Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering

Development of a Semantic Web Solution for Directory Services

Master Thesis 2005

Carlos Buil Aranda

Supervisor: Sari Hakkarainen
Department of Computer and Information Science

Trondheim, June 30, 2005

Abstract

The motivation for this work is based in a common problem in organizations. The problem is to access and to manage the growing amount of stored data in companies. Companies can take advantage with the utilization of the emerging Semantic Web technology in order to solve this problem. Invenio AS is in a situation where it is necessary to access a directory service in an efficient way and the Semantic Web languages can be used to solve it.

In this thesis, a literature study has been done, an investigation about the main ontology languages proposed by World Wide Web Consortium, RDF(S) and OWL with its extension for Web services OWL-S and the ontology language proposed by the International Organization for Standardization, Topic Maps. This literature study can be used like an introduction to these Web ontology languages RDF, OWL (and OWL-S) and Topic Maps.

A model of the databases has been extracted and designed in UML. The extracted model has been used to create a common ontology, merging both the initial databases. The ontology that represents the database in the three languages has been analysed. The quality and semantic accuracy of the languages for the Invenio case has been analysed and we have obtained detailed results from this analysis.

Sammendrag

Motivasjonen for dette arbeidet baserer seg på et felles problem for organisasjoner: hvordan få tilgang på og håndtere store mengder lagret data, og bruken av den voksende Semantic Web-teknologien for å løse dette problemet. Invenio AS har foreslått et tilfelle hvor det er nødvendig å få tilgang til en *Directory Service* på en effektiv måte, og hvor de semantiske webspråkene blir brukt for å løse problemet.

I dette dokumentet har det blitt gjort et litteraturstudie ved å undersøke hovedontologispråkene som har blitt foreslått av World Wide Web Consortium, RDF(S), OWL og OWL-S, samt ontologispråket Topic Maps, som har blitt foreslått av the International Organization for Standardization. Litteraturstudiet kan brukes som en introduksjon til disse tre web-ontolgispråkene.

Databasemodellen har blitt hentet fra og er designet i UML. Modellen har blitt brukt til å skape ontologien som representerer databasen på de tre foreslåtte språkene. Databasen har blitt integrert fra en svensk og en norsk database. Språkenes kvalitet med hensyn til Invenio-eksempelet har blitt analysert, og vi har oppnådd detaljerte resultater fra denne analysen.

Sinopsis

La motivación de este trabajo esta basada en un problema común a todas las empresas. Este problema es el acceso y la gestión y acceso a grandes cantidades de datos y la posible utilización de la Web Semántica para resolver dicho problema. Invenio AS ha propuesto una situación práctica donde es necesario el acceso a un servicio de directorio de forma eficiente y la posible utilización de lenguajes relacionados con la Web Semántica para resolverlo.

Este informe contiene una investigación teórica y un análisis de los lenguajes estudiados. Estos tres lenguajes son recomendaciones propuestas por el World Wide Web Consortium (W3C) y la Organización Internacional para la Estandarización (ISO). W3C ha recomendado los lenguajes Resource Description Framework Schema (RDF(S)), Web Ontology Language (OWL) y OWL-S e ISO que ha recomendado Topic Maps . Este análisis teórico puede ser utilizado como introducción a estos lenguajes RDF(S), OWL y Topic Maps.

Se ha extraído el modelo de las bases de datos propuestas en el caso de estudio de Invenio y dichas bases de datos se han modelado en UML. Los modelos creados se han unificado en uno que ha sido utilizado para la creación de la ontología en los tres lenguajes propuestos. La calidad de los lenguajes utilizados ha sido analizada y se han obtenido resultados concluyentes de dicho análisis.

Acknowledgements

I would like to acknowledge the effort and dedication to my project of my coordinator, Sari Hakkarainen. Without her dedication, time and counsels I would never have arrived to the final document of this Master's Thesis. Her help is priceless. I would like also to thank Anders Kofod-Petersen for his help and guidelines in this project. He and Invenio AS provided the case study of my Master's Thesis.

