Simplifying the Development of Semantic Web Portals Through GUI Based Rule Generation

By

Josh Stewart

Submitted to the School of Information Technology and Mathematical Sciences in partial fulfilment of the requirements for the degree of

Bachelor of Computing (Honours)

at the

UNIVERSITY of BALLARAT

June 2005

Certified by: Mr. Greg Simmons
Lecturer, School of ITMS
Thesis Supervisor

Simplifying the Development of Semantic Web Portals Through GUI based Rule Generation

By

Josh Stewart

Submitted to the School of Information Technology and Mathematical Sciences on October 31, 2005, in partial fulfilment of the requirements for the degree of Bachelor of Computing (Honours)

Abstract

The enormous growth of the World Wide Web has resulted in vast amounts of data being made available to anyone with an Internet connection. The sheer quantity of this information and the way it is currently represented, however, has lead to difficulties in searching and locating quality resources. The semantic web is a proposal for a next generation Internet, and hopes to address these issues, however, its uptake has been slow due, in part, to a steep learning curve and high development cost. This paper presents a method for simplifying development within the field of semantic web portals through a user friendly, GUI based rule generator. An example based on the OntoViews semantic portal framework is presented.

Thesis Supervisor: Mr. Greg Simmons (Title: Lecturer, School of ITMS)

Acknowledgements

The author would like to thank the following people, without whose assistance this project would not have been achievable:

- My family and friends for continued support throughout this year.
- Mr Greg Simmons for his ideas and direction.
- Current and past honours students who have assisted both technically and otherwise.

Contents

Table of Figures

Chapter 1: Introduction

The enormous success of the Internet has come primarily from its ability to store, manipulate and retrieve massive amounts of information very quickly. Developments in technology have allowed media rich pages containing images, sound, animation and even video to be accessed rapidly and easily. Consequently, this has lead to the convergence of content and presentation, making the two difficult to distinguish. This makes little to no difference for an end user of the Internet who typically only sees the end result of this merger, a webpage. The problem with this, however, is that the data then becomes difficult and in some cases, impossible, to manipulate and read automatically.

One of the great strengths of large database systems is their ability to retrieve data quickly and in a logical manner, most commonly through the use of a query language. Whilst the database doesn't 'know' anything about the content it holds, it is still able to answer queries thanks to the metadata that makes up its design.

This concept of 'metadata' refers to the inclusion of information that describes the data itself. This metadata is of little use to a user directly and is not shown to them. Instead the metadata allows the computer to gain an understanding of how a piece of data may relate to another. The semantic web takes this idea and applies it to the existing web. By including machine interpretable metadata, such as categories and relationships, a semantic server not only has the ability to respond to basic queries but can also deduce other information about the content. In semantic terms, the format this metadata takes is that of an ontology (Berners-Lee et al,. 2001) that contains objects, properties, hierarchies and, most importantly, relationships.

The semantic web however has not had the fast initial uptake that other technologies in the online field have experienced.

While other web-based technologies (Example: server side scripting, vector graphics/animations etc) have enjoyed almost immediate uptake success, the semantic web has been conspicuous in its failure to gain acceptance. This can be attributed to a number of causes, but primarily the semantic web does not enjoy the obvious and immediate benefits that other innovations have relied upon, thus limiting uptake. This is especially critical in the

massive corporate segment. Benjamin et al 2002, identified 6 areas of difficulty that the semantic web faces:

- Multilinguality (Internationalisation)
- Visualisation
- The availability of content
- Ontology availability, development and evolution
- Scalability
- Stability of Semantic Web languages

Underpinning the latter four points is ease of use. Whilst the systems required to construct, for example, a semantic portal, are freely available (generally under open source licenses) they are very much 'first generation' systems, designed to simply provide a framework for semantic development. To setup a portal these tools require a high level of knowledge in a number of areas, typically ontologies, web administration and programming experience in the language of the particular application (ie perl, prolog etc). For the semantic web to find a wider acceptance, the move from developing "proof-of-concept" software to developing applications that are more practical in real world scenarios is crucial.

This project focuses on assisting a web developer in setting up a semantic portal. It will focus on reducing both the time required to move to a semantic portal and the number of new skills that would typically be required. Doing so will increase the ease of semantic development and make it a more attractive option than is currently the case when compared with other orthodox web applications.

Like developing for any web application, building a semantic portal can be approached from a number of angles. Developing a portal from scratch is often too expensive for a potential user and so to save both time and effort most people elect to choose one of the existing semantic portal frameworks and adapt their content to this expensive (Della Valle and Brioschi). In this research project we have selected the OntoViews semantic portal (Makelä et al., 2004) developed by the Semantic Computer Research Group (Seco) at the University of Helsinki. The portal was primarily designed to serve cultural content, specifically museum items, however is useful for any scenario where descriptions of many items and their relationship to one another are required (an online catalogue for example).

Whilst this research focuses on OntoViews, the primary goal is to demonstrate one technique for simplifying semantic portal development. The implementation presented

whilst being specific to the OntoViews project has broader implications. A common problem observed with current semantic web applications is the lack of reuse and technical complexity required to implement said application. Techniques like the one presented in this thesis will need to be developed to increase the uptake of semantic web technology. This solution is not aimed to be appropriate in every scenario, but instead to show the need for such solutions within this field.

Unlike some other models/systems, OntoViews is a complete semantic portal that includes all services required to run a semantic website (excluding the creation of the content itself) and is written in a combination of languages including Java, perl and Prolog, allowing it to be deployed on all major platforms (Windows, Linux / Unix, Mac etc). OntoViews semantic functionality is presented to the user via both semantic browsing and semantic searching capabilities.

1.1 Research Question

At the highest level this paper aims to investigate how uptake of the semantic web portals (and therefore the semantic web) can be increased. By providing tools to simplify the process of developing a semantic portal, in this case OntoViews, it is hoped the steep learning curve and high initial investment of time and effort that is required to implement a semantic portal application be mitigated.

Ontologies are essential to semantic web applications (and in particular to semantic portals) but are historically very difficult to develop. One key difficulty in developing ontologies is generating semantic rules, which require the input of a domain expert in addition to someone who is comfortable expressing the rules in logical language. It is the intent of this research to provide a tool to make the creation of semantic rules (and therefore ontologies and semantic portals) more user friendly. With this goal in mind this research project aims to present a solution to the question:

> *"Can a user-friendly semantic rule generator be developed to facilitate smoother adoption of semantic portal technology?"*

1.1.1 Research Direction

For this project it was decided that the primary focus would be to reduce the amount of time required to setup an OntoViews instance. This is to be done through simplifying the process of adapting data by reducing the amount of prior knowledge that was required by a developer. After analysing OntoViews it was found that there were a number of paths that could be taken to complete this goal.

Throughout the various applications that make up OntoViews there are a number of programming and scripting languages that have been used. These include:

- Java
- XML
- Perl
- CSS
- Prolog

These first four languages are areas where a traditional web developer or team of developers would typically be skilled. The final language however, prolog, is one that is not often related to existing web technologies and hence newcomers to OntoViews (especially those moving from a conventional web background) are not likely to posses skills in this area. Understandably this will cause setbacks in development as people require training and practice in this field before they are able comfortably able to take on new work.

For this reason I believe that the greatest benefit when simplifying OntoViews will come from making the use of prolog, as it relates to OntoViews, a more straightforward process, and therefore the research undertaken within this project will revolve around this goal.

1.2 Summary

This chapter has provided a brief introduction in to the problem of semantic web usability that is to be the focus of this paper. The following chapter presents a detailed outline of the background of the semantic web, and explores the problems with it that will be addressed within this paper.

Chapter 2: Background and Literature Review

2.1 Introduction

This chapter examines the history and core technologies behind the semantic web, as well as containing a more in depth description of the OntoViews semantic portal.

2.2 Background

The semantic web brings new functionality to the current web with the hope of improving efficiency. This section will give a brief outline of what the semantic web is and the technologies behind it.

2.2.1 The Semantic Web

The concept of the semantic web came from the father of the internet as we know it today, Tim Berners-Lee. Despite the current web's ability to store and spread large amounts of information very quickly and to a very large audience, this massive amount of data has made finding specific content in a short time difficult. In searching the web for specific information, the user can get easily lost in large amounts of irrelevant material, and may often miss the relevant matter (Fensel, et al,. 2002). The problem lies with the way data is stored and retrieved on the web.

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation (Berners-Lee et al,. 2001). Whilst the Hyper Text Markup Language (HTML) of the current generation web provides an excellent and simple mechanism for both storing and presenting content, it is a system that was designed for use by humans rather than for computers themselves. Content and presentation are mixed into the one document, making development and presentation a relatively straightforward process, however this also makes it very difficult to automatically deduce meaning from the data.

2.2.2 Core Concepts

The semantic web is based upon a number of concepts and technologies that are not found within the current web field. These are described below.

2.2.2.1 Knowledge Representation

The web, as we know it today, contains much more information than could ever be known to a single person. Theoretically it should be a better source of information. However, unlike the internet, a person is able to *reason* with the knowledge they have using logic, inference and related information to give new value to existing raw data. Such techniques are known as knowledge representation and it is something that is instinctive to humans but not to computers (Studer et al, 1998).

Knowledge representation has been studied since before the growth of the web, particularly within the Artificial Intelligence field. It is focussed primarily on exploring methods of storing knowledge representations in a logical manner, with the aim of increasing the value of the raw information it contains. To be of use to parties other than simply the developer, a representation must be both specific to the context of the application and general enough to have reference in similar areas. Therefore a developer wishing to express information domain has two options:

- Define their own knowledge representation from scratch; or
- Use and/or modify a common (or shared) knowledge representation if one is available

Of the two, the second option is most desirable, as it then allows for the new application to be reasoned with in the same manner as other systems that have used the same representation. This concept of a shared data structure is fundamental to the usefulness of the semantic web, as will be described further in this chapter.

2.2.2.2 Language

Whilst knowledge representations can take any form the developer wishes, the concept of a global, semantic based web, requires consistency in these formats. As with any new technology, however, a number of different techniques have been developed, none of which has yet become either an official or de facto standard. This paper focuses on the Resource Description Framework (RDF), a markup language that formally describes web resources.

RDF is an XML language developed by the World Wide Web Consortium (W3C) and whilst being powerful in its own right, serves also as a foundation language and formalisation of the semantic web (Gómez-Pérez et al, 2004).

Beyond RDF, other languages have been defined that add additional capabilities. Within the context of the semantic web the most common of these languages are:

- The Ontology Interchange Language (OIL)
- DARPA Agent Markup Language + OIL (DAML+OIL)
- Ontology Web Language (OWL)

Whilst each of these languages adds their own capabilities to RDF, they will not be covered further in this research.

