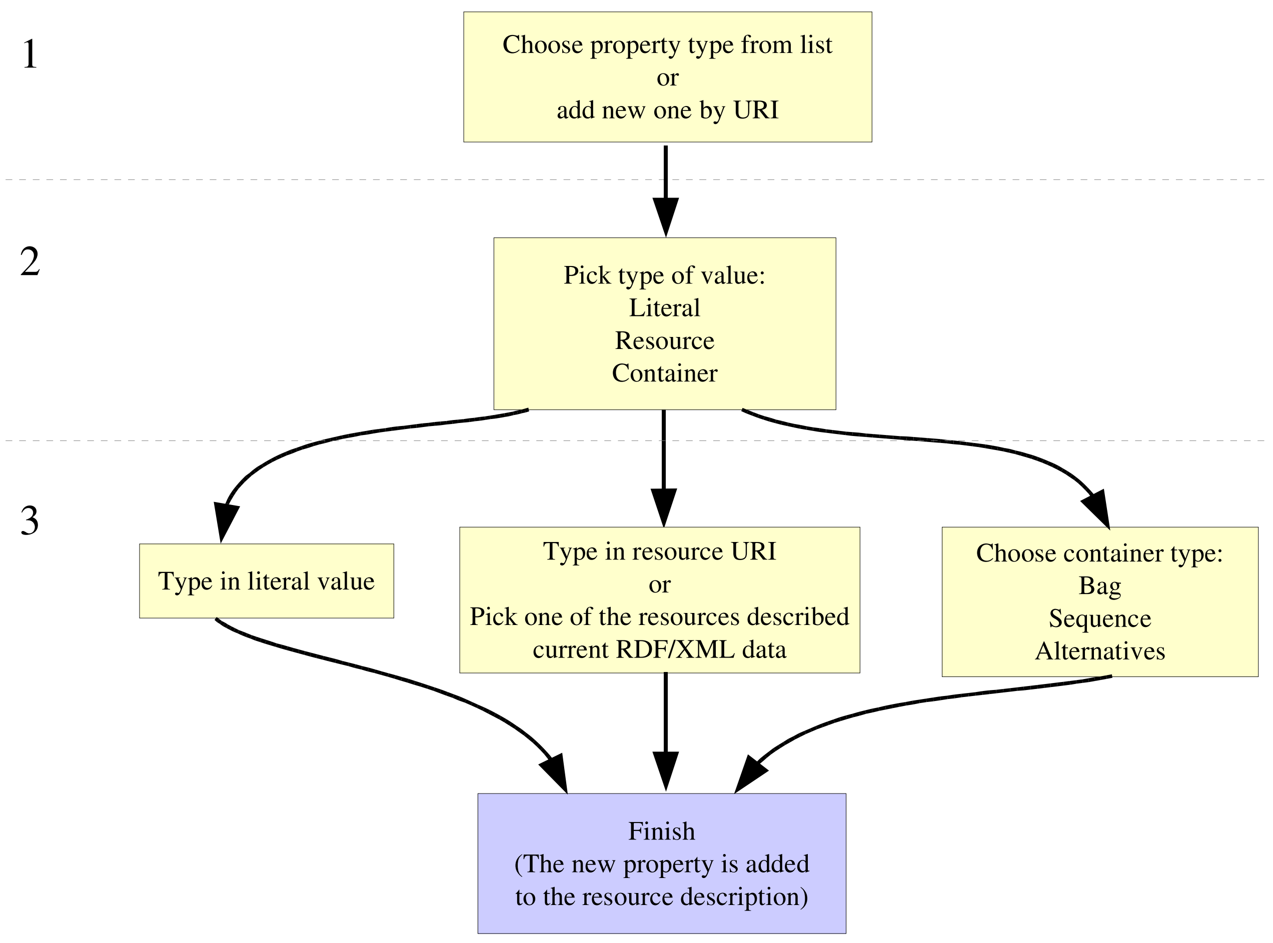

1
2
3
Choose property type from list or add new one by URI
Pick type of value: Literal Resource Container
Type in literal value
Type in resource URI or Pick one of the resources described current RDF/XML data
Choose container type: Bag Sequence Alternatives
Finish (The new property is added to the resource description)

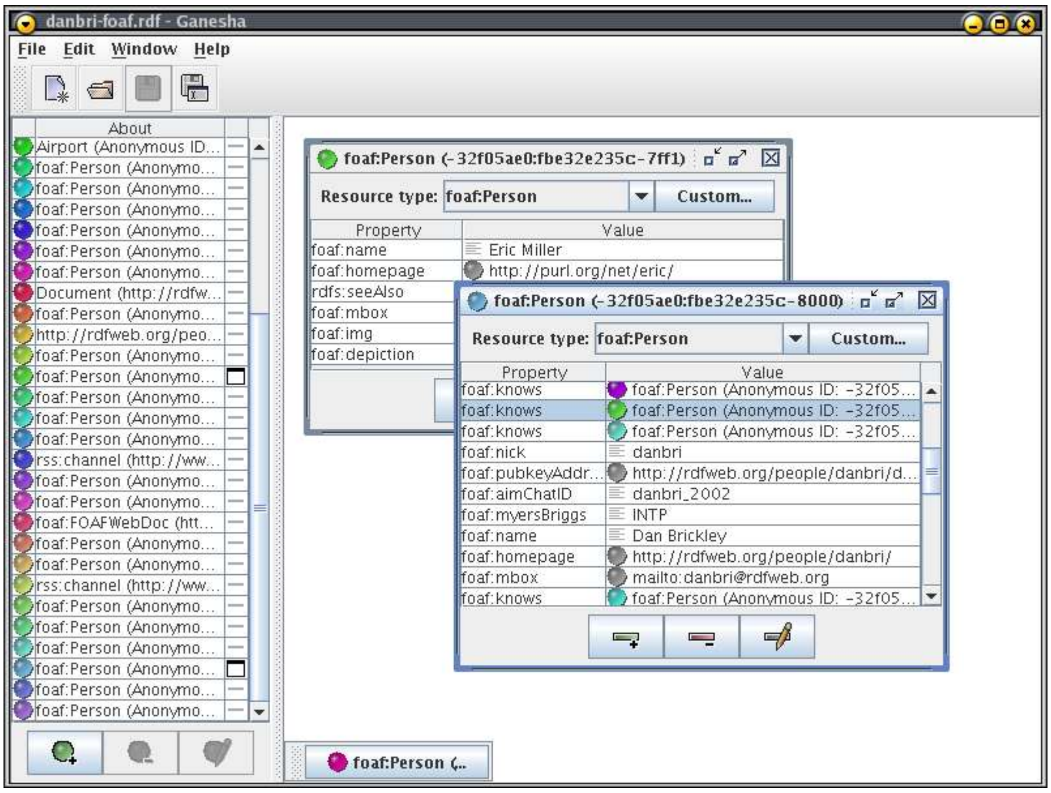
Figure 3-1: "Add property" wizard structure

The wizard incorporates several HCI principles. It guides the user through the process of adding a resource by using constraints: At any given point the user only has options relevant to their task – there are no controls they could wrongly use by mistake. As it only has three steps (see figure 3-1) the wizard also adheres to the “rule of five” (Campbell-Kelly, 2003) which states that users should not need to go through more than five (+/- two) levels of screens or menus to reach a given item in a user interface, since they will have trouble remembering where things are otherwise. It also practices the principle of forgiveness – users can step backwards through the wizard to change or amend any of their previous choices. By not having to worry about making mistakes, users will feel more comfortable exploring the user interface.

When adding a new resource description the same wizard interface is used except that it is preceded by an additional step prompting the user to either provide the URI of the resource to be described or to indicate that it is an anonymous resource description. Adding values to containers is done via a wizard that only uses the last two steps (as no property type is required). Both of these modified wizards are still within the “rule of five”. Furthermore the strong similarities between the wizards adds consistency to the user interface allowing users to apply knowledge gained from experiences using one interface to using another.

3.2.3 Layout

Figure 3-2 shows a screenshot of Ganesha's user interface with a large RDF/XML file loaded. The resource description list is located on the left. The pane on the right contains any open resource or container editing windows. Two resource descriptions are open in editing windows along with a third whose editing window has been minimised.


danbri-foaf.rdf - Ganesha
File
Edit
Window
Help
foaf:Person (-32f05ae0:fbe32e235c-7ff1)
Custom...
foaf:Person (-32f05ae0:fbe32e235c-8000)
Custom...
foaf:Person (...

About
Airport (Anonymous ID...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
Document (http://rdfw...
foaf:Person (Anonymo...
http://rdfweb.org/peo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
rss:channel (http://ww...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:FOAFWebDoc (htt...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
rss:channel (http://ww...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...
foaf:Person (Anonymo...

Resource type: foaf:Person

Property	Value
foaf:name	Eric Miller
foaf:homepage	http://purl.org/net/eric/
rdfs:seeAlso	
foaf:mbox	
foaf:img	
foaf:depiction	

Resource type: foaf:Person

Property	Value
foaf:knows	foaf:Person (Anonymous ID: -32f05...
foaf:knows	foaf:Person (Anonymous ID: -32f05...
foaf:knows	foaf:Person (Anonymous ID: -32f05...
foaf:nick	danbri
foaf:pubkeyAddr...	http://rdfweb.org/people/danbri/d...
foaf:aimChatID	danbri_2002
foaf:myersBriggs	INTP
foaf:name	Dan Brickley
foaf:homepage	http://rdfweb.org/people/danbri/
foaf:mbox	mailto:danbri@rdfweb.org
foaf:knows	foaf:Person (Anonymous ID: -32f05...