I would also like to thank Jennifer Sampson for her help during the design of the ontology, Csaba Veres for his help with Protégé and the several query languages and Darijus Strasunskas for his help when I arrived to the Department of Computer and Information Science and his books. I would also like to thank Ellen Solberg and Berit Hellan for their help with all my questions at IDI.

I would like to thank my old Erasmus coordinator Toni Cortes and the new Erasmus coordinator Nuria Castell for giving me this great opportunity in my life. I would also like to thank Ramón Sangüesa for his counsels for choosing a university.

Finally, I would like to thank my parents, Carlos and Concha for their help, patience and support in every day of my year in Trondheim. Without them this would have been impossible.

Table Of Contents

List of Figures

List of Tables

Part I

Introduction to the Case Study

Chapter 1

Introduction

This introduction gives an overview to the problem of accessing and interpreting machine generated data and possible solutions to it. Further the objectives, way of working, expected result and the outline of the report are presented.

1.1 Background

A major problem today is to access to the information stored in our organization. Each day we have more and more information generated by machines. The data flow ranges from the huge repositories of data used to store Acme's staff information to the data that is possible to obtain from databases with the history of sales and transactions generated by some customers via one Web Site. That generates millions of information megabytes and it is important to manage it efficiently, without waste of time and resources. That implies a need to use the most adequate technologies.

In [8, 13], a study of the viability to adapt two databases containing the directory service for Sweden and Norway was done. An analysis of three Web ontology languages RDF(S), OWL and Topic Maps, and their possible use for accessing in an intelligent and quick way by an informatics agent was completed. We arrived to the conclusion that the most useful language to solve the problem was OWL due to its capacity for creating logical rules that will enable the agent to create better answers to our queries. But is also necessary to improve the design and the mapping from the databases to the ontologies. This is necessary due to the necessity to add more information to the database about the classes and to better organize the model in order to provide a better reasoning from the information agents. Then it is necessary to create a more knowledge model oriented rather than data ontology oriented one. If the data is not modeled with correctness a process of adjustment is needed.

1.2 Problem

The management of the stored data becomes critical for obtaining business results. The data we have, forms the basis for the knowledge we can obtain and we must manage it in an optimal way. Information management will be the key for the success of any knowledge intensive company, because proper use of the data gives information about the system (via ERP[1], Data Mining, CMS[2]. etc) will be the key access point for the other components of our system like DSS[3], EIS[4] or CRM[5].

The main technical problem that we can see in this situation is to translate the databases, with a limited amount of useful information for an information agent to the Web ontology languages format that is OWL, RDF(S) and Topic Maps. Also an enrichment of the databases with first order logic rules and data is needed.

1.3 Objectives

The objective is to support organisations in selection and feasibility of emerging semantic Web technologies. One subgoal is to adopt an existing evaluation framework for analysis of the actual semantic Web technologies for representing knowledge. Another subgoal is to provide a proof of concept to a particular organization by implementing and testing trial-ontologies in their specific domain, i.e., knowledge management for directory services.

1.4 Scope

The IT industry is currently changing focus from providing storage, processing and network services to providing knowledge intensive information and services to large numbers of customers. The diversity and multitude of resources and applications on the Web places elaborate requirements on methods and tools for efficient generation, manipulation and compositional usage of information and services. Metadata, ontology/domain model and semantic enrichment can bridge the heterogeneity and facilitate the efficient usage of information assets on the semantic Web [4]. However, a formal, standardized representation of signs and meaning is required [37] for supporting ontologies, i.e. explicit and shared conceptualisations [11] of the domain.

In [8, 13], a study of the viability to adapt two databases containing the directory service for Sweden and Norway was done. An analysis of three Web ontology languages RDF(S), OWL and Topic Maps, and their possible use for accessing in an intelligent and quick way by an informatics agent was completed. We arrived to the conclusion that the most useful language to solve the problem was OWL due to its capacity for creating logical rules that will enable the agent to create better answers to our queries. But is also necessary to improve the design and the mapping from the databases to the ontologies. This is required due to the necessity to add more information to the database about the classes and to better organize the model in order to provide a better reasoning from the information agents. Then it is necessary to create a more knowledge model

[1]Enterprise Resource Planning
[2]Content Management System
[3]Decision Support System
[4]Enterprise Information System
[5]Customer Relationship Management

oriented rather than data ontology oriented one. If the data is not modeled with correctness a process of adjustment is needed.