RDF, being an XML language, does share the use of markup with current web standards such as HTML, however, it differs from it in both purpose and structure. The primary difference between the two is in the goal of the markup they utilise; the tags used within HTML are for presentation rather than metadata as is the case in RDF. Very simply, RDF is a form of text data store where the primary goal is to describe the content in such a way that it may be interpreted and reasoned with by a computer. It is useful in situations where data may need to be moved between applications but also, as is the case with the semantic web, where the ability for the computer to understand the data allows for significantly more efficient searching across multiple applications.

An RDF data set is made up of two parts:

1. The data itself (With required metadata)
2. The RDF Schema (RDFS) which defines objects that will be described. This schema provides the metadata that can later be used for functions such as locating, linking and inferring.

Combined, these two elements make what is known as *an ontology*. RDF is therefore an *ontology language*.

2.2.2.3 Ontologies

Ontologies, as they pertain to the information technology industry, have been constantly evolving since their first uses around 1991. Consequently the definition of *'ontology'* has also changed. This research uses the following definition from Studer et al 1998, which is itself a version refined from previous authors:

"An ontology is a formal, explicit, specification of a shared conceptualisation. Conceptualisation refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group."

An ontology is closely tied to knowledge representation, however whilst the latter will show *how* to represent a concept, an ontology will specify *what* concepts to represent, and how they are interrelated (Fensel et al, 2002).

In practice, an ontology will typically describe a potential item, or more likely, a group of items, which will then have an instance. Much like a database schema, the schema for an ontology defines the format of the data to be represented. Rather than simply defining tables however, an RDF schema provides a vocabulary of class resources, properties and relationships between them.

For example: An RDF schema about a dog may state that that it requires the properties colour, breed, age and name. The schema may also provide allowable values for these properties. There may then be many instances of a dog, each describing a different object.

Ontologies differ from relational databases through the way they describe data, specifically the relationships between data. Each statement within an ontology is made up of triples containing a subject, predicate and object.

For example: The schema for the dog in the previous example would describe its properties in the form "The **DOG hasBreed terrier**".

- The subject is the dog
- The predicate is the relationship hasBreed
- The object is terrier (This will be described elsewhere)

The ability of the semantic web to reason and infer information from multiple content sources is the result of ontologies providing a shared understanding of a domain (Antoniou, and Van Harmelen 2004). In short, the sharing of ontologies, or more likely partial ontologies, allows for related information to be logically linked by a reasoning service.

Additionally, even if two datasets representing similar content do not share the same ontology, the semantic service may be able to map elements onto one another.

> **For example:** Ontology A may describe a property 'hasZipCode' whereas ontology B defines the same data as 'hasPostCode'. Both ontologies are attempting to store the same information and this can be recognised by a logic engine, which would then treat the two as equivalent.

2.2.2.4 Reasoning Services

Crucial to the usefulness of the semantic web are reasoning services. Without them even the most well designed and descriptive ontology is no more functional than a relational database. Reasoning services, commonly known as inference engines, can be used to reason over instances of an ontology or over ontology schemas. In this context the ability to perform this reasoning is known as *logic* and can take a number of forms relevant to the semantic web:

- Inference: The ability to determine a *fact* pertaining to an object, which is not expressly stated within the ontology, is known as inference
- Justification: If a semantic web application uses reasoning to make a decision, the logic behind the decision can be used to justify why this was made
- Contradiction: By comparing rules and content, logic can be used to determine instances of contradiction that may exist within the ontology. These may then be handled in either a generic or specific manner
- Combining ontologies: Logic can be used to decide whether two representations of data, that may have been defined differently, actually describe the same object or property

2.2.2.5 Communication

For the purposes of clarification, it should be noted that the semantic web is built on top of existing web technologies, thus all required infrastructure is currently available. The medium for all semantic communication, just as with existing web communication, is the Hyper Text Transfer Protocol (HTTP), a method for sending and receiving text or binary data over the Internet Protocol (IP). Just as with traditional web sites, the semantic web is decentralised, meaning that anyone is able to start a web server/service without a central point of control.

2.2.3 Core Concepts Summary

Figure 2.1 shows the layering that makes up the semantic web. When considering only languages, the semantic web begins with XML, a flexible, generic, self-describing language. Extending XML is the base semantic language, RDF. This is the first possible data level for any application on the semantic web. If further flexibility is required to describe the ontology, one of the more extensive vocabularies, such as DAML, OIL or OWL can be utilised. Once the ontology is created, semantic reasoning can be performed in the logic, proof and trust layers. This research focuses mainly upon the creation of the logic layer through rules within the reasoning engine.

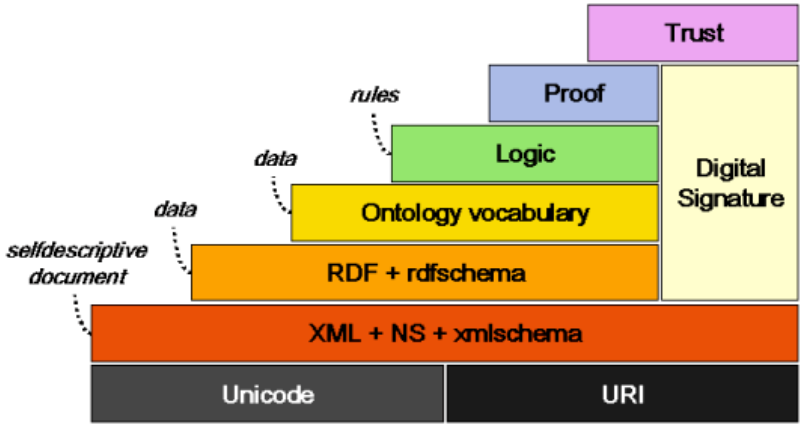

Figure 2.1: Layers of the semantic web

Image courtesy of w3.org (www.w3.org/2001/12/semweb-fin/swlevels.png)

2.3 Semantic Web Applications

Whilst the semantic web is essentially a method of describing and storing data, it requires additional software to format and display this data in a way that is both useful to the user and makes use of its semantic nature. There have been a number of differing approaches to displaying semantic content and both have strengths and weaknesses. Typically a method for displaying semantic content can be generalised into one of the following two categories:

1. A semantic browser that allows the user to view content directly
2. A semantic server (portal in this case) that uses one or more applications to convert semantic data to a standard web format (ie HTML, CSS etc) and then serve this in

the manner of traditional web servers (ie HTTP connection with data interpreted by an existing browser package).

The OntoViews portal examined in this paper falls into the second category. This section will explain in detail both of these methods with reference to an existing instance for each.

2.3.1 Semantic Browsers

The idea of a semantic browser is very similar in approach to standard web pages. Currently, the steps for a user to view a page on the Internet typically are as follows:

- A request for a certain URL is made by a client through a browser and sent to the web server
- The web server identifies the resource being sought (typically a file located on the server) and returns the contents of the file
- The users browser receives the file and renders the contents for the user.

Note that this is a very much simplified example that does not take into account technologies such as server side processing of the data or dynamic pages where content is stored within a database etc. The important thing to note, however, is that the file that is located on the server and the file that is rendered for the user are in the same format (HTML, CSS etc).

A semantic browser, such as IBM's Haystack (Quan and Karger, 2004), follows this same idea of a server simply being an interface to the content. This method has the advantage that the data itself will only need to be decoded once, in this case by the browser. It also allows for an existing web server, or one with very few modifications, to be used, as it is simply performing a standard role of passing data along a HTTP interface.

The idea of the semantic browser, however, has one significant drawback. The requirement for a very specific browser on the users end is something that makes this method largely unfeasible, as a shift such as this across the entire Internet is not practical. The semantic browser still finds uses in many areas however, specifically ontology creation and related semantic development, as it removes the need for a dedicated server when viewing data.

2.3.2 The SEAL approach

The SEmantic portAL (SEAL) approach (Stojanovic et al,. 2001), is a framework or template for designing a semantic portal system that can be accessed via traditional web means (i.e. a browser). The OntoViews portal follows many of the recommendations listed within SEAL.

The SEAL framework, unlike semantic browsers, relies upon a central web server that has been built specifically for working with ontology-based content. A system based around the SEAL model will act in a way that is much more transparent to someone browsing the content. They will not require a special browser and the semantic portal will appear in a similar manner to traditional web pages.

Functionally, the semantic processing is performed on the server side, and is completed through a multi layer system. The SEAL framework dictates that the web server and reasoning engine (or Ontobroker in SEAL terminology) are separate components that communicate once a request is received. This additional layer of abstraction makes for a system where tasks are clearly defined and assigned to a particular component.

The SEAL model suffers from a high processing requirement at the server end, however due to the modular nature of a SEAL system, tasks can be balanced across multiple servers to increase performance. This approach also requires two instances of data processing and conversion; once when the web server (or separate process) converts data from ontology format (ie RDF) to HTML (or other common language), and once when this language is processed and finally rendered by the client.

2.3.4 Other Uses of the Semantic Web

In addition to the 'end-user' driven technologies described previously, the semantic web also offers great potential in 'back end' systems that a user does not see. Such systems provide enormous power, particular in the business segment, by dramatically increasing the ability of information to be found automatically. These systems include:

- Semantic web services
- Semantic agents

Whilst being similar in their goal of locating specific information, web agents and web services differ in their execution.
A web service is a published system that clearly states its function in terms of inputs and outputs. Web services have become very popular within businesses who are offering a service online but do not wish to give other businesses direct access to systems such as databases.

> **For example:** A web service offered by an airline might state that it will locate and book the cheapest air travel available based on criteria such as departure location, destination, flight dates and desired class. Such a service could then be utilised by a travel agency without the need for them to ever interact directly with the airlines databases or booking system.

A web agent on the other hand is a decentralised task that is designed to search out information from any number of sources. The agent is given a goal that it then automatically completes, using either previously known or newly discovered web resources. The semantic web provides an excellent mechanism for agents as it is based around a common set of standards (ontologies), which can be interpreted and used as required by third parties.

2.4 Background – OntoViews

The OntoViews semantic portal framework will be used throughout this research. A high-level description of the system is given below.

2.4.1 Description

OntoViews is a web portal that uses semantic concepts to display content to users in a way roughly in line with the SEAL framework. Similar to a traditional web portal that is using a database to store content, page templates are created for OntoViews that describe the visual layout and presentation of the data in a way that is completely independent of the content itself. Unlike the existing style of portal however, the selection criteria for content within OntoViews (ie what data gets shown on which page) is decided by a ruleset that makes selection decisions based on the content itself.

Looking from a users perspective, the end product of OntoViews is a website containing information describing different objects. These objects can be found by using either a text based search that will identify an item by name or property, or by browsing categories of similar items. Once an object is found, its properties may be viewed and on the same page a list of related items are shown. The relationship between the current item and those being linked to, is generally based upon a property value that they may share, eg: They were manufactured during the same time period.

From a back end perspective, OntoViews is doing a number of things within each abovementioned step. Firstly it must be understood that OntoViews itself contains very few HTML files and none of these are what make up the main pages. For every request it receives OntoViews is transforming data from semantic format (RDF in this case) into HTML via an XSLT translation. In this way all content shown via an OntoViews portal is dynamic. The RDF content that OntoViews is transforming can come from a number of places, including results from a search or relationship lookup. To understand this better an explanation of the parts that make up OntoViews is required.