Figure 3-2: GUI layout

The visual representation and layout of the interface and its various components can make a big difference to the usability of an application. A good interface must be uncluttered and intuitive to navigate. To this end Ganesha was split into two main panes – the resource description list on the left and space for the editing windows on the right – to provide a visual and logical separation between the contents of an RDF/XML file (the resource description list) and the contents of individual resource descriptions.

The resource list serves as starting point when navigating RDF data, so it was important to position it prominently. Making it a constant part of the application window rather than its own separate window ensured that users could not inadvertently 'loose' it by closing or hiding it. To make the individual resources easier to locate visually each is assigned a differently coloured icon[14]. This is especially useful for differentiating between anonymous resources. Whenever a described resource is referred to in any other part of the GUI (such as the title of an editing window or the value of a property) the same coloured icon is used to allow users to locate it in the resource description list at a glance. The list also features three buttons – for adding, removing or editing descriptions – which have been positioned at the bottom of the list rather than on the toolbar. This was done to make their association with the list more obvious.

The resource editing windows are deliberately laid out similarly to the resource description list since users interact with them in a similar way (see section 3.2.2). This provides consistency throughout Ganesha's user interface to encourage users to apply what they have discovered in one element to another. Icons are used next to the property values to show what kind of values they are (literal, resource or container). The editing windows also contain a drop-down list at the top for selecting a resource type. A drop-down menu was chosen because it automatically implies a “one out of many” choice to the user and is more space saving than alternative widgets such as radio buttons allowing for a less cluttered window. The status of editing windows – open, closed or minimised – is depicted next to the corresponding entry in the resource list by a small icon. This provides users with instant feedback about which resource descriptions are currently open which can be a great help when many editing windows are open at the same time.

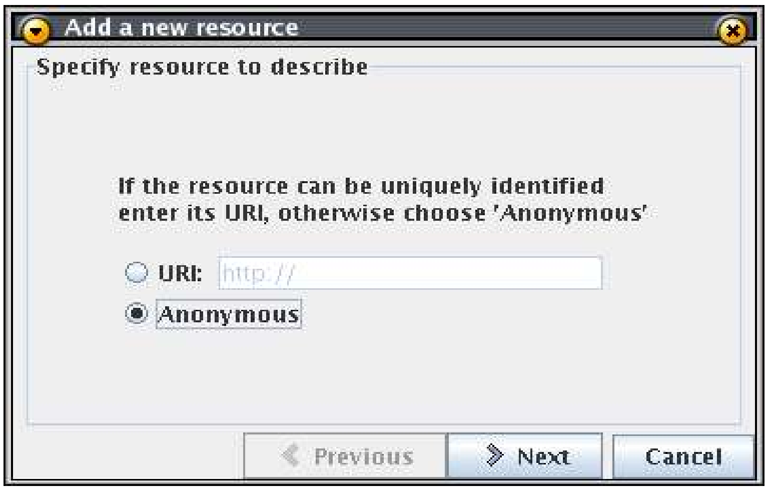
Controls for saving and opening files and quitting the application are located in the menubar and toolbar. Both are persistent elements of the GUI since their controls are frequently used and should therefore be easily accessible by the user. The choice of icons on the toolbar and the menu layout and naming adhere to the recommended “Java Look & Feel Guidelines” (SUN, 2001) so that they are consistent with other Java applications.

14 The colours are chosen randomly by the program since RDF data does not hold any presentational information that could be used to determine the colouring.

3.2.4 Example walk-through

To illustrate the points made about the GUI design in the previous sections this section demonstrates the use of Ganesha's GUI to create the example RDF data used in Figure 2-1 (in chapter 2.1.1):

Once Ganesha has started up the “New file” button ( ) on the toolbar is clicked to begin a new file. The 'Add Resource' button ( ) on the resource description list becomes available and is clicked to start the “Add resource wizard” (notice the input controls of the wizard have been centred to immediately draw the users attention to them and how the URI entry field is disabled when the “Anonymous” option is selected to prevent the user from entering information that is not required):


Add a new resource

Specify resource to describe

If the resource can be uniquely identified
enter its URI, otherwise choose 'Anonymous'

URI: http://

[✓ Anonymous]

Figure 3-3: Creating an anonymous resource description

In order to describe the person “James Nash” the “Anonymous resource” option is chosen. The wizard prompts for an initial property so the “foaf:firstName” property type is chosen and the literal value “James” is input. The new anonymous resource description is added to the list on the left and its contents are displayed in an editing window on the right (Notice the green resource icon in the window title matches that in the resource description list):

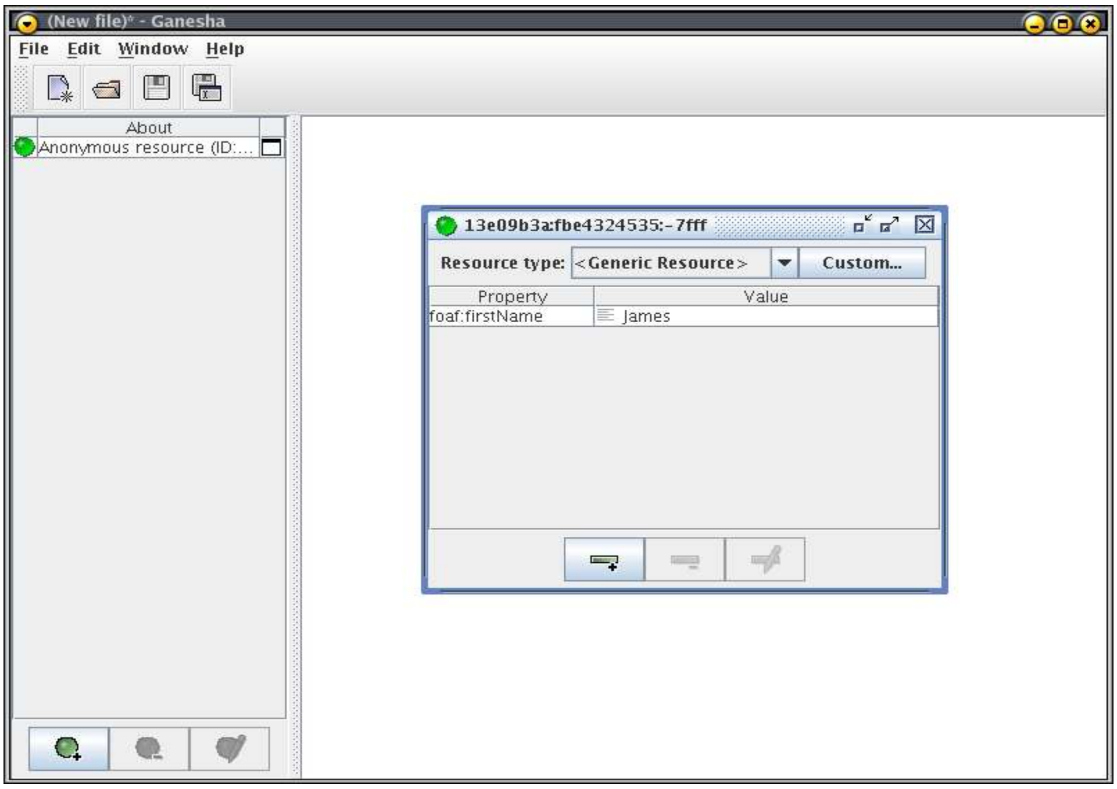


Figure 3-4: Anonymous resource description added

Using the "Add property" button of the editing window the surname and homepage properties are added. Next the type "foaf:Person" is assigned to the resource by selecting it from the resource type list at the top of the editing window (notice the built-in FOAF vocabulary resource types):

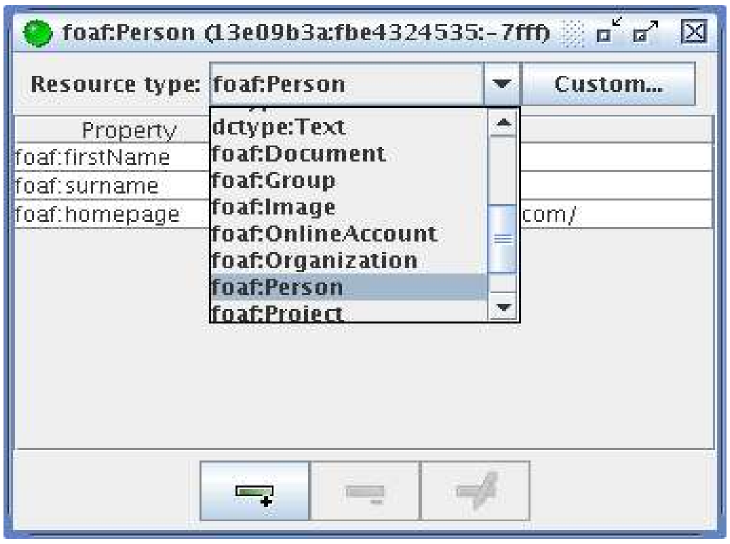
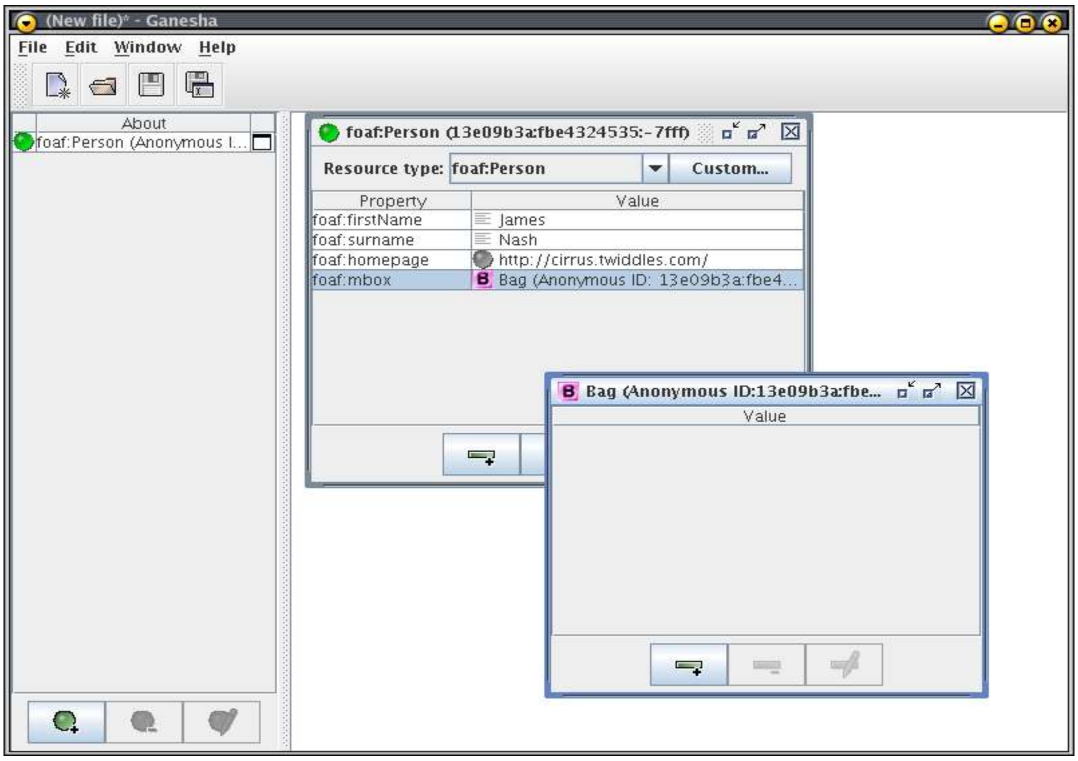


Figure 3-5: Choosing a resource type

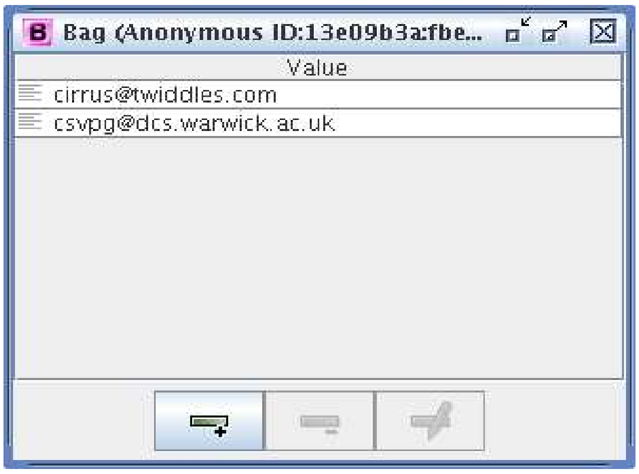
Now the “foaf:mbox” property is added with a bag container as its value. This produces an empty bag container. Choosing to edit it opens a bag container editing window:


(New file)* - Ganesha
File
Edit
Window
Help
About
foaf:Person (Anonymous I...
foaf:Person (13e09b3a:fbe4324535:-7fff)
Resource type:
Custom...
Bag (Anonymous ID:13e09b3a:fbe...
Value

Property	Value
foaf:firstName	James
foaf:surname	Nash
foaf:homepage	http://cirrus.twiddles.com/
foaf:mbox	Bag (Anonymous ID: 13e09b3a:fbe4...

Figure 3-6: Empty bag added

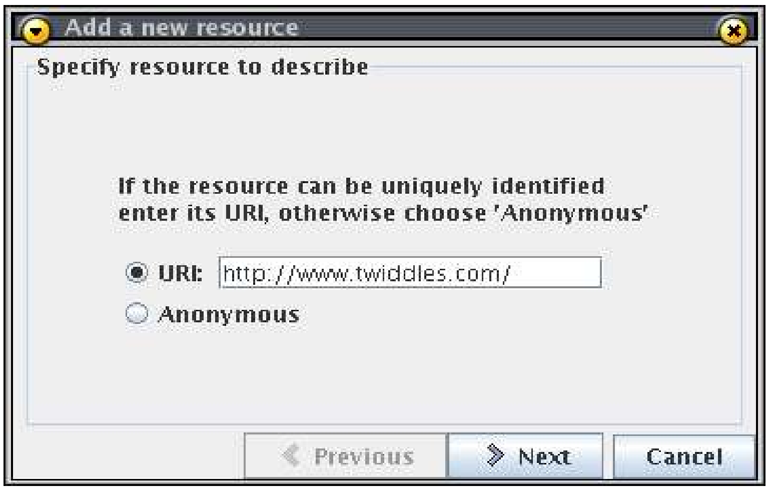
Adding values to it works in much the same way as adding properties to a resource description (see chapter 3.2.2). As the two email addresses are added to the bag they are immediately visible in the bag editing window:



Value
cirrus@twiddles.com
csvpg@dcs.warwick.ac.uk

Figure 3-7: Bag with values

Finally a new resource description for “http://www.twiddles.com” is added. This time the URI is given to the wizard rather than the “Anonymous” option:


Add a new resource

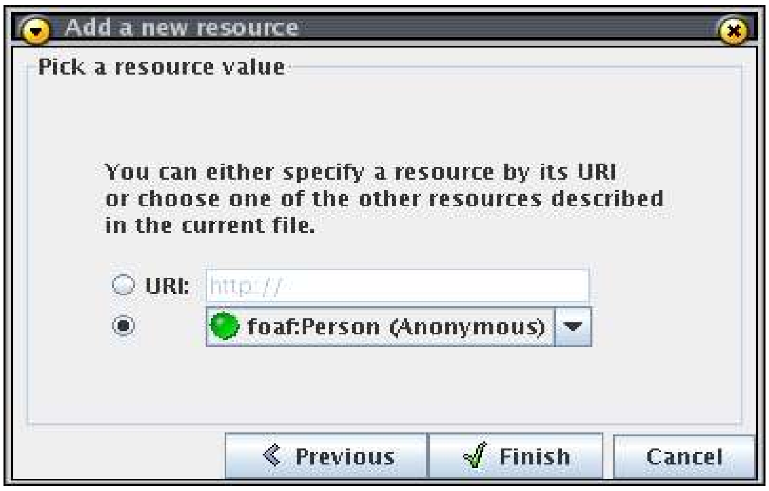
Specify resource to describe

If the resource can be uniquely identified
enter its URI, otherwise choose 'Anonymous'

[✓ URI: http://www.twiddles.com/]
Anonymous

Figure 3-8: Describing a resource with a URI

In the following steps the property type “dc:creator” and the value type “resource” are chosen. The wizard then presents the option to supply a resource URI as the value or to choose the resource that has been previously described. The latter option is chosen. (Notice how the list of other resources also displays their unique, coloured icons):


Add a new resource

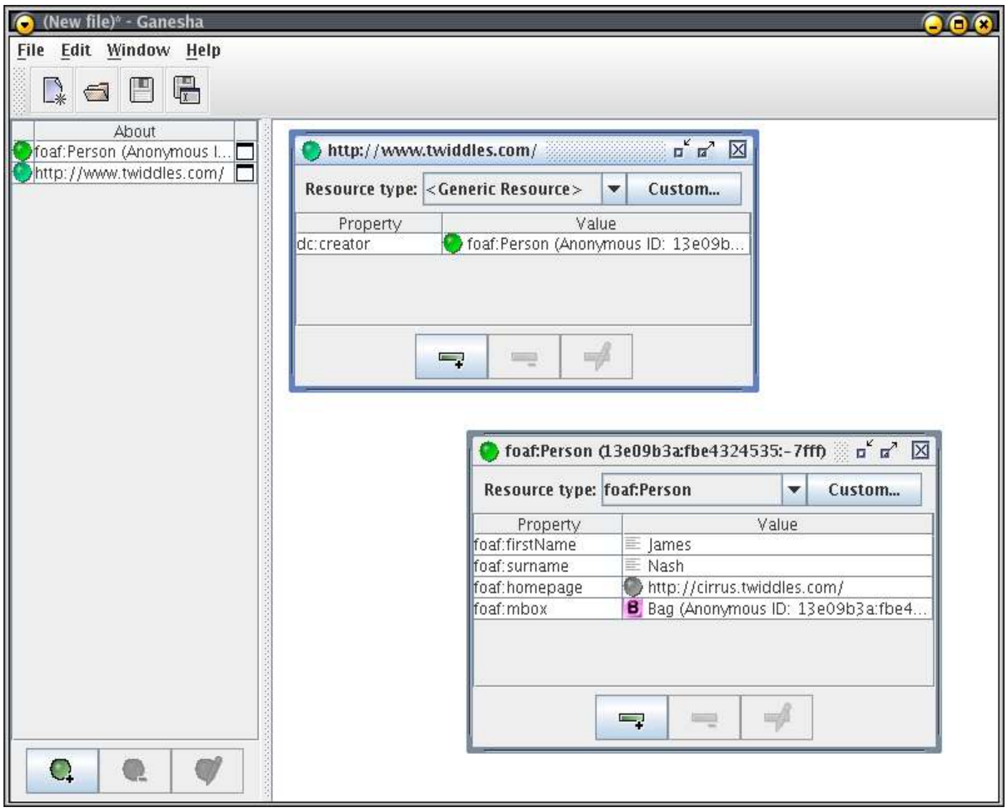
Pick a resource value

You can either specify a resource by its URI
or choose one of the other resources described
in the current file.