OntoViews is the merging of three separate applications into one complete semantic portal system. These three parts are:

- Semcocoon: The webserver process used by OntoViews. Translates ontology data to HTML using XSLT transformations and serves them through HTML/CSS based webpages. Semcocoon is build from the Apache Cocoon framework and functions like a typical webserver from an external perspective.
- Ontogator: OntoViews' search facility. Ontogator searches through the semantic content and returns results in RDF format to Semcocoon. These results are then translated and displayed to the user. In this way OntoViews is able to provide users with a method to search through ontology data with a standard web search interface.
- Ontodella: This is the reasoning engine that OntoViews uses to find relationships between objects. It runs via a HTTP interface that accepts GET or POST requests containing a URI and the request type (eg: category lookup, relationship lookup etc). Ontodella then returns, in RDF format, results that can be interpreted and translated by Semcocoon.

The way these three applications integrate is shown in Figure 1 below

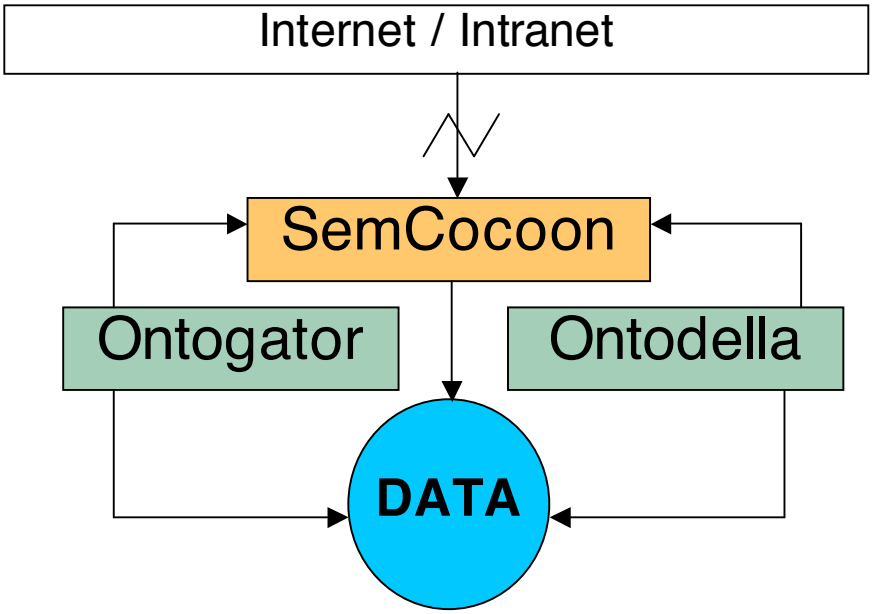

Figure 2.2 - OntoViews application structure

As can be seen in **Figure 2.2**, the data itself maybe accessed by any one of the three applications, depending on the request from the user.

2.4.2 OntoViews weaknesses

OntoViews, like any semantic system, is disadvantaged by a steep learning curve. Such a problem is a major contributor to the small uptake of the semantic web and tools such as OntoViews. In the case of OntoViews, however, there are three additional factors that work as a disadvantage:

1. There is very little documentation on setting up a new instance of OntoViews. The information that comes with the package when it is downloaded focuses on starting the various servers with the example content from the Museum of Finland site. While this is a good way to demonstrate how to get OntoViews running, it does not help a developer who is trying to move their own content onto this platform. Additionally, being a new technology, there are not yet any third parties providing assistance with OntoViews through way of documentation.
2. There is a large amount of work required to adapt existing semantic content (ie content that is already in RDF or similar form) to OntoViews. This is partially a traditional web problem of designing the layout of pages etc, however OntoViews also requires the creation of rules that dictate the way it functions as a semantic

portal. This is both a time consuming process and one that requires skills not necessarily associated with web development.

3. The language. The entire Museum of Finland site is written and displayed in Finnish. There are no other languages that the content, code and rules are available in. This is a major disadvantage for OntoViews as it can turn people off the product almost immediately, simply because they realise that they will have difficulty understanding both the example system and the back end elements.

Whilst these problems are specific to OntoViews and will no discourage some people initially, they are not directly related to the way the system operates. Therefore the solutions for these problems are not a matter of altering the OntoViews software in any way, but instead altering the way a new or potential developer uses and sees the software.

2.5 Summary

In this chapter a background to the semantic web and specifically semantic portals has been presented along with the technical work that this research will be based around. The next chapter outlines the methodology that will be used in the approach to solving the research questions.

Chapter 3: Methodology

3.1 Introduction

This chapter outlines the scientific process that was followed to provide answers to the research questions proposed in chapter one.

3.2 Approach

As the solution to the stated problem will be demonstrated through the use of a software artefact, the System Development Research Process (SDRP) as described by Nunamaker, Chen and Purdin (Nunamaker et all., 1990) was used. The SDRP describes five steps in the process of developing solutions to research problems where an example in software is desirable. These steps are listed below:

1. Definition of conceptual framework: Within this step the researcher outlines the problem that is to be investigated and gives a justification as to why this needs to be solved.
2. Definition of Requirements: In this step a listing of all functional and non-functional requirements of the solution are given. Each requirement should be shown to be providing a solution (or part thereof) to the problems.
3. System Analysis and Design: This step is the implementation of a software framework for the requirements outlined in Step Two.
4. Reference implementation: The building of a prototype system that is based on the design created in Step Three, and hence satisfies the requirements of Step Two. This prototyping process is crucial as it both demonstrates feasibility of the proposed system and makes apparent any advantages and disadvantages of the chosen solution.
5. Experimentation, Observation and Evaluation: This step is required to ensure that the implemented system meets the requirements listed in Step Two.

3.2.1 Conceptual Framework

Both the background and reasoning for this research have been given in the Introduction chapter of this paper.

3.2.2 Definition of Requirements

A solution for the problems of high work and wide ranging skill requirements when producing semantic web content, as detailed in the Conceptual Framework, will be given in the form of a Prolog rule generation system. Such an artefact demonstrates the type of solution that is required in many areas of the semantic web field to increase uptake.

The system presented will provide the following functionality as a means of solving a given problem:

- The ability to create a full 'set' of rules as required by Ontodella without the need to write Prolog code directly
- A GUI to all object definitions and previously defined rules
- A basic set of comparison methods that can be used within rules.

3.2.3 System Analysis and Design

The first step in providing the solution described, is to analyse and document the existing Ontodella rule system. After this the planning of how to fulfil the following software milestones is required:

- Loading and visually displaying ontologies
- An interface for creating category and relation rules between ontology objects
- How a rule specified within the application will be transformed into the Prolog format required by Ontodella.

The product of this stage will be documentation detailing the chosen design and justification of why such an approach has been taken.

3.2.4 Reference Implementation

At the completion of system design, a reference implementation based around this conceptual model can be produced. This will test the assertion that the requirements listed

sufficiently fulfil the aims of the research and simplify the process of rule creation for Ontodella, and hence reduce the time and effort required to develop a semantic portal.

3.2.5 Experimentation, Observation and Evaluation

The reference implementation will be used to integrate a well-known sample ontology into Ontodella. Firstly a list of desired rules will be produced and an attempt made to represent these rules within the system. If successful the rule file generated will be tested with Ontodella to ensure that it is both syntactically and logically correct.
These tests will demonstrate that the reference implementation has the functionality to meet all the given requirements.

3.3 Summary

This chapter has given the methodology that this research will follow in attempting to produce a system for automated Ontodella rule generation.

Chapter 4: Implementing OntoViews

4.1 Introduction

This chapter will look at adapting new data into the OntoViews semantic portal system. It will focus primarily around incorporating the data into the Ontodella package and configuring Ontodella to serve this new content. In doing this the potential difficulties in creating a new OntoViews portal will be identified and explained in context.

4.2 Package details

To better demonstrate the functionality of the OntoViews framework, a detailed description of it will be given.

4.2.1 Relevant File Listing / Explanation

As there is significant customisation required when adapting content to Ontodella, it is useful for an explanation of relevant files within the package to be given. As the file/directory structure of Ontodella in its default form is not 'neat', it may also be useful for new developers.
Whilst this is by no means a comprehensive description of every file that makes up Ontodella, it is useful for clarifying the various sections a developer will need to use or alter.

Directory	File	Description
bin/	run-onto.sh	A script used for starting the Ontodella http server
conf/fms/	ontodella_config.pl	The primary configuration file for Ontodella. Rule and data files are specified here.
conf/fms/	fmsweb_sewehgrius_rules.pl	By default this is the file where all category, leaf and relation rules are described
data/fms/		The default directory for ontology files (Both RDF and RDFS)
lib/prolog/	sewehgrius_rdfs_core_rules.pl	A prolog file containing utility predicates used by the rules. May require customisation.
src/prolog/	ontodella_server.pl	The prolog http server itself

src/prolog/	ontodella_request_handler.pl	Called by ontodella_server.pl each time a request is received. Currently contains hardcoded data requiring customisation.

4.4.2 OntoViews Navigation Structure

Before an explanation of how OntoViews works can be presented, it is important to understand and appreciate the purpose and theory behind web portals.

A web portal is a method for presenting large amounts of information within a particular domain to a user. The content itself is almost always removed from the presentation layer and stored in some form of database, usually relational, however in the case of semantic portals this is normally an ontology.

A web portal, semantic or traditional, will generally provide the user with a number of different methods for locating the content they require. All these methods strive to ensure the user can find any content within the portal in the shortest amount of time and with the fewest 'clicks'. Within OntoViews, 3 different paths are available to users browsing the portal:

Searching: A term may be entered into the search box and relevant entries from the ontology will be found and presented. OntoViews provides this functionality through the Ontogator package. This will not be discussed in this paper.

'Tree' / Categorical browsing: Like most web portals, OntoViews' primary navigation method is through a tree of objects. Objects are grouped with categories being displayed on the main page. Each time a user selects a category they are taken to a page showing any sub-categories and objects contained within. In this way users can methodically locate what they are searching for, provided the tree is correctly defined.

Relational browsing: Relational browsing allows users to see related items once they are viewing an object. The relationship between two items may be simple or complex and can be based upon any of the properties contained by an item. The relationship becomes most useful when it is based on a property that was not part of the categories making up the portals navigation tree.

> **For example:** A portal for an online video store may present a navigation tree that groups movies by genre. Such a tree is logical and would be useful to users attempting to locate a

certain movie. Relational browsing would allow the user, once they have found a movie title, to quickly see other related items such as movies with the same director, movies made in the same year or movies containing the same actors.

This method, whilst not unique to semantic portals, is made significantly simpler when an ontology is used for storing content. This style of navigation is secondary in nature as it relies upon a user having already found an item to act as a starting point.

From a business perspective the ability to link an item to another without interrupting the logic of the categorised browsing 'tree' can be very appealing.