URI: http://
[✓ foaf:Person (Anonymous)]

Figure 3-9: Choosing a resource value

The finished resource description then appears in the resource description list and its editing window is displayed. This step completes the creation of the RDF data depicted in figure 2-1:



About
foaf:Person (Anonymous I...
http://www.twiddles.com/

http://www.twiddles.com/

Resource type: <Generic Resource>

Property	Value
dc:creator	foaf:Person (Anonymous ID: 13e09b...

foaf:Person (13e09b3a:fbe4324535:-7fff)

Resource type: foaf:Person

Property	Value
foaf:firstName	James
foaf:surname	Nash
foaf:homepage	http://cirrus.twiddles.com/
foaf:mbox	Bag (Anonymous ID: 13e09b3a:fbe4...

Figure 3-10: Finished RDF file

Clicking the "Save" button prompts the user for a filename using the standard Java SWING file selecting dialogue and then saves the data as RDF/XML. The serialisation is completely transparent to the user.

3.3 Program Design

3.3.1 Structure

After establishing what functions were going to be available to the user in the GUI it was necessary to design the internal methods to provide that functionality. Many RDF functions available to the user such as adding a new resource require several actions to take place internally. Furthermore the same functions may be accessible from different places in the GUI – editing a resource is possible by double-clicking its entry in the resource list or by selecting the entry and clicking the 'Edit resource' button. In these cases the same internal method will be called, which raises the question of where such methods should be located.

The decision was made to structure the code as two distinct parts – the internal RDF and I/O (input / output) functionality and the GUI code – separated by a set of interfaces (illustrated in Figure 3-11). As long as each part only communicates with the other via the interfaces they can each be maintained or changed independently. This greatly simplifies any future modifications to the code and even allows for the internals to be replaced (for example to support an RDF API other than Jena) without changing the GUI code.

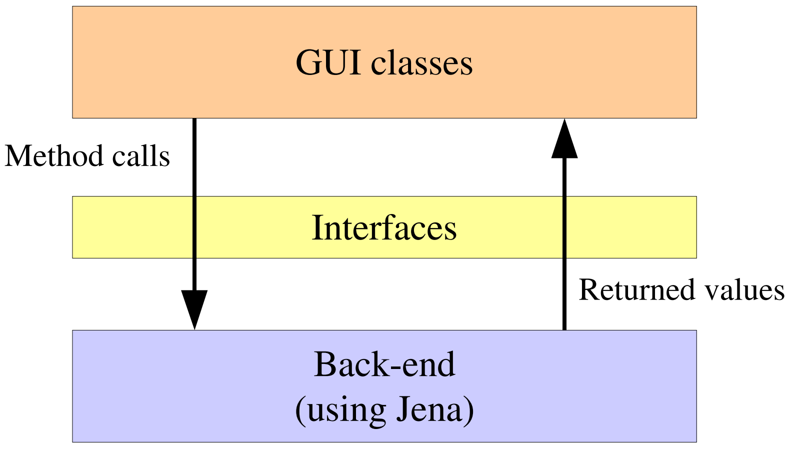


Figure 3-11: Program structure

3.3.2 Interface design

The design of the interface methods was driven by what was required in the GUI. It follows the GUI's approach of presenting the RDF data in a hierarchical form. The RDF data contains resource descriptions, each resource description consists of properties, properties in turn consist of property types (predicates) and values (objects) and objects can be either literal values, resources or containers[15]. Each of these pieces is represented by a Java interface defining methods to interact with it.

- The RDF data interface (`RDFFile`) contains methods to:
  - Read in data from an RDF/XML file
  - Output the current data to an RDF/XML file
  - Clear the current RDF data (to create a new file)
  - Retrieve a list of all resource descriptions in the current data
  - Create a new resource description
  - Fetch or create data needed for creating or modifying properties of resource descriptions such as:
    - A list of known property types
    - A new property type
    - A new resource (description)
    - A new container

- The resource description interface (`RDFResource`) contains methods to:
  - Retrieve a list of properties
  - Add and remove properties
  - Retrieve and change the resource type
  - Delete the entire resource description

15 Of course containers are really just resources with a special property, but since the GUI presents them to the user differently the interface mirrors this.

- The property interface (`RDFProperty`) contains methods to:
  - Retrieve the property type (predicate)
  - Retrieve and change the value (object)

- The property type interface (`RDFPredicate`) contains methods to:
  - Retrieve the URI and name of the type

- The value interface (`RDFObject`) contains methods to:
  - Query the type of object (literal, container or resource)
  - Retrieve the value

- Each of the three containers has its own interface (`RDFBag`, `RDFSeq` and `RDFAlt`) which are all subclasses of `RDFContainer`. They all have methods to:
  - Retrieve the objects in the container
  - Perform container specific actions (such as setting a default item in an alternatives container)

A more detailed description of the interfaces can be found in the code documentation included on the source code CD-ROM (see Appendix III).

3.3.3 The back-end design

The internal functionality classes or the 'back-end' were to use Jena, as determined by Objective 1.3. They had to map the `RDF*` interface methods (covered in the previous section) to Jena's API. Because of how Jena works these classes were going to need to include additional methods to pass references to the underlying Jena objects from one to another. Due to this interdependency alternative Ganesha back-ends would not be able to replace the individual Jena-based classes and would need to re-implement all the `RDF*` interfaces themselves. This implementation restriction has been included into the interfaces' source code documentation (see Appendix III).

3.3.4 GUI code design

The GUI code was divided into classes that correspond to the various elements of the user interface. A main `GUI` class to represent the application window was designed. Its components, the menubar, the toolbar, the resource list and space for the resource windows were all designed as individual classes containing any associated methods. Other GUI components such as the resource and container editing windows were also given their own classes.

The purpose of the main `GUI` class is to display the application window and keep track of certain states that can affect the options available to the user:

- Whether a file is currently open (if there is no file the save operations will be disabled)
- Whether an open file has been edited since its last save (if the file has been edited users will be asked to save if they attempt to close the file or application or try to open a different file)
- The current file's name (if a new file has not been named saving it will prompt the user for a name)

Since the different GUI elements could affect these states the `GUI` class design included public methods that the other classes could call to notify it of any state changes. To facilitate this the constructor methods of the GUI elements needed to take references to the `GUI` class as arguments so that they could access its public methods.

The same mechanism was included in the design of any other GUI classes that needed to notify each other of changes. For example resource windows notify the resource list if they are closed or minimised so that the list can display an appropriate icon. Figure 3-12 shows how the class designs were linked to each other. The direction of the arrows indicates the direction in which references are to be passed down – for example when the `GUI` object instantiates a `ResourceList` object it must pass a reference back to itself to the `ResourceList` constructor.

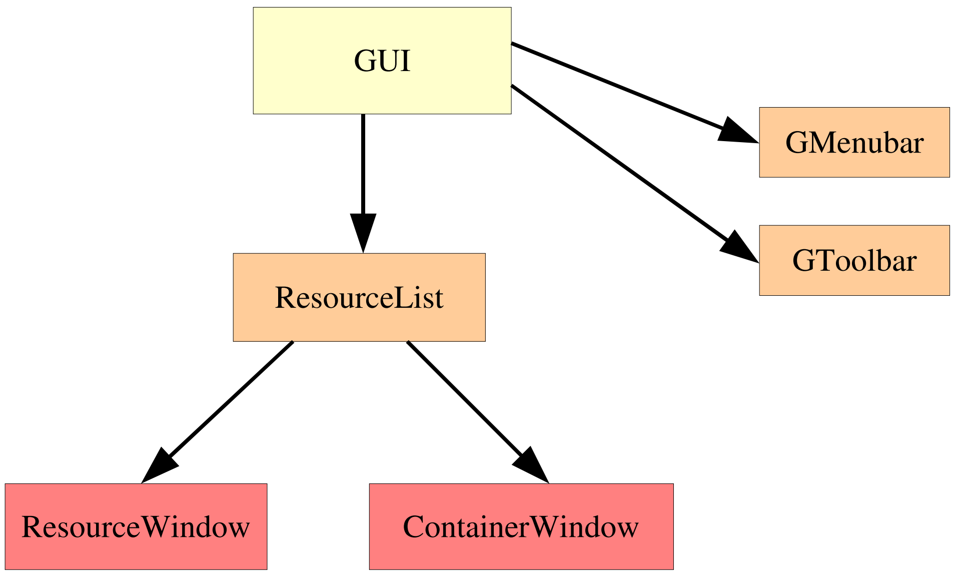


Figure 3-12: GUI component structure

This was still an abstract, high-level design and it was up to the implementation (Objective 3) to make it work using Java and the Java's SWING classes.

Chapter 4. - Development & Testing

4.1 Development Methodology

With the overall design of the GUI and the code in place a suitable methodology had to be found in order to structure the coding and testing. Due to the design of the program (see chapter 3.3.1) the coding was split into two main tasks – writing the back-end functionality classes and writing the GUI classes. The decision was made to develop each half using a spiral approach. On each cycle the details of the inner workings of the classes would be designed or refined, coded, tested and then fixed. The process repeated until the classes were deemed to be working satisfactorily.

A more formalised waterfall approach involving detailed specifications or UML diagrams was deemed inappropriate. This was not a group project so there was no need to create formal designs and diagrams to avoid misunderstandings between multiple developers. Such an approach would have been unnecessarily time-consuming for a one-man project such as Ganesha.

Since the GUI depended on the back-end the back-end was developed before the GUI. Due to this dependency most of the GUI code could not be tested in isolation. Therefore testing it automatically meant testing its integration with the back-end too. Thus Objectives 3.1, 3.2 and 4.1 were passed through twice – once just for the back-end and once for the GUI code together with the back-end.