For example: Relational browsing allows for a retailer to display to a prospective customer other items that they may not have initially considered purchasing. The inclusion of these links does not affect the users ability to find items in the first place as the logic of both the searching and categorical browsing remains unaffected.

4.2.3 Overview and Implementation Requirements

To adapt any new data to Ontodella there are a number of steps that must be taken. From a process perspective, Ontodella's main function is to take content in RDF form and, when queried, return subsets of this content. Within OntoViews these subsets are typically categories and collections of items that share some similarities and can be grouped together when displayed in the portal.

Ontodella uses a rules based method for defining these categories and it is the creation of these rules that makes up the majority of work when adapting the system to new data. The rules themselves are written in Prolog and are divided into three different types, each performing a different function in OntoViews.

Ontodella Rule Types:

- Category Rules
    - Subcategory Rules
    - Leaf Rules
- Relation Rules

The function of each rule type can best be explained by showing where and how they fit into the end portal product.

Category Rules

Category rules makeup the most basic form of portal navigation. They provide an initial list of object categories that can be selected to show sub-categories and items contained within.

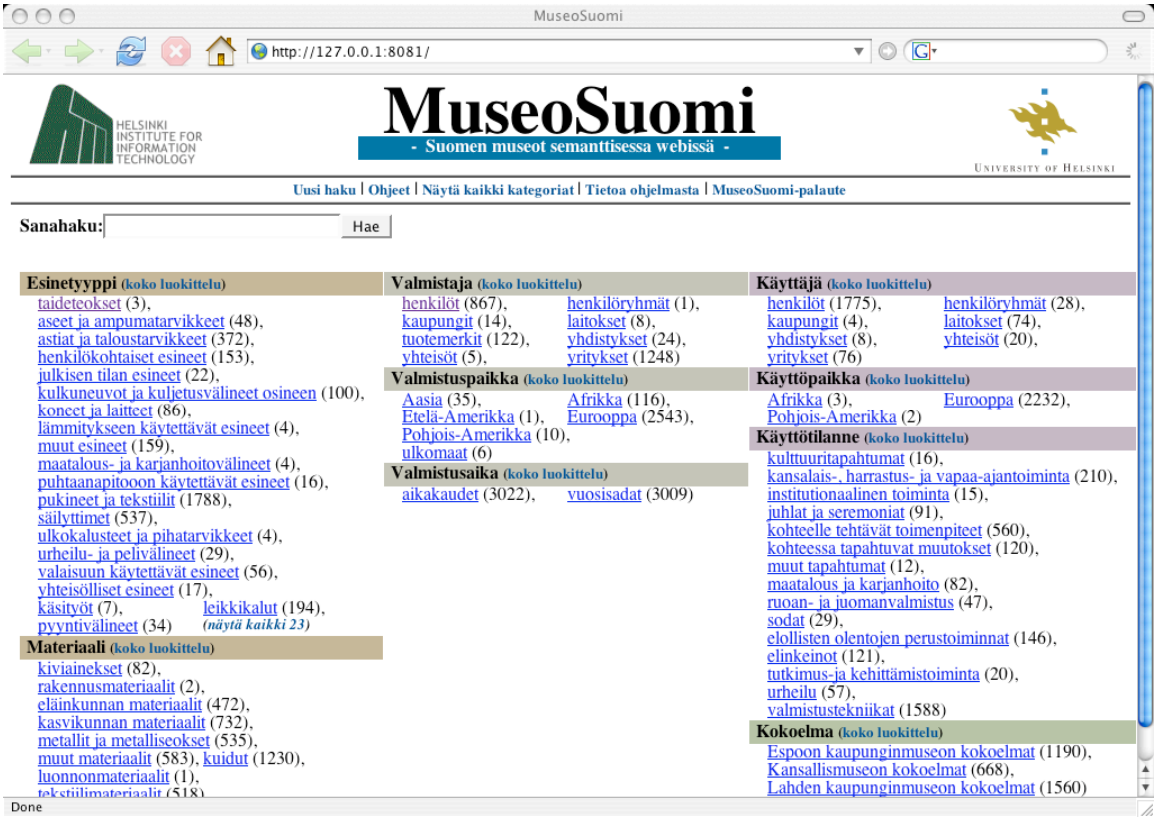

Figure 4.1 - OntoViews Main Page

Figure 4.1 shows the front page of the 'default' OntoViews portal, displaying the content from the Museum of Finland website. This page is broken up into a number of areas:

- The heading
- The search bar
- The content itself

The heading region and search bar are controlled by the Semcocoon / Ontogator applications and their functionality and creation fall outside the scope of this document. The content section, however, is made up of results returned directly from Ontodella through the use of the prolog rules for this ontology. On this front page, the content is broken up into 9 different categories (Esinetyyppi, Materiaali, Valmistaja, Valmistuspaikka, Valmistusaika, Käyttäjä, Käyttöpaikka, Käyttötilanne and Kokoelma), each containing subcategories. These

areas directly relate to rules that have been defined within Ontodella, the technical details of which will be discussed later in the chapter.

When navigating the portal, a user would typically click one of the subcategories shown to view its contents. Upon doing this OntoViews displays a page similar to the following:

Figure 4.2 – Subcategories within a portal

The right hand side of the page lists the relevant **sub-categories** and shows the first 5 **'leaf'** items in this subcategory. It should be noted here that Semcocoon, OntoViews' webserver, allows for this data to be displayed in any way the developer desires, therefore any new portals using OntoViews are not required to maintain this navigation path.

A leaf, as shown in the **Figure 4.2**, is the projection of an object described in the ontology onto the web portal's navigation 'tree'. Leaf objects are displayed within sub-categories and similarly are defined by prolog rules.

In this way Ontodella defines a navigation tree of **categories, sub-categories** and **leaf objects** that are then translated and presented by Semcocoon. This tree, however, should not be confused with the object tree defined in the ontology itself. The ontology describes a logical hierarchy that defines object types and their relationships within the ontology. Mechanisms such as inheritance of properties are built in to an ontology where strict definition is required. Conversely, a navigation tree/path is free from such restrictions allowing a developer to design what they believe to be the best system for a user to find information. The ruleset defined for Ontodella describes a navigation path that may bear no resemblance in structure to the ontology itself and is created with user accessibility as the primary requirement. Whilst it would be possible to mimic the ontology hierarchy within Ontodella, this would typically prove to be a very ineffective way for users to browse content.

Relation Rules

Relation rules differ in functionality from sub-category and leaf rules as they do not make up part of the 'primary' navigation structure. Instead they allow for 'relational browsing', where links to similar objects are shown when a user is already viewing an item.

Within the prolog code, relation rules are independent of content in the sense that when a rule is written, it does not define the object type it applies to. At run time any items meeting the criteria for the rule, regardless of any other properties, are shown.

Figure 4.3 – An object description on a portal

Figure 4.3 shows a page from the Museum of Finland site of an object in the collection. In addition to the details about the item itself (Eg name, time period, description etc), a list of 'related' items is shown on the right hand side. Any number of relation rules maybe defined and these are generic across all objects in the ontology, meaning that all relation rules will apply to all objects. If no matches are made for a particular object/rule combination then no indication of this rule when be given when the object is displayed.

4.3 Existing Documentation

In using any system for the first time, the quality of documentation will be crucial both to reducing the time required and avoiding any hurdles with implementation. Unfortunately being a new system with few implementations, OntoViews is disadvantaged by a lack of comprehensive and helpful documentation.

The standard Ontodella distribution contains only a single help file that explains how a user may setup a portal with the Museum of Finland content. No information is given in this document about the modification of this content or the inclusion of new ontologies.

Whilst there is further documentation that can be found on the internet, there is no comprehensive or official reference for anyone wishing to use OntoViews / Ontodella on their own portal. Additionally the official "walk-through" guide that is provided makes no mention of the changes that will be required to Ontodella core files as part of a move to new content.

4.4 Example Implementation

To better demonstrate the process of adapting new content into Ontodella, an example implementation will be outlined. This scenario will take an existing ontology and walk through the process of defining an Ontodella ruleset that could be used to present the content through a portal. The rules presented are not meant to be comprehensive (ie cover all areas that would be required by a full portal) and only show how each rule type is used. This example will not show the steps required to build the portal itself within Semcocoon, only the selection rules used by Ontodella.

4.4.1 Pizza

This example implementation will use the pizza ontology developed by the University of Manchester (Horridge et al. 2004). As the name suggests, this ontology describes different pizza types including, among other properties, their base, toppings, country of origin and category.

A portal designed to present information regarding pizzas could choose to display the following categories from its front page:

- All pizzas by category (ie meat, hot & spicy, seafood)
- Countries where pizzas originate
- All possible toppings

In addition to these category rules the portal would most likely wish to implement a series of relation rules to create the following links:

- Other pizzas that come from the same country
- Other pizzas with the same or similar ingredients
- Other pizzas that are of a similar 'spiciness'

The combination of these rules, especially when employed with OntoViews' search capability, would allow a user of the portal to quickly find a desired pizza, as well as ones sharing some properties that might also be of interest.

4.4.2 Ontodella Rules

Whilst Ontodella uses standard prolog for all rules, it also requires a strict syntax and naming format so that OntoViews' other components can communicate with it. This syntax is not complex, however it adds both another level of difficulty for a new developer, and also can make simple rules quite cumbersome to construct.

The most basic category rule for the pizza ontology (or any ontology) is one that simply follows the hierarchy of objects defined in the ontology schema. That is, a browsing tree is created that displays the children of a 'Pizza' object that, in turn, when selected will show their children. Such a function is perfect for prolog as it deals with the recursive nature of such a problem automatically. Ontodella makes this even easier as it provides an *'rdfs_subClassOf()'* predicate within its utility library.

An Ontodella rule that that fits these requirements is shown below:

```
%sewehgrius_category( -CategoryRootURIAtom, -CategoryLabelPredicateAtom, -SubStructuringPredicateAtom, -BookmarkPredicateAtom )
sewehgrius_category(
        'http://www.ballarat.edu.au/semweb#allPizzas',
        allPizzas_root_labels,
        allPizzas_sub_categories,
        allPizzas_leaf
).

allPizzas_root_labels(_, LabelList) :-
LabelList = [en:'Pizzas by Category'].

allPizzas_sub_categories( URI, SubCategoryURI ) :-
        Pizza_AllSubclasses_subCategory(URI, SubCategoryURI).

allPizzas_leaf( URI, leafURI ) :-
        Pizza_Istypeof_leaf(URI, leafURI).

Pizza_AllSubclasses_subCategory( URI, SubCategoryURI ) :-
        rdfs_subClassOf( 'URI',SubCategoryURI); % or!
        URI=='http://www.ballarat.edu.au/semweb#allPizzas',
        rdfs_subClassOf('http://www.ballarat.edu.au/semweb#Pizza', SubCategoryURI).

Pizza_Istypeof_leaf( URI, leafURI ) :-
        rdf_type( leafURI,'http://www.ballarat.edu.au/semweb#Pizza').
```

Figure 4.4 – Pizza category rule

Despite this being possibly the simplest category that could be defined**, Figure 4.4** shows that it is still non-trivial to create an Ontodella prolog rule that matches it.