4.2 Implementation

Writing the interface classes was trivial since the methods they defined had been outlined during the design process (see chapter 3.3.2). Creating the Jena-based back-end involved wrapping the corresponding Jena classes to fit the interfaces. In some cases operations required by the Ganesha interfaces were more complex and required several calls to Jena or augmentations to Jena's functionality (for example: providing user-friendly property or resource type names rather than URIs).

To program the GUI several additional classes had to be created besides the ones outlined in the high-level design (see chapter 3.3.4). These mainly included `ActionListener` classes to process GUI events such as the user clicking a certain button. If the event was unique to one part of the GUI the `ActionListener` was implemented as an anonymous class and if it was an event common to several controls in the GUI it was implemented as a separate class that they could all use. Other additional classes included custom data model and rendering classes for the tables used in the resource description list and editing windows.

All classes and methods were given Javadoc comments describing their purpose and usage. This provided a valuable resource while writing the code and will provide a useful reference for any future maintenance or modification of the source-code.

The code was packaged as `com.twiddles.ganesha` to simplify its distribution as a Java JAR file and to avoid namespace conflicts with any other Java programs users may have installed. In order to keep the source code tidy the decision was made to divide the different parts of the program as sub-packages of `com.twiddles.ganesha`:

- The interfaces and Jena back-end are included in the `functionality` sub-package
- The GUI code is included in the `gui` sub-package
- Graphics and icons used by the GUI classes and associated code in included in the `gui.gfx` sub-package

A class called `Ganesha` was added in the base package for users to invoke in order to start up the Ganesha application. It does a few basic sanity checks (Java version, display available etc.), constructs and `GUI` object and then displays it. It can optionally take a file name as a command-line argument which it then opens within Ganesha immediately.

4.3 Code testing

4.3.1 Overview

The testing of the program code (Objective 4.1) had to take into account the code design's strict separation of back-end and GUI. The task was split into two parts – the testing of the back-end code and the testing of the GUI code. Due to the different purposes of each part different testing strategies were required for each. These are described in the following two sections.

4.3.2 Back-end

The back-end of Ganesha can be regarded as a black box – given certain inputs it produces certain outputs but the means by which it does so is of no consequence. Such programs lend themselves well to automated tests where many different kinds of inputs can be simulated. Two such test programs were written to call the interface methods of the Jena back-end and check the resulting outputs. The first one, `PrintRDF`, took an RDF/XML file as its input and used the back-end classes to read the data and output a textual rendering of it. This could be then compared to the raw RDF/XML code or the output of another utility (such as the W3C's RDF Validator) to verify that the data had been read in and processed correctly.

```
About: http://www.twiddles.com/
   dc:creator = --> Anonymous ID: 8073...

About: Anonymous ID: 8073 (Type: foaf:Person)
   foaf:firstName = James
   foaf:mbox = BAG (anonymous):
      1) cirrus@twiddles.com
      2) csvpg@dcs.warwick.ac.uk
   foaf:homepage = --> http://cirrus.twiddles.com/...
   foaf:surname = Nash
```

Output 4-1: PrintRDF listing

The above listing shows the output of `PrintRDF` when given the RDF/XML found in output listing 2-2 (Chapter 2.1.2). Each resource description is listed with all its properties. As can be seen all properties are present and their property types and values have been correctly identified.

The second test program, `BuildRDF`, builds up an RDF model from scratch using the back-end methods and prints it out in the same format as `PrintRDF`. In the process of creating the RDF data `BuildRDF` calls every method defined by Ganesha's interfaces. The RDF model it produces is hard-coded into `BuildRDF` so that the output of running it can be compared to the expected output.

Together the two test applications could emulate all the RDF operations that the GUI could perform (as listed in chapter 3.3.2). After each revision of the back-end code these tests were run in order to establish that all parts were producing the correct outputs and working together correctly. Using this approach many bugs were located and eradicated during the development iterations.

4.3.3 GUI code

The GUI code did not lend itself well to automated testing due to the complexity of the inputs and outputs. There are no simple values such as strings or numbers to input or verify within the output as there are with the back-end, the inputs to the GUI are mouse clicks and key presses and the outputs are visual elements on the screen. These are far easier for a human to test rather than a machine so a different approach was used. The possible states that the GUI or certain parts of it could be in were determined. Any states that were supposed to be reachable from a given other state were also determined. To GUI was then manually tested to ensure that each state behaved as expected and only allowed the transition to other states that were supposed to be accessible from it.

The diagram below shows a selection of some of the states of the main GUI window and the properties each was expected to have (depicted by the rectangles)[16]. The transitions sometimes include prompts that prevent the user from accessing the rest of the GUI until they have been answered or cancelled. These are known as modal prompts and are depicted by the ovals.

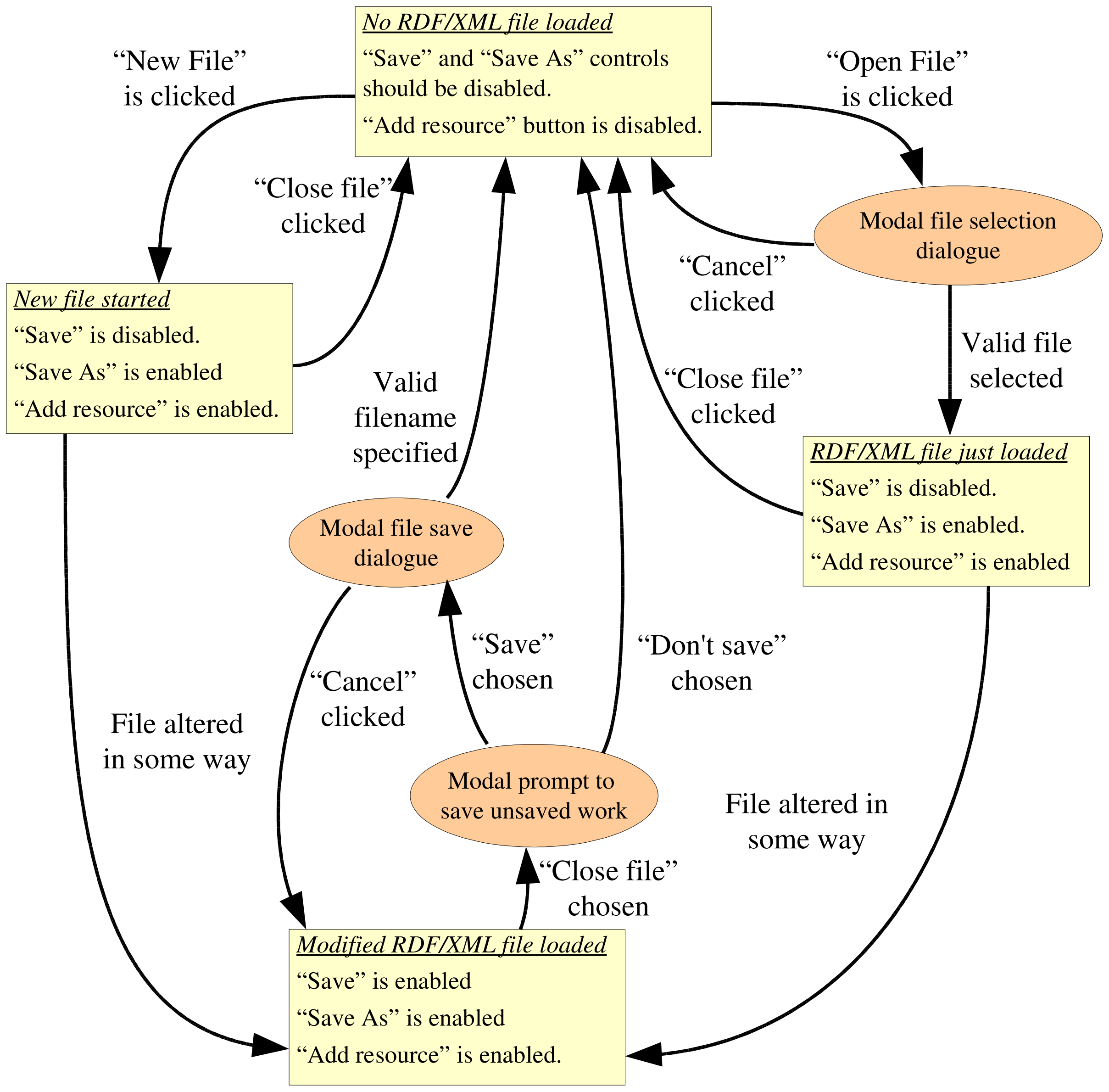


Figure 4-1: Selected GUI states

Using this approach the various GUI states could be tested to confirm that everything was working both functionally and internally at the source-code level. Many GUI bugs and mistakes were found and eliminated over the various development iterations.

16 This diagram is not an exhaustive list of all the states and transitions that were tested

4.4 GUI Usability testing

In order to evaluate how users found using Ganesha's GUI some focus group testing was undertaken. Once the GUI code had reached a mature state and the important features were working a group of testing volunteers who matched the description of Ganesha's target users was selected. Each was asked to read the Ganesha user manual (see appendix II). Then they were given tasks to perform using the GUI without any additional assistance. They were asked to speak out aloud their impressions and considerations while they were attempting the tasks. Any steps that they had trouble with or found confusing were noted. The feedback gained from observing the test subjects during these usability tests was then used to refine and tweak the design of the user interface.

For example, the colouring of the resource icons (which occurs in the order they appear in the rainbow) originally started with a shade of red. On a file with three resource descriptions this caused the resource list to contain a red icon followed yellow one and a green one. Because the result looked like a traffic light one user was mislead into thinking there was a special meaning to the colours of the resource. As a result of this finding the starting colour was changed to a shade of green which solved this problem. Most other alterations to the GUI involved changing the names and labels of controls in the GUI to make them more intuitive.