The primary predicate for any category is *'sewehgrius_category()'* which takes 4 arguments:

- A name (URI)
- A predicate that provides category labels
- A predicate that identifies subcategories
- A predicate that identifies leaf objects

The selection predicates used may take any name or format the developer wishes provided that they have the correct arguments and that '*sewehgrius_category()'* retains the required syntax. The selection predicates shown above (eg: *'Pizza_AllSubclasses_subCategory'*) mostly use utilities provided in Ontodella's libraries however they can be written from scratch and as complex as the developer wishes.

> **For example:** The Museum of Finland portal uses rules that reference additional ontologies to determine whether an object within the museum's collection can be matched to an historical event.

Similar to category rules, relation rules have a mandatory predicate that must be defined, *'sewehgrius_relation_rule'*. Again this takes name (URI) and label arguments but only needs 1 selection predicate that matches related items. **Figure 4.5** shows an example

relation rule that matches items that have the same value for property *'hasCountryOfOrigin'*.

```
sewehgrius_relation_rule(
        'http://www.ballarat.edu.au/semweb#sameCountry',
        [en:'Similar Pizzas'],
        sameCountry_relation
).

sameCountry_relation( SubjectURI, RelationDescription, TargetURI ) :-
        hasCountryOfOrigin_Withsamevalue_relation( SubjectURI, RelationDescription, TargetURI ).

hasCountryOfOrigin_Withsamevalue_relation( SubjectURI, RelationDescription, TargetURI ) :-
        PropertyURI = 'http://www.ballarat.edu.au/semweb#hasCountryOfOrigin',
        rdf(SubjectURI, PropertyURI, SameTopic),
        rdf(TargetURI, PropertyURI, SameTopic),
        SubjectURI \= TargetURI,
        RelationDescription=[commonResources( SameTopic ), label(en:'Pizzas that come from the same country')].
```

Figure 4.5 – Same country relation rule

4.4.3 Customisations

In addition to simply writing the prolog rules required by a portal, a developer must also make a number of other, undocumented changes to Ontodella's configuration before it can be used.

The first is the addition of a number of mandatory predicates within the rule file that Ontodella looks for (and will fail without) when processing categories. In some cases these can simply be copied from the Museum of Finland configuration files, however a number of them require modification to suit the new content.

Secondly there are instances where Ontodella's core files (ie 'src' files that perform the actual processing of requests) contain hard coded references to the Musuem of Finland content that it was originally designed around. Whilst many of these references do not require alteration, there are some cases where it is either necessary or highly advisable to bring these inline with any new content, particularly new namespaces, being used.

4.5 Areas for improvement

After conducting this analysis of the Ontodella reasoning system, the following areas were identified as those where significant improvement is required from a developer perspective.

4.5.1 Rule Generation

Ontodella's prolog based rule and reasoning system is a powerful way of quickly performing queries on the ontology. Prolog allows for both simple and arbitrarily complex rules to be developed as required by the portal, however it comes at the cost of ease of use. For someone with a 'typical' web development background (ie no experience with prolog), the sharp learning curve required to develop these rules can be a daunting challenge.

Prolog, despite its many technical strengths, has two major disadvantages in an application such as Ontodella:

- Obscurity: Prolog has its traditional roots in the Artificial Intelligence (AI) field of computing, and is not widely used outside of this sphere, particularly not in the area of web development.
- Syntax ideology: Unlike the majority of programming languages, prolog is not procedural. This in itself is not a problem, however in the case of Ontodella the target developer is not one that is likely to have experience with such a language.

Rule generation within Ontodella is currently a major part of the development of an OntoViews portal, both in time and resources. Whilst this will always be true in the sense that the rules are closely related to the business logic of the portal, and must be well thought through, this stage does not need to be as technically complex as is currently the case. Simplification of the rule generation process, particularly in easing the learning curve required by a developer, would have a dramatic positive impact on the required development time and the appeal of the OntoViews portal system.

4.5.2 Documentation

The configuration of Ontodella (including rule creation) is, in itself, a relatively trivial task. However this is made both time consuming and frustrating for a developer by the lack of useful documentation for the system. Being a new system OntoViews cannot rely upon the

contribution of its user base for documentation, like in many other open source projects, and this is something that needs to be created and released in an official manner.

4.5.3 Ontodella core files (Lib and application)

Whilst undertaking this analysis of Ontodella it was discovered that, whilst a working portal implementation (ie The Museum of Finland portal) had been created, adapting new content to this system is not as straightforward a task as the documentation describes. It was found that many of Ontodella's core files, for example its prolog library (lib) and source (src) files are in many places customised for the museum content. In some cases this was simply a matter of having a namespace hard coded, however there are other places where the logic itself is specific to this particular data set.

4.5.4 Language

The Museum of Finland content that makes up the 'default' OntoViews / Ontodella installation is a very useful guide to a potential developer. Specific to Ontodella, it contains both simple and complex rules of both types (Category and Relation). The configuration files contain examples of all the systems capabilities that could be adapted and used by a developer.

One major problem with the museum example content however is that, in most places (both in content and rules), the language used is Finnish. This is possible because Ontodella defines very little formal syntax for configuration and rule files. Such freedom allows for a developer to write significant portions of the application (Eg Object selection predicates etc) in whatever language they choose. For a developer who does not speak / read Finnish however, this makes the process of understanding the example content extremely difficult. The problem is made somewhat easier by the internationalisation features within Ontodella that allow for objects to be labelled and described in any number of languages, however whilst the museum content does describe some objects in both Finnish and English, all predicates (ie functions) are described in Finnish only. Additionally this does not help developers who speak neither Finnish nor English.

4.4 Conclusion

This chapter has detailed the makeup of OntoViews with a focus upon the Ontodella reasoning engine it utilises. Descriptions of the different rule types required, from both end user and developer perspectives, have been given along with an overview of OntoViews' navigation system. Furthermore this chapter has outlined some of the potential difficulties that are associated with adapting new content to OntoViews.

Chapter 5 Referenced Implementation (Solution)

5.1 Introduction

This chapter will present and demonstrate the solution that has been created to the research question of simplifying development for the Semantic Web. Both the technical and logical facets of the solution will be explained in relation to the goal of simplifying the use of OntoViews.

5.2 Summary of solution requirements

The process of writing the three rule types contains a number of steps that can be daunting for a developer new to the semantic web or the prolog language. The solution being presented in this paper attempts to demonstrate a method that allows developers to define the required rules without the need to directly write any prolog code.

Within OntoViews prolog is used in Ontodella, the reasoning engine, to make decisions about how the objects in the data may be related. Ontodella itself is written in a combination of prolog / perl and communicates through the HTTP functionality provided in prolog's core libraries. Whilst there are some potential pitfalls in setting up a default instance of Ontodella, it is the development of content rules, not the running of a server itself, that is a source of difficulty for a developer.

To make decisions about data relationships, Ontodella compares content contained within the semantic data store (ontology) against a series of rules, which again are written in prolog. These rules are specific to every different web portal as their contents are directly related to the information being served.

> **For example:** In the case of an online movie store, a rule maybe created that links two movies if they have the same director. Clearly a rule such as this would be pointless in a portal presenting the content of museum collection.

It is these rules that enable OntoViews to make full use of the power provided by the semantic web, and will have a direct affect on the sites value to a person browsing the portal. The end portal product is made up of a series of links that are generated from the results of

one of more Ontodella queries, which inturn are controlled by its rules. It is for this reason that the number, complexity and purpose of these rules must be selected very carefully.

With the end goal being to make using Ontodella, and hence OntoViews, easier to use, this research aimed to investigate the Ontodella development process and create a piece of software that would allow the rules required for a typical OntoViews / Ontodella application to be created in a visual environment. Any relationships needed are specified through a GUI that provides users with listings of objects and their properties rather than through a 'code only' environment, as is the current case with Ontodella. The required prolog code is then generated automatically.

With such a package a user would not need to understand or be able write prolog code directly. Hence the requirements for a person using this application would be limited to:

- A good knowledge of the data being served
- An understanding of the required rules for this data
- Basic experience with GUIs and Boolean logic

5.3 Design Specifications

A number of desirable non-functional requirements were identified for the referenced implementation:

- Cross Platform: As the OntoViews system itself is capable of running on most major platforms, it would be unproductive and frustrating to many developers for any tool that had the goal of simplifying an implementation, to be limited to a single operating environment. For this reason a cross-platform solution was required and in this case the Java programming language was chosen as it allows for portable, GUI based applications to be created without the need for multiple versions. For this solution Java version 1.5.0 is required as a minimum.
- Ontology Format: As the semantic web is still a developing technology, many improvements are still being made to the fundamental components, the most relevant being ontology formats. Currently there are numerous ontology formats including DAML/OIL, OWL and RDF. While an in-depth discussion of the merits of each format is beyond the scope of this paper, the referenced implementation needed to support, at a minimum, one of these specifications. As the current version of Ontodella is based upon RDF this was the logical selection, however as this may

change in the future any design needed to be written in such a way that other formats might be incorporated with as little work as possible.

5.4 Implementation Description

This section will describe the major features of the referenced implementation. The order in which these features are presented roughly demonstrates the process a developer would need to follow when using the system.

5.4.1 Imports

Before any rules can be constructed, the dataset upon which they are based must be loaded. In the current form, the referenced implementation supports the importing of any valid RDF Schema (RDFS) file. Multiple files can be imported if the ontology is split or multiple ontologies are being used within a single portal.

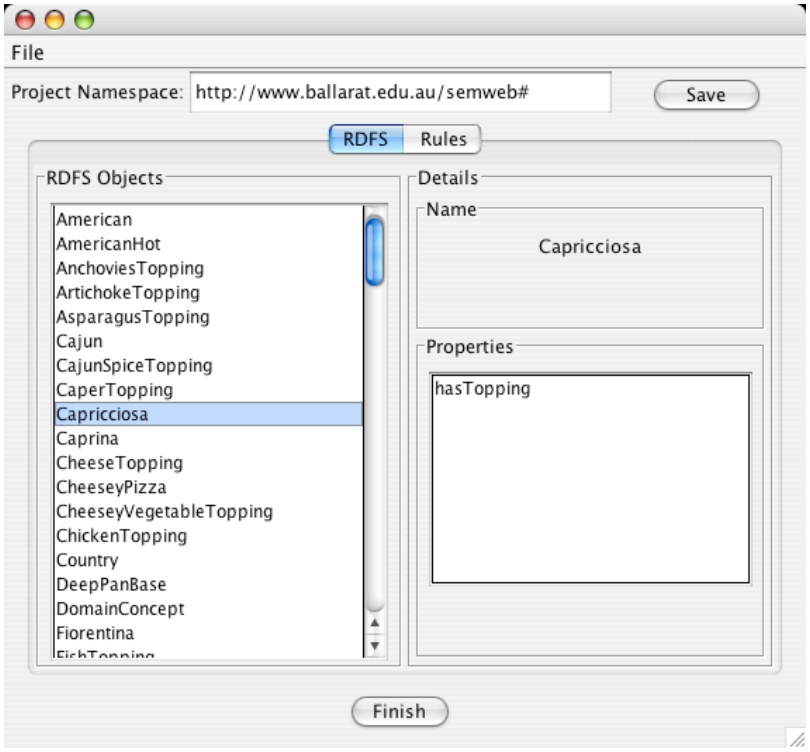

Figure 5.1 – Imported RDFS Data

As shown in **Figure 5.1** once an ontology has been imported, all objects and their properties are displayed. Here a developer can ensure they have all the required content.