Chapter 5. - Conclusions

5.1 Summary

The Ganesha project aimed to create a simple RDF/XML editor in order to aid the development of the Semantic Web. Background research was undertaken to establish what aspects of RDF would be most useful and most difficult for the target users. Other RDF editors were also evaluated. Based on the results thereof a simple user interface was designed that followed established human computer interaction (HCI) principles.

Internally the application was structured as two halves – the GUI code and the back-end code that provided the actual RDF functionality. Each part was built and tested individually. All the features that were planned in the design were implemented successfully and within the timetabled deadlines.

Designing a user-oriented tool from scratch was a challenging yet rewarding experience. The project made use of several disciplines that had been taught on the Computer Science course: Programming, HCI & Software Engineering. Applying the theory from those courses to a real-world project was an effective learning experience and has a provided the author with a valuable insight into the process of creating software.

5.2 Self Assessment

5.2.1 What is the technical contribution of this project?

The Ganesha project has resulted in a useful piece of software that succeeds at making RDF/XML creation more accessible to web designers than has so far been possible. Some interesting applications that use RDF already exist, such as the FOAF co-depiction experiment (Brickly, 2002), and in future many more will appear. Ganesha allows web designers to make the best of such RDF applications and thus represents an increasingly useful tool for them.

5.2.2 Why should this contribution be considered important to computer science?

Due to its user-friendliness Ganesha will hopefully encourage many new designers and publishers to create RDF data. It is therefore a valuable contribution to the development of the Semantic Web, which in turn will open up many new exciting possibilities for developers and users alike to explore.

5.2.3 How can others make use of this project?

The Ganesha application and its source code will be released on the project website[17] under the GNU General Public License[18]. Anyone will be able to download and use the tool free of charge. Furthermore they will have the freedom to view and modify the source-code, provided they share their changes under the same conditions.

5.2.4 Why should this project be considered an achievement?

This project was an original idea of the author (as opposed to one suggested by the department) and is an achievement in itself as an example of the objectives being set and carried out in within the allowed time. The Ganesha application has received positive feedback from everyone who has used it.

5.2.5 What are the weaknesses of this project?

Ganesha succeeds at being more usable for average web designers than competing RDF editors but in order to produce sensible RDF data it still requires users to read the manual and familiarise themselves with the meanings and purposes of the various property and resource types (see appendix II). Ideally Ganesha should automatically limit and/or check input values to ensure that property types are being used as intended. Such functionality would require the addition of an RDFS (the language used to define vocabularies) parser to Ganesha which is beyond the scope of this university project but may be added in future releases of Ganesha.

17 http://event-horizon.kicks-ass.net/uni/ganesha/
18 http://www.gnu.org/licenses/gpl.html

5.3 Future developments

With the completion of this project the Ganesha application has reached version 1. That is by no means the end of Ganesha's development though – RDF is a constantly evolving standard and Ganesha will need to evolve with it. Short-term issues that the continued development of Ganesha should address include:

- Creating OS-specific installers or packages to simplify the distribution and installation of Ganesha (Possibly using Apache ANT, a tool for packaging Java applications)
- Applying for development hosting at GNU Savannah. This would provide tools such as CVS, mailing-lists and bug tracking tools and also give Ganesha extra publicity.

In the longer term Ganesha's features could be extended. Possible enhancements could include:

- The addition of an RDFS parser that allows vocabularies to be added and values input by the user to be type checked according the vocabulary's specification.
- The ability to automatically extract meta-data from common file formats such (X)HTML, PDF, JPEG, OGG etc. Since the kind of meta-data that can be extracted from different formats varies and may only be suitable for specific vocabularies this could perhaps be implemented using a plug-in mechanism. Different plug-ins could then be written for different file formats.

The contribution of such enhancements to Ganesha may even be suitable as a third year project ideas for future generations of students.

Acknowledgements

This project has benefited from the contributions of several individuals. I would like to thank:

- Dr. Stephen Jarivs for his supervision of the project and advice
- Alexander Forrest, Deniz Özsen, Jonathan Turnbull, and Raghu Narayanan for participating in the usability tests and providing valuable feedback

I am also grateful to several organisations without whom, although they were not directly involved, this project would never have been possible:

- The World Wide Web consortium for developing the RDF and XML standards
- The Jena project for developing the RDF API used in Ganesha
- The Free Software Foundation and open-source community whose software was used extensively to produce Ganesha and this report

Bibliography

Berners-Lee T. et al (2001). *The Semantic Web*. Available online at: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21 [Accessed on: 23.3.2004]

Brickly, D. (2002). *Photo metadata: the co-depiction experiment*. Available online at: http://rdfweb.org/2002/01/photo/ [Accessed on: 22.4.2004]

Campbell-Kelly, M. (2003). *HCI Lecture 4: Attention & Memory*. Available online at: http://www.dcs.warwick.ac.uk/~mck/Campus/CS231/Not [Accessed on: 11.4.2004 (Access restricted to Warwick campus)]

Dodds, L. (2002). *FOAF-o-matic*. Available online at: http://www.ldodds.com/foaf/foaf-a-matic.html [Accessed on: 30.3.2004]

IBM (2004). *Design Basics*. Available online at: http://www-306.ibm.com/ibm/easy/eou_ext.nsf/Publis [Accessed on: 11.4.2004]

Pietriga, E. (2003). *IsaViz Overview*. Available online at: http://www.w3.org/2001/11/IsaViz/ [Accessed on: 30.3.2004]

Powers, S. (2003). *Practical RDF* (O'Reilly & Associates).

Punin, J. (2002). *RDF Editor*. Available online at: http://www.cs.rpi.edu/~puninj/rdfeditor/ [Accessed on: 30.3.2004]

SUN MICROSYSTEMS (2001). *Java Look and Feel Design Guidelines*. Available online at: http://java.sun.com/products/jlf/ed2/book/ [Accessed on: 11.4.2004]

W3C (2004a). *RDF/XML Syntax Specification (Revised)*. Available online at: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-2 [Accessed on: 22.3.2004]

W3C (2004b). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. Available online at: http://www.w3.org/TR/2004/REC-rdf-concepts-2004021 [Accessed on: 26.3.2004]

W3C (2004c). *Resource Description Framework (RDF) website*. Available online at: http://www.w3.org/RDF/ [Accessed on: 17.3.2004]

Appendix I: Further reading

This section is intended to provide interested readers with useful material to further their knowledge about RDF, the Semantic Web and related technologies. The sources listed here are ones that were not referenced by this report and hence not listed in the bibliography but which are still relevant and useful.

Books

"*XML in a Nutshell*" by Elliotte Rusty Harold & W. Scott Means (published by O'Reilly and Associates) – A great introduction and reference book for working with XML. Recommended for those interested in gaining a greater understanding of the RDF/XML serialisation.

"*Content Syndication with RSS*" by Ben Hammersly (published by O'Reilly and Associates) – Detailed explanations of the various RSS standards, including version 1.0 which is based on RDF. As discussed in chapter 2.2, RSS is not particularly relevant to Ganesha, but it is nonetheless a very popular format and worth knowing about.

Websites

"*Dave Beckett's RDF Resource Guide*" (available at: http://www.ilrt.bris.ac.uk/discovery/rdf/resources/) – A very large collection of links to articles, tutorials and tools for RDF.

"*Friend of a Friend developer site*" (available at: http://rdfweb.org/) – News, information and discussions about the FOAF vocabulary for developers. Essential reading for anyone wanting to write applications that create or use the FOAF vocabulary.

"*Jena documentation*" (available at: http://jena.sourceforge.net/documentation.html) – Documentation and tutorials for the Jena API. Potential Ganesha developers should familiarise themselves with this.

Appendix II: User Manual

1 - Introduction

RDF/XML is a standardised format for storing meta-data (descriptive data about data). It can be thought of as a highly advanced replacement for HTML meta tags, MP3 ID3 tags or any other similar meta-data formats. It can be used to describe any kind of electronic file or real world object as well as the relations between them in form that is convenient for processing by a computer (a search engine for example).

Anything that is being described in an RDF/XML file is referred to as a *resource*. An RDF/XML description of a resource consists of any number of properties of that resource. Each property has a type and a value. For example, if we wanted to state that the author of http://www.xyz.com/ is Mr X then “http://www.xyz.com/” would be the resource, “author“ the property type and “Mr X” the value.

Ganesha is utility that aims to make it as simple as possible to create and edit RDF/XML documents. It has built in support for many property types that are frequently used by web designers but new ones can be added by users at any time.

2 - Installation

2.1 - System requirements

- Java Runtime Environment (JRE) 1.4 or higher
  (available for free from http://java.sun.com/)

To determine if you have Java already installed on your system open a terminal (Unix & MacOS X) or command prompt (Windows) and type: `java -version`

If a message similar to: `java version "1.4.0"` with a version number equal to or higher than 1.4 appears then you already have a suitable Java installed. If a lower version number or an error (such as “Command not found”) appears you will need to download and install Java.

- Jena 2.0 or higher
  (available for free from http://jena.sourceforge.net/)

2.2 - Downloading and installing Ganesha

Ganesha can be downloaded from http://event-horizon.kicks-ass.net/uni/ganesha/download/

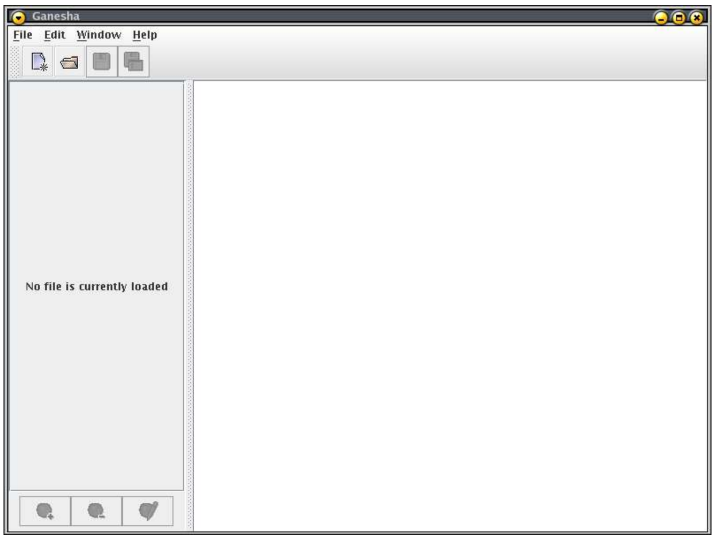
Choose the appropriate package for your operating system and follow the installation instructions on the website.