5.4.2 Rules Overview

The rules that are defined within Ontodella (Both category and relation) are simply selection criteria for content that is to be shown at different points on the portal. A portal may define as many or as few of each rule type as required, depending on the application. **Figure 5.2** shows the rule page within the solution displaying lists of category and relation rules.

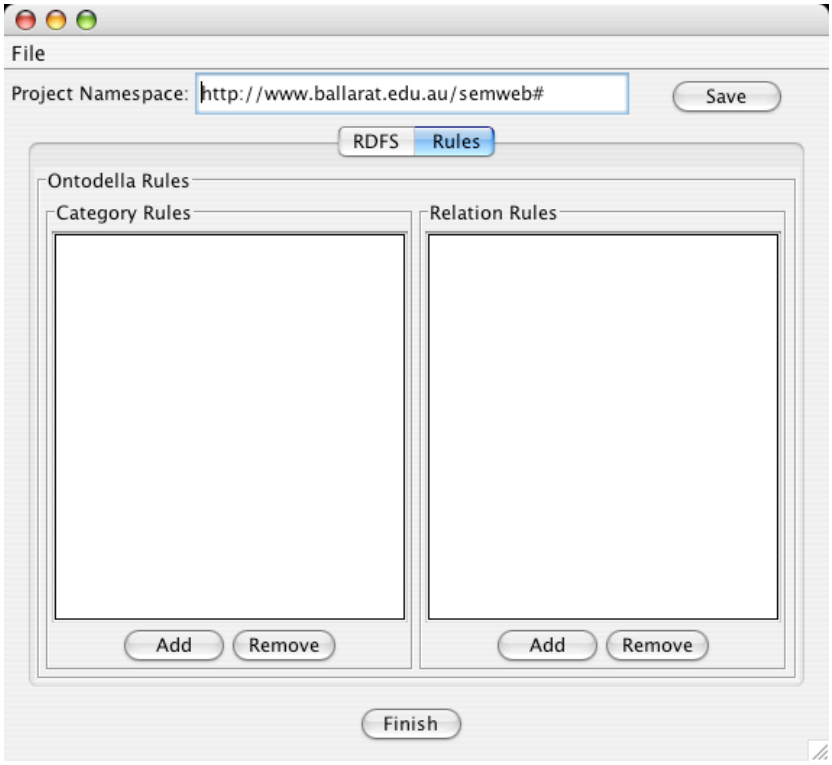

Figure 5.2 – Rule Listings

Ontodella defines its own prolog syntax for each rule type and in addition to the selection criteria they also specify a number of pieces of metadata depending on their purpose. There are three such types of metadata used by Ontodella:

- URI: Similar to properties, relations etc within an ontology, a rule is given a URI to each rule defined within Ontodella. Within the referenced implementation, a single, common namespace is used throughout a ruleset, as typically there will be no need for different rules to utilise varying namespaces.
- Labels: When a rule, be it category or relation, is called upon on the portal, it is headed by a label that briefly describes its content. Ontodella allows for this label to be specified in any number of languages providing support for internationalisation.

The language shown on the portal is specified through a parameter in the HTTP request.

- Titles: Titles apply only to relation rules. Titles function in a similar manner to a sub-heading for the relation. Whilst the rules label is what will be displayed as the major heading for the relation, the title serves as a way of categorising the results of the relation lookup for ease of reading on the portal. Like Labels, these may be specified in any number of languages.

5.4.3 Category Rules

Category rules are broken into two parts, subcategory definitions and leaf definitions. They are grouped together logically, as together they define the content that is displayed when a user selects a category within the portal itself. These two elements also share two pieces of metadata, a single name and a common label set. The name given to a category, along with the namespace of the project itself, forms the URI that defines the current rule.

5.4.3.1 Subcategories

A subcategory is the primary device for creating a navigation structure. The 'paths' within the structure will typically follow a family of objects within the ontology, gradually becoming more specific at each level. Within the ontology this is simply a matter of following the class -> subclass structure. For this reason there is very little customisation required for a subcategory rule. The ability to describe the rule through a GUI however significantly reduces the time and effort required when compared to writing the prolog code directly.

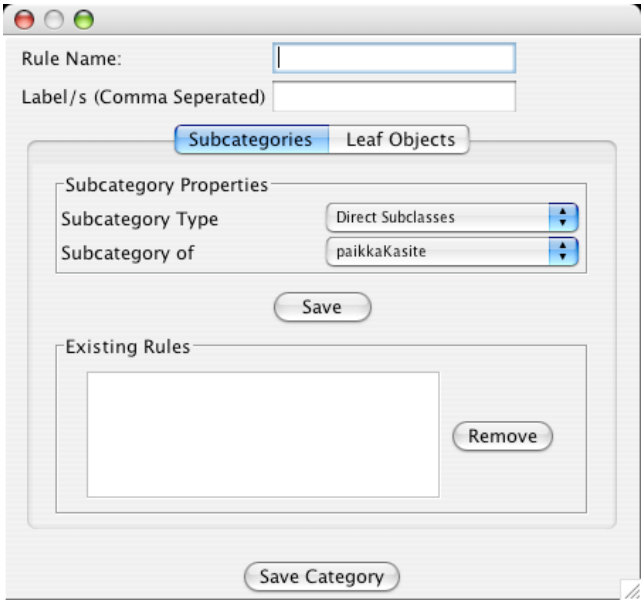

Figure 5.3 – Subcategory rules

Figure 5.3 shows the GUI screen for creating a Subcategory Rule. In its simplest form, a subcategory rule requires only two pieces of information, the rule type and an argument. Provision for two types of subcategories are given in the referenced implementation:

- Direct subclasses: Any object within the ontology that is a direct child of the class specified will be matched to this subcategory
- All Subclasses: Any object within the ontology that is a child, grandchild, great-grandchild etc, of the class specified will be matched to this subcategory

When multiple subcategory rules are defined for a single category, they are combined using a logical 'OR' relationship. Whilst there may be scenarios where it would be preferable to join subcategories with something other than an 'OR' relationship, (ie a logical 'AND'), these would be in the minority of use cases and have not been accommodated in the referenced implementation.

5.4.3.2 Leafs

Making up the other half of category rules are leafs. These rules define objects from the ontology that will appear as links ('Bookmarks') on subcategory pages within the portal. Similar to subcategory rules, leafs share the name and labels of the category rule they are defined within.

The solution provided defines two types of classification for a leaf rule:

- Hierarchy: As with subcategory rules, an object for a leaf can be chosen by its position within the ontology tree, in this case by its parent object. A user is able to choose a class from the ontology, and all objects that are a direct instance of this will be bookmarked on the category. **Figure 5.4** demonstrates how a hierarchical leaf where all instances of class 'paikkaKasite' are to be matched, could be defined.

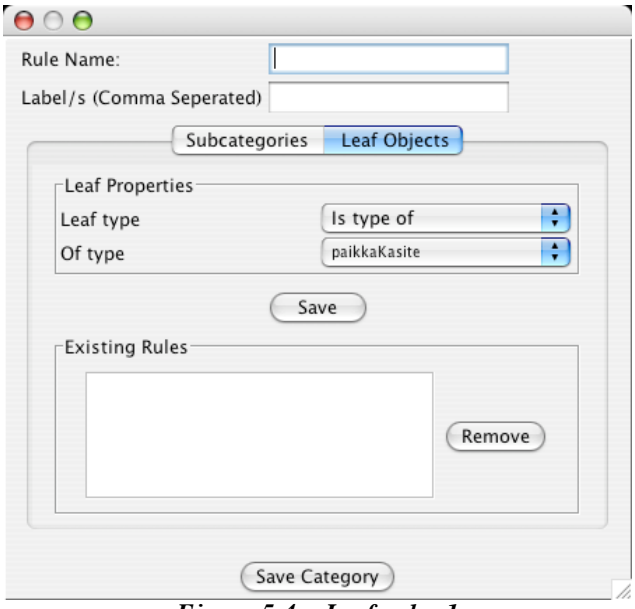

Figure 5.4 – Leaf rules 1

- Property Value: Logically the alternate way of selecting an item to appear as a bookmark, is by the value of one of its properties. As shown in **Figure 5.5** the referenced implementation also allows for a developer to select a property and enter a value for this. Any item within the ontology with this property value will be bookmarked on this category.

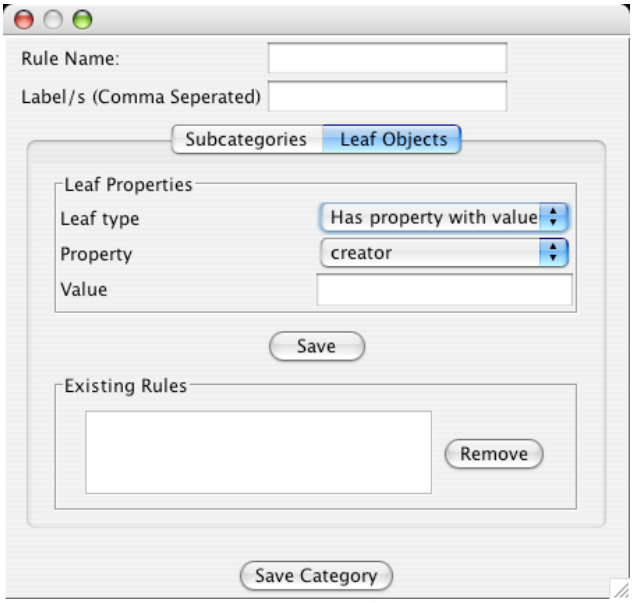

Figure 5.5 – Leaf rules 2

Just as with subcategory rules, when multiple leafs are defined within a single category they are related with a logical 'OR'. This allows for a developer to be very flexible with the items that are shown at each level of the portal navigation.

5.4.4 Relations

Relation rules allow for links to be created to other objects when a portal user is currently viewing an item. They provide a quicker method than navigation tree browsing for a user to move between objects they might be interested in, and also allow a developer to introduce links to items that otherwise may not have been found by a user.

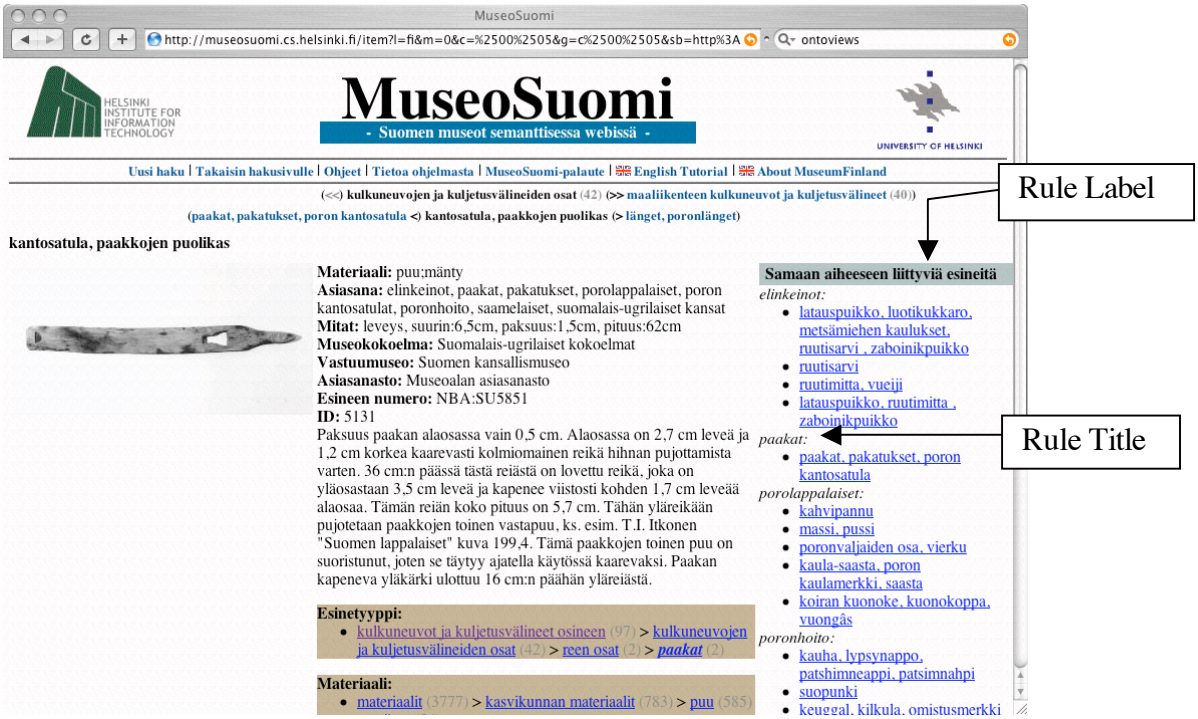

Figure 5.6 – Relation labels and titles

Similar to category rules, a relation rule has both a name and a series of labels. Again the name is used to form the URI whilst the label/s are what is shown within the portal. In addition to these properties, a relation rule also contains one or more titles. A title is used to provide a sub-heading when any related items are shown, and these take the same format as a label, therefore allowing multiple languages to be specified. This is shown in **Figure 5.6**.

In the initial version of the referenced implementation, only 1 form of relation rule is provided, however this will cover the majority of cases. This allows for items to be selected based on whether they share the same value of a specified property (ie 'With same value').

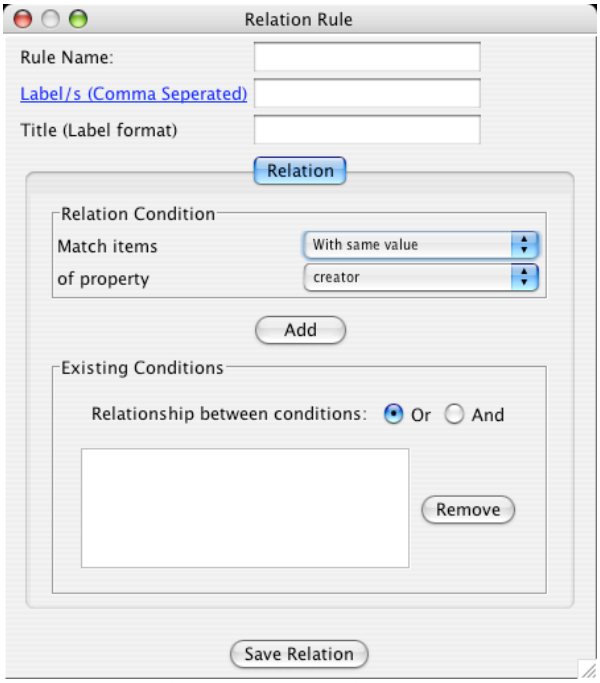

Figure 5.7 - Relation rules

As can be seen in **Figure 5.7**, multiple conditions for a single rule are again allowed. Unlike category rules, however, relation rules allow a developer to select whether these conditions are related by a logical 'OR' or 'AND'. This is because, unlike a category rule, both 'AND' and 'OR' will be used with a similar frequency, and to restrict the developer to one would cause significant problems in a many cases.

5.5 Walkthrough of use

To demonstrate the use of the referenced implementation, the example rules shown in **Chapter 4.4** will be recreated, step-by-step, using the solution that has been presented. This walkthrough will show how the solution is used to complete the rule generation process, but will stop short of covering how the rule file is incorporated into Ontodella as this is both trivial and covered in Ontodella's own documentation.

Step 1: Before any rules can be defined, the ontology schema must first be loaded as shown in **Figure 5.8**.

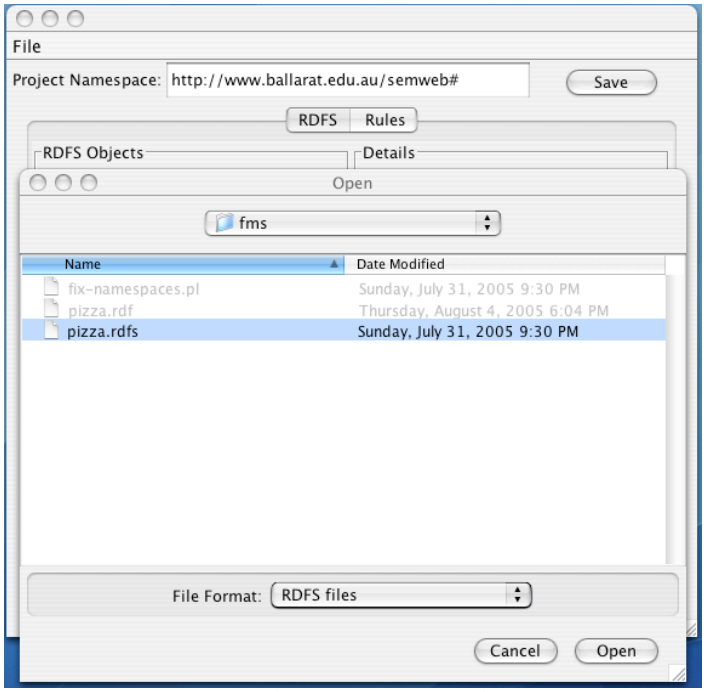

Figure 5.8 – Loading RDFS data

Step 2: Once the ontology is loaded, a basic subcategory rule can be defined that will select all subcategories of the 'Pizza' object. **See Figure 5.9.** As in chapter four, a basic label is given to the rule, in this case in English only.

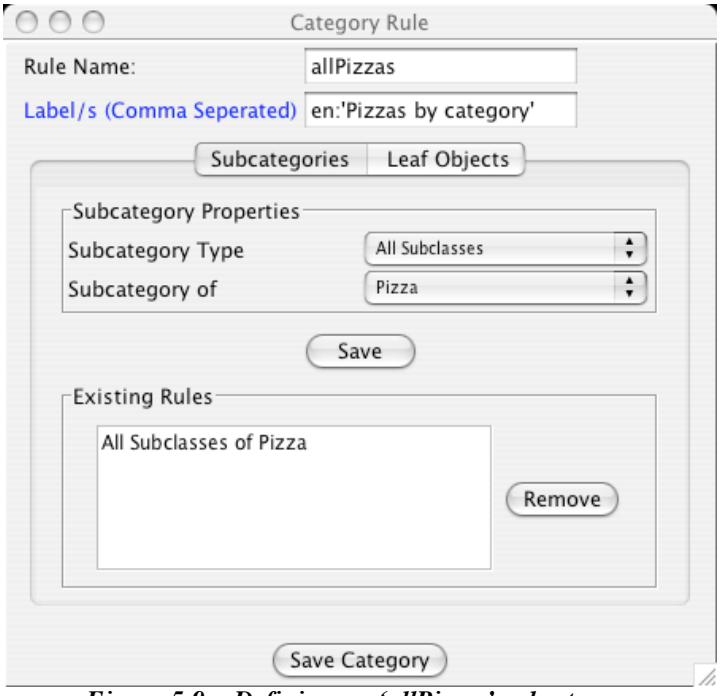

Figure 5.9 – Defining an 'allPizzas' subcategory

Step 3: As we also wish to show any actual pizza items, a leaf rule must be defined. The category rule can then be saved.

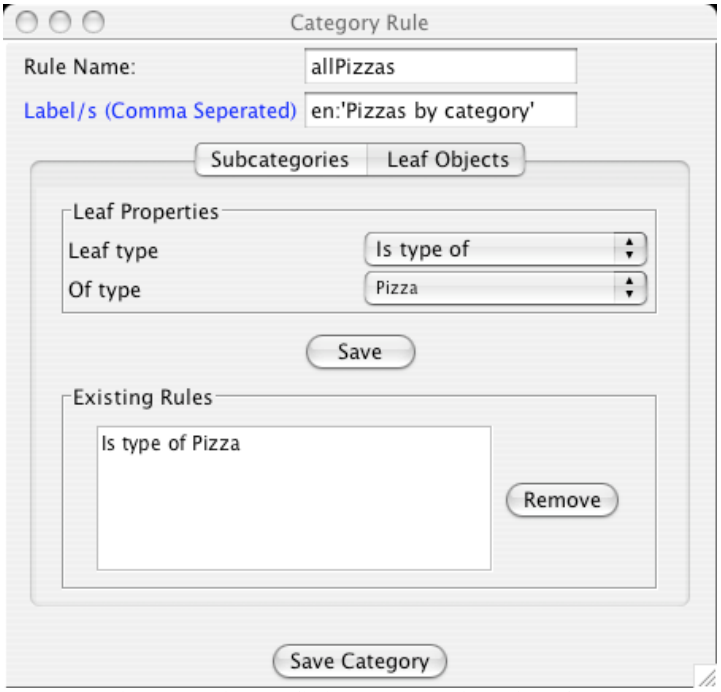

Figure 5.10 – Defining an 'allPizzas' leaf rule

Step 4: We also wish to have a relation rule that will show any other pizzas that originate from the same country when a user is viewing an item.