3 - Creating and editing RDF/XML data

3.1 - Starting Ganesha

Once Ganesha is installed you should be able to start it from your system's program menu. Alternatively you can open a terminal / command prompt and type: `ganesha` to start it.

You will be presented with a window like this (the colours may vary on different systems):

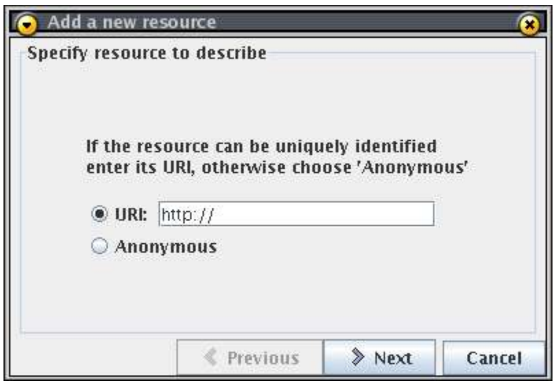


Screenshot 3-1: Ganesha at start up

The pane on the left is the *resource list* and displays all resources in a file that have descriptions. The larger pane on the right is where *resource windows* appear to let you edit resource descriptions.

3.2 - Making a new RDF/XML file

To start a new RDF/XML file either click the 'New File' button ( ) on the toolbar or choose 'File > New' from the menubar. The 'Add Resource' button ( ) at the bottom of the *resource list* will now be enabled – clicking it pops up the *resource adding wizard* which allows you to add new resources (things) that you wish to describe:



Add a new resource

Specify resource to describe

If the resource can be uniquely identified
enter its URI, otherwise choose 'Anonymous'

● URI: http://
○ Anonymous

Screenshot 3-2: Resource adding wizard

If what you are describing has a unique resource identifier (URI) – such as page or file on the Web, or book with an ISBN number – type the URI in the field provided and click 'Next'. If your resource cannot be uniquely identified by a URI – for example a person – choose the 'Anonymous' option and continue.

Next you must choose a property type from the list provided and click 'Next'. (The meanings of the various types are explained in chapter 5 of this manual. If you need to add a property type that is not in the list you may do so by clicking the 'Custom type...' button)

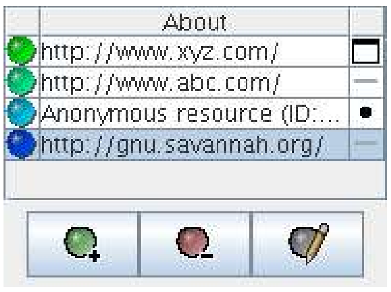
You will now be able to choose one of 3 types of value: Literal, Resource and Container. If the value of your property is a simple text value (such as the title or description of a document) choose 'Literal'. If the value is itself another resource (for example a web-page) choose 'Resource'. Finally, if there are several values for your property (for example multiple email addresses for the same person) choose 'Container'. Once you have chosen an appropriate type click 'Next' to continue.

If you chose 'Literal' you can now type in the text value of your property. If you chose 'Resource' you can key in the resource's URI or choose one of the resources that you have described in the same RDF/XML file (if there are any). Clicking 'Finish' will close the *wizard* and your resource will appear in the *resource list*.

If you chose 'Container' you will need to choose a type of container: Use 'Bag' if you have several values that are not in any special order; use 'Sequence' if your values have a set order; use 'Alternatives' if your values represent multiple choices where one is the default choice. Once you have chosen a type and clicked 'Finish' the resource description will be created with an empty container and the *wizard* will close down. You may now add values to the container by editing the resource you have just created.

3.3 - Editing a resource description

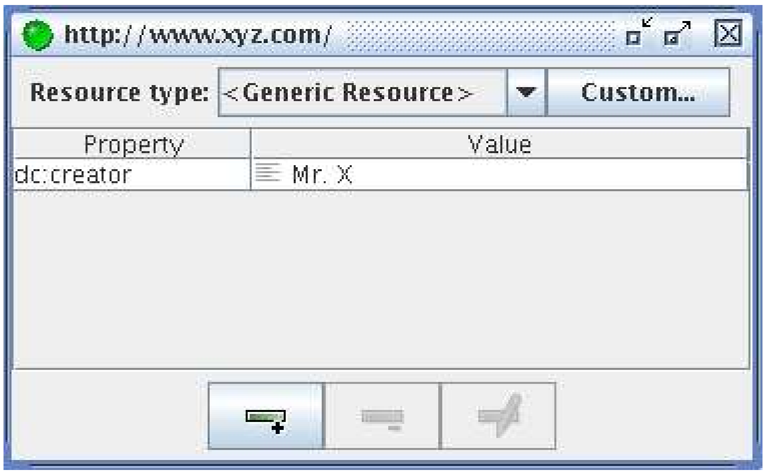
Any resources that you are describing in your RDF/XML file will be listed in the *resource list* along with a coloured *resource icon* on the left, and an *editing window status icon* on the right. The colours of the *resource icons* are chosen at random and do NOT have any special meaning – they are purely a visual aid to help differentiate the resources and navigate between them. Whenever one of your described resources is being referred to, its icon will be displayed alongside it so that you can see at a glance which entry in the *resource list* it corresponds to. The *window status icon* simply indicates whether a given resource description is currently open in an *editing window* (☐), minimised (●) or not open at all (—).



	About	
🟢	http://www.xyz.com/	☐
🟢	http://www.abc.com/	—
🔵	Anonymous resource (ID:...	●
🔵	http://gnu.savannah.org/	—

Screenshot 3-3: Sample resource list

You can delete resource descriptions by selecting them and clicking the 'Remove resource' button ( ). To view or edit a resource's description simply double-click its entry in the *resource list* or select it and click the 'Edit resource' button ( ). This will pop up a *resource editing window*.



Property	Value
dc:creator	Mr. X

Screenshot 3-4: Resource editing window

A *resource editing window* behaves in much the same way as the *resource list*. The three buttons at the bottom allow you to add, remove and edit properties of the resource description. Property values may also be edited by double-clicking them. If the value happens to be a resource that is also described in the same RDF/XML file then editing it will open its corresponding *resource editing window*.

Adding new properties pops up the *property adding wizard* which is identical to the *resource adding wizard* (covered in the previous section) with the exception that the first screen (where a resource is specified) is omitted. You may add any number of properties to a resource description.

You can also assign a type to a resource. This allows you describe what kind of thing your resource is. To do this simply choose an appropriate type from the pull-down list at the top of the *editing window*. Ganesha has several common resource types built in (covered in chapter 5) but you may add more by clicking the 'Custom...' button next to the list of built in types. To remove a resource type simply pick the '<Generic Resource>' option from the list.

3.4 - Editing container contents

When editing property values that are containers a *container editing window* will appear. It is structured very similarly to the *resource editing windows* except that it does not have a resource type column and only lists values, not property types (except when editing sequences - then there is an extra column indicating each element's position in the sequence). The three buttons at the bottom allow you to add, remove or edit values, just as you would in a *resource editing window*. If the container happens to be a sequence there will be two extra buttons at the bottom for moving a selected value up ( ) or down ( ) in the sequence order. Likewise *editing windows* for alternatives containers have an additional 'Make default button' ( ) to make the selected value the default alternative (The default value appears at the top of the list and is highlighted green).

4 - Saving and publishing your work

Once you have described your resources you can save your work by clicking the save button on the toolbar ( ) or choosing 'File > Save' from the menubar. You will be asked for a file name. You may use any file name and extension, however it is custom to use the file extension '.rdf' for RDF/XML files.

If you are using Ganesha to generate RDF/XML files related to a website of yours, you can use Ganesha to automatically insert a hidden link to an RDF/XML file into an XHTML webpage for you. To do this save the RDF/XML file to the same directory as the XHTML file into which you want to insert the link. Then choose 'Edit > Add this RDF to a webpage...'. Enter the XHTML file's name into the window that appears and click the 'Insert Link' button. Your webpage will then contain a hidden link to the RDF/XML file allowing search engines and other application to discover it automatically.

To make your work publicly accessible you should upload your RDF/XML files (and any XHTML files you have added links to) to your webspace using your preferred method (such as FTP). You should also ensure that your web server has the correct MIME-type settings for RDF/XML files. The MIME-type for these files should be set to: 'application/rdf+xml'. Please consult your web host or server documentation if you are unsure how to do this.

5 - Resource and property types

In order to allow computers to process RDF data in a sensible way standard sets of property and resource types and their intended meanings (referred to as *vocabularies*) need to be defined and adhered to. Ganesha has several types from popular vocabularies built in. The following tables describe their intended purposes and limitations so that you can make the best use of them.

5.1 - Creative Commons

Creative Commons (http://www.creativecommons.org/) is an initiative to encourage artists, authors, film-makers and musicians to license their works in a liberal way. They have produced a vocabulary for describing the licensing restrictions of a copyrighted work. It is mainly used for Creative Commons licenses but could in theory be used to describe any kind of licensing restrictions.