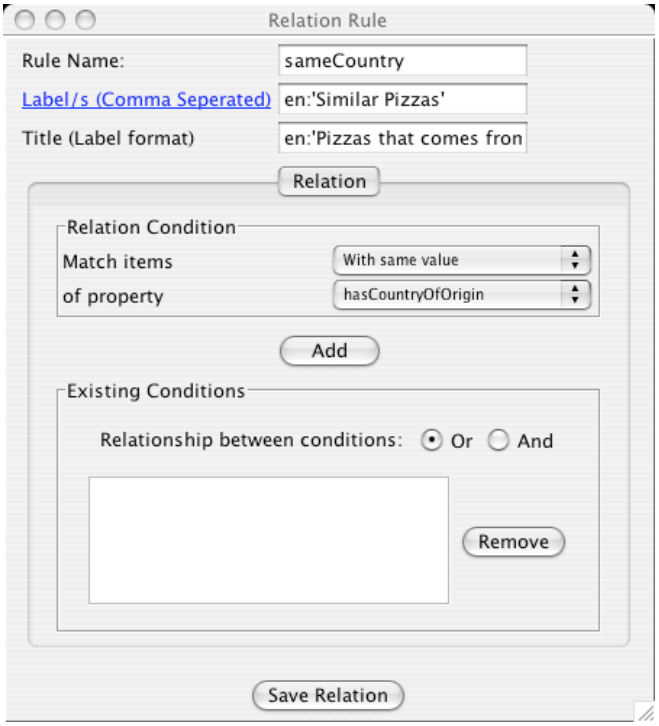

Figure 5.11 – Defining a 'sameCountry' relation rule

Step 5: Once both categories and relations have been defined a rule file needs to be saved and it is then ready for Ontodella to use with appropriate ontology data.

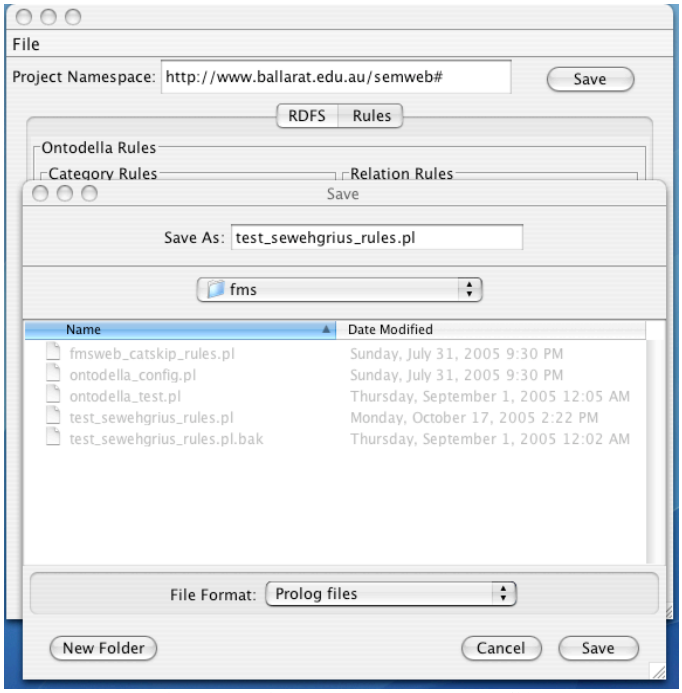

Figure 5.12 – Exporting a rule file

5.6 Other Work

Throughout both the analysis of OntoViews and the development of the referenced solution, a number of changes were made to the Ontodella 'default' package. In some cases this was to allow for easier integration, but in many instances this was to provide Ontodella with the ability to work with content other than the 'default' Museum of Finland data. Because of this, any usage of the referenced implementation should be with the altered version of the Ontodella libraries that are packaged with it.

5.7 Areas of future improvement

Whilst the solution presented within this paper greatly simplifies the steps required by a developer, as well as easing the learning curve for new developers, it does not fully address the stated problem. The following areas have been identified as places where changes may be made in the future to improve the usefulness of this solution.

5.7.1 Additional Prolog Capabilities

In any complex OntoViews portal it is likely that the developer will find that the prolog rules required will go beyond the capabilities provided in this application and they will need to edit the rules source directly. Despite it being impossible to cater for every scenario, there a number of additional, basic rule capabilities that, due to time restrictions, could not be included in the referenced implementation. These include:

- Relation rules that allow for Greater Than (>) or Less Than (<) comparison of values.
- Multi-part rule conditions. The ability to group together different rule conditions (Either for relation or category rules) with different logical relationships would be advantageous in many instances.

  For example: In the case of an online movie store it may be desirable to create a relation rule that links two films where they are:

  1) Of the same genre and have the same director; or

  2) Of the same genre with the same lead actor

- Allowing comparison, again for either relation or category rules, by a logical Not Equal to (!=)

It was known from the beginning of development that covering all possible scenarios was impossible and it was for this reason that the referenced solution was written in such a way that the addition of new prolog capabilities is a relatively simple process. If a scenario can be expressed in a generic manner through prolog then, typically, it will not be difficult to incorporate this into the solution.

5.7.2 Additional application features

In addition to further prolog capabilities, a number of desirable program features were unable to be included in the referenced solution due to time restraints. These do not directly affect the end prolog code being generated, but are features that would improve the ease of use from an end-user perspective. These are:

- Session saving: If a developer is working on a medium to large project, the inability to close a ruleset within the solution and resume working on this at a later time could become very frustrating.
- Rule Editing: Currently once a rule is created within the solution it cannot be edited, only deleted and recreated. The ability to edit an existing rule would cut development time, especially on larger projects.
- Further validation: Whilst there is basic rule validation in the referenced implementation, additional feedback and checking would allow a developer to correct problems with a rule at an earlier stage. This would be of particular benefit to someone using the system for the first time.
- Rule previewer: Once a rule has been created, the ability to apply this rule to a set of data and see the results immediately would allow for on-the-fly testing, significantly reducing the time required to check the rule logic.

5.8 Conclusion

This chapter has presented a solution to the research question of simplifying development for the Semantic Web. Each step in the process of Ontodella rule generation has been shown, as well as how these apply to the final portal product. Areas where future development would provide great assistance to a user have also been identified.

Chapter 6 Conclusion

6.1 Summary of work

The simplification of development for the semantic web would lead to increased uptake in its usage, as well as make it a more viable option than is currently the case. By reducing both the learning curve and the amount of time required in setting up a semantic portal, this technology can begin to enter the mainstream where it will benefit both the end user and developers. To help achieve this goal, this research project asked the following question:

> *"Can a user-friendly semantic rule generator be developed to facilitate smoother adoption of semantic portal technology?"*

To answer this question, an analysis of the OntoViews semantic portal system was first undertaken with the goal of identifying areas where such a product would prove advantageous.

The following aspects of OntoViews were considered those that would benefit the most from additional development:

- Rule generation
- Documentation
- Core application
- Language

It was concluded that the writing of Ontodella selection rules was a significant hurdle for a new developer, in both time and experience, and therefore this was chosen as the primary focus for research.

To satisfy this question, a software package was developed that has the ability to generate complete Ontodella rule files without the need for a user to have any knowledge of the prolog language.

Instead of writing the prolog code required by Ontodella directly, a developer could now specify rules through a GUI environment. This GUI is comprised of elements from the ontology, and a set of the most common rule types that are combined to define the navigational logic of the desired portal. The package is end-to-end in that it is able to generate all the rule types required for an Ontodella instance, and produce a file that will compile within Ontodella without any changes being made.

The package exists completely independently of Ontodella and hence requires no modification before the resulting rule file can be used. However, throughout the development of this system, a number of areas within Ontodella were identified as containing functionality that was specific to the Museum of Finland portal it was initially created for. In some instances this was code that would handle a special case for this content (hence was ignored for all other data), however in other places the core code of Ontodella required alteration to work in a more generic way (i.e. with other ontologies). For this reason a modified version of Ontodella is distributed with the package, as this is the only way to ensure the rules generated will function as expected.

The artefact generated, as a practical representation of the research presented in this paper, demonstrates the ability for development in semantic portal technology to be simplified through a GUI based rule generator. The techniques used within this software product are applicable to any semantic portal system that requires rule development for its logic layer and are not restricted to the OntoViews framework alone. This research therefore, has shown that adoption of semantic portal technology can be simplified through the use of a GUI based rule generator.

6.2 Future work

The software presented in this paper has demonstrated a method for simplifying development on semantic web portals, specifically the OntoViews system. Further improvement to this package could be made in the following ways:

- Investigation of how such a system could be extended to function across a variety of portal frameworks.

- Improve the completeness of the rule library by including additional rule types and comparisons.
- 'On-the-fly' testing of a rule against a dataset to validate the logic it is based on.
- Allow a developer to save an incomplete rule set and resume editing at a later time.
- Including further validation of each rule type to reduce the chance of a rule being invalid when generated in prolog.

6.3 Summary

This chapter has provided a summary of the work undertaken as part of this research project, and a description of further work extending from the results.

6.4 Bibliography

Antoniou, G., and Van Harmelen, F. (2004). *A semantic web primer*. Cambridge, Massachusetts, London: The MIT Press.

Benjamins, V., and Contreras, J. (2002). *Six challenges for the semantic web*. Paper presented at the 8th International Conference on Principles of Knowledge Representation and Reasoning, Toulouse, France.

Berners-Lee, T. (1998). Semantic web road map, *W3C Design Issues*.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

Della Valle, E., and Brioschi. (2004). *Toward a framework for semantic organizational information portal*. Paper presented at the First European Semantic Web Symposium, Greece.

Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W. (2002). *Spinning the semantic web*. Cambridge, Massachusetts, London: The MIT Press.

Horridge, M., Rector, A, Drummond, N., Rogers, J., Knublauch, H., Stevens, R., Wang, H., and Wroe, C. (2004). *Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns*. Paper presented at the 7th International Protégé Conference, Bethesda, Maryland.

Hyvönen, E., Holi, M., and Viljanen, K. (2004). *Designing and creating a web site based on rdf content*. Paper presented at the World Wide Web Workshop on Application design, Development and Implementation Issues, New York.

Karger, D., Quan, D., and Huynh, D. (2003). *Haystack: A platform for authoring end user semantic web applications*. Paper presented at the Proceedings of the Twelfth World Wide\ Web Conference 2003.

Maedche, A., Staab, S., Stojanovic, N., Studer, R., and Sure, Y. (2001). Semantic portal - the seal approach. *Creating the Semantic Web. D. Fensel,\ J. Hendler, H. Lieberman, W. Wahlster (eds.) MIT Press, MA, Cambridge, 2001.*

Makelä, E., Hyvönen, E., Saarela, S., and Viljanen, K. (2004). *Ontoviews -- a tool for creating semantic web portals*. Paper presented at the 3rd International Semantic Web Conference (ISWC), Hiroshima, Japan.

Nunamaker, J., Chen, M., and Purdin, T. (1990). Systems development in information systems research. *Journal of Management Information Systems, 7*(3).

Studer, R., Benjamins, VR., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *IEEE Transactions and Data on Knowledge Engineering, 25.*