Table 5-1: Supported Creative Commons resource types

Resource type	*Intended usage*
cc:Agent	Someone or something that does something. (Eg. The creator of a **cc:Work**)
cc:License	A copyright license description containing **cc:permits**, **cc:prohibits** and **cc:requires** properties
cc:Work	A piece of Work that has an associated copyright license. (Eg: A picture, piece of music, text etc.)

Table 5-2: Supported Creative Commons property types

Property type	*Intended usage*	*Allowed within*	*Allowed values*
cc:license	To point to a copyright license for a resource	Any resource description	Resource of type: **cc:License**
cc:permits	To indicate what a copyright license permits people to do with a certain work	**cc:License** resource descriptions	One the following resources: **http://web.resource.org/cc/Reproduction** **http://web.resource.org/cc/Distribution** **http://web.resource.org/cc/DerivativeWorks**

Property type	*Intended usage*	*Allowed within*	*Allowed values*
cc:prohibits	To indicate what a copyright license prohibits people from doing with a certain work	**cc:License** resource descriptions	**http://web.resource.org/cc/CommercialUse**
cc:requires	To indicate any restrictions that a copyright license of a certain work imposes	**cc:License** resource descriptions	**http://web.resource.org/cc/Notice** **http://web.resource.org/cc/Attribution** **http://web.resource.org/cc/ShareAlike**

Further details are available at: http://creativecommons.org/technology/developerdocs

5.2 - Dublin Core

The Dublin Core meta-data initiative (http://www.dublincore.org/) have created a vocabulary to describe typical attributes of documents such as title, description, authors etc.

Table 5-3: Supported Dublin Core property types

Property type	*Intended usage*	*Allowed within*	*Allowed values*
dc:contributor	Someone or something that contributed to the content of the resource	Any resource description	Any value
dc:coverage	The geographical (or temporal) coverage of the resource	Any resource description	A literal value representing a location ID such as those defined at: http://www.getty.edu/research/tools/vocabulary/tgn/
dc:creator	Someone or something that made the content of a resource	Any resource description	Any value
dc:date	The date when the resource was created	Any resource description	A literal value in the ISO 8601 date format: YYYY-MM-DD
dc:description	A verbal description of a resource (an abstract for example)	Any resource description	Any value

Property type	*Intended usage*	*Allowed within*	*Allowed values*
dc:format	File format of the resource	Any resource description	A literal value representing a valid MIME type as defined at: http://www.isi.edu/in-notes/iana/assignments/media-types/media-types Eg: text/html
dc:language	The language of the resource's content	Any resource description	A literal value representing a language code as defined by RFC 3066 at: http://www.ietf.org/rfc/rfc3066.txt
dc:publisher	Someone or something that made the resource available	Any resource description	Any value
dc:relation	A reference to a related resource	Any resource description	Any resource
dc:rights	Copyright information about the resource	Any resource description	Any value
dc:source	A reference to another document that the resource is derived from	Any resource description	Any value
dc:subject	Keywords indicating topics that the resource is related to	Any resource description	Any value
dc:title	The name of a resource	Any resource description	Any value
dc:type	The type of thing the resource is	Any resource description	One of the resources listed on http://dublincore.org/documents/dcmi-type-vocabulary/ Eg: http://dublincore.org/documents/dcmi-type-vocabulary/Image

Further details are available at: http://www.dublincore.org/documents/dces/

5.3 - Friend of a Friend

The Friend of a Friend (FOAF) project (http://www.foaf-project.org/) has defined a vocabulary to describe people, the relations between them and things they are involved in.

Table 5-4: Supported Friend of a Friend resource types

Resource type	*Intended usage*
foaf:Document	To indicate that a resource is some kind of document
foaf:Group	To indicate that a resource represents a group of people, organisations or other groups
foaf:Image	To indicate that a resource is an image of some kind (Eg: a JPEG file)
foaf:OnlineAccount	To indicate that a resource is an online account held by a person
foaf:Organisation	To indicate that a resource is a social institution of some kind (Eg: a company)
foaf:Person	To indicate that a resource is a person
foaf:Project	To indicate that a resource is a project that a person is involved in

Table 5-5: Supported Friend of a Friend property types

Property type	*Intended usage*	*Allowed within*	*Allowed values*
foaf:accountName	The name or user ID of an online account	**foaf:OnlineAccount** resource descriptions	Any literal value
foaf:accountServiceHomepage	The homepage of an online service providing an online account	**foaf:OnlineAccount** resource descriptions	Resources of type: **foaf:Document**
foaf:aimChatID	A person's AIM chat ID	**foaf:Person** resource descriptions	Any literal value
foaf:currentProject	A project that a person is currently involved with	**foaf:Person** resource descriptions	Any resource
foaf:depiction	To point to a depiction of the resource	Any resource description	Resources of type: **foaf:Image**
foaf:depicts	To indicate what an image resource is depicting	**foaf:Image** resource descriptions	Any resource
foaf:firstName	To indicate a person's first name	**foaf:Person** resource descriptions	Any literal value
foaf:holdsAccount	To indicate an online account that a person or group holds	**foaf:Person** or **foaf:Group** resource descriptions	Resources of type: **foaf:OnlineAccount**

Property type	*Intended usage*	*Allowed within*	*Allowed values*
foaf:homepage	The homepage of the resource	**foaf:Person**, **foaf:Project** or **foaf:Group** resource descriptions	Resources of type: **foaf:Document**
foaf:icqChatID	A person's ICQ chat ID	**foaf:Person** resource descriptions	A literal value representing the person's ICQ number
foaf:img	A particularly representative depiction of a person	**foaf:Person** resource descriptions	Resources of type: **foaf:Image**
foaf:jabberID	A person's Jabber chat ID	**foaf:Person** resource descriptions	A literal values representing the person's Jabber ID
foaf:knows	To indicate someone known by the person being described	**foaf:Person** resource descriptions	Resources of type: **foaf:Person**
foaf:member	A member of a group	**foaf:Group** resource descriptions	Resources of type: **foaf:Person**, **foaf:Organisation** or **foaf:Group**
foaf:mbox	A person's email address	**foaf:Person** resource descriptions	The persons email as a resource using the mailto: URI scheme (Eg: mailto:someone@example.com)
foaf:mbox_sha1sum	A SHA1 hashed version of someone's email address (to hide the actual address from spammers)	**foaf:Person** resource descriptions	A literal value representing the SHA1 hash of the email address)
foaf:msnChatID	A person's MSN Messenger chat ID	**foaf:Person** resource descriptions	A literal value representing the person's MSN user name
foaf:name	The name of the resource	Any resource descriptions	Any literal value
foaf:nick	A person's online nick name (like their usual IRC handle)	**foaf:Person** resource descriptions	Any literal value
foaf:pastProject	A project that person worked on in the past	**foaf:Person** resource descriptions	Any resource
foaf:school Homepage	The homepage of the school a person went to	**foaf:Person** resource descriptions	Resources of type: **foaf:Document**

Property type	*Intended usage*	*Allowed within*	*Allowed values*
foaf:surname	A person's surname	**foaf:Person** resource descriptions	Any literal value
foaf:title	A person's salutation (Eg: Mr, Mrs, Dr. etc..)	**foaf:Person** resource descriptions	Any literal value
foaf:weblog	A weblog associated with a person, project or group	**foaf:Person**, **foaf:Project** or **foaf:Group** resource descriptions	Resources of type: **foaf:Document**
foaf:workplace Homepage	The homepage of an organisation that a person works for	**foaf:Person** resource descriptions	Resources of type: **foaf:Document**
foaf:workinfo Homepage	A page about a person's work at some organisation	**foaf:Person** resource descriptions	Resources of type: **foaf:Document**

Further details are available at: http://xmlns.com/foaf/0.1/

5.4 - Others

Ganesha currently only includes one additional property type form the RDF Schema language (RDFS).

Table 5-6: Other supported property types

Property type	*Intended usage*	*Allowed within*	*Allowed values*
rdfs:seeAlso	Indicates another resource that holds further information about the resource being described	Any resource description	Any resource

Further details are available at: http://www.w3.org/TR/rdf-schema/

Appendix III: Source Code (CD-ROM)

Files on the CD

`/src/`	Ganesha source code.
`/docs/`	Code documentation output by javadoc
`/lib/`	Compiled copy of Ganesha
`/tests/`	Test programs and sample RDF files
`/Jena-2.1.zip`	Copy of the Jena API
`/project_report.pdf`	Copy of this report in PDF format

Installing Jena and Ganesha from the CD

1. Unpack the supplied ZIP file to a directory of your choice:

   ```
   unzip /path-to-CD/Jena-2.1.zip -d /destination-dir/
   ```

2. Add each of the jar files located in `/`*destination-dir*`/lib/` to the CLASSPATH environment variable. In a bash shell this can be done as follows:

   ```
   for jar in `ls /destination-dir/lib/*.jar`
   do
           CLASSPATH="${CLASSPATH}:$jar"
   done
   export CLASSPATH
   ```

3. Copy the `/lib/` directory of the CD to a directory of your choice...

   ```
   cp -a /path-to-CD/lib/ /ganesha-dir/
   ```

4. ...and add it to your CLASSPATH environment variable.

   ```
   CLASSPATH="${CLASSPATH}:/ganesha-dir/"
   export CLASSPATH
   ```

5. Jena and Ganesha should now be installed. To start Ganesha type:

   ```
   java com.twiddles.ganesha.Ganesha
   ```

Viewing the documentation

Open the file `/docs/index.html` in a web browser to read the code documentation.

Updates and news

New versions of Ganesha and news about its future development will be available from the project website which is located at:

`http://event-horizon.kicks-ass.net/uni/ganesha/`