The problem is a complex combination of available information resources and requirements. We have an exhaustive amount of information, the accessing is required to be quick, we should be able to locate the data that we need in every moment and the results of the query should not have wrong answers. The manipulation of this information and the access to the data will be the core of this report. That manipulation is about accessing data with different structures using the Semantic Web Technologies.

It is insufficient just to translate the data from one representation to another. As it is referred in [38] this approach grows from the insufficiency of the structure resemblance in order to make the semantics of terms in a database schema clear. The ontology is used to define terms from the database and is linked to the information by the term defined. It is necessary to add extra information that should be processable by the information agents because if the agents do not have this information will not answer correctly to the users' queries.

1.5 Technological Scope

Today there are two kinds of ontology languages, the traditional Ontology specification languages like CycL, Ontolingua or OCML and the Ontology languages based on the emerging Web Standards[35]. In this report the emerging second kind of languages will be studied. The languages currently dominating proposed by W3C (OWL, RDFS) and by ISO (Topic Maps) have been selected.

The technological solutions used in this project are Java for creating the parser between the databases, the available semantic Web technologies for creating the ontologies, the Definition of Terms approach for mapping the databases and the ontologies as described in [38].

The motivations for selecting Java as technology for developing the parser which will translate and add information from the databases to the ontology was the ease it provides for managing strings. Java provides several functions and commands for developing quickly a parser for managing strings. The databases were in two different formats, XML and txt, therefore we need a language which provides enough properties for managing them. We thought also about developing the parser in C++ due to its speed processing or CLIPS but finally, Java was selected.

The main function of the developed conversion parsers was to translate the data from one representation into another, not only by copying the data, also by adding more concrete details that can be inferred from the data in the database. To do that it is necessary to read the data from the files and to add some specific information and tags into this information. When this task is done, it will be possible that an information agent that is capable of processing and reasoning using RDF(S), OWL and Topic Maps. This data should be read and processed as strings for being stored in a string format. For this situation Java provides the most adequate functions and procedures, better than the other languages that had been considered. An example of the suitability of java for our task is that the first parser that we made, a parser for RDF(S) is a one hundred lines program.

The tool selected for developing the ontologies in the three languages was Protégé 3.0. Using the same tool for creating the ontologies provides the objectivity required to make an analysis based in the ontology languages without external influences. Also, it is out if the scope to investigate alternate editors or ontology language tools.

1.6 Way of Working

In this report we will describe the ontologies created (a first version of these ontologies has been done) and the parser developed for mapping the databases and the ontologies. The description of the several test and analysis of the ontologies will be described. We have added to the study an analysis of OWL-S. Below, we describe the steps in our research method in more detail.

- **Literature Study**. First we look to adequate references for the kind of investigation that we will do. Due to the topic of Semantic Web being relatively new there are not enough books about it, hence we look for articles and specifications. We have read key articles like [4] or [27] and the current specifications at W3C or ISO. There are some exceptions however. Our knowledge of RDF has been obtained from the books *Practical RDF* [31] and *Creating the Semantic Web with RDF [15]*. Those books mainly contain RDF explanations and examples of RDF but also make some reference to OWL.

- **Re-engineering of the Databases: schema level**. The second step is to improve the initial model extracted from the databases. To do that we will investigate another model for directory services databases and we will make an interview with the technical expert of Invenio AS.

- **Domain level**. The third step is to create and improve the parsers between the databases and the ontologies. Initially we started creating the basic ontology in RDF and later in OWL and Topic Maps. To test the ontologies we developed a parser that is a bridge between the databases and the ontologies.

- **Bringing the databases with the ontologies**. In the fourth step we will test the parser and the model of the databases with the ontologies. We will make test to guarantee the maximum correctness of the model.

- **Databases and ontology test**. In the fifth step we will test the ontologies.

This report has three main parts: one theoretical approach to the problem where the requirements are analysed and a solution proposed, one practical approach where the solution to the problem is tested by using several quality test and another theoretical section where the languages proposed are described.

1.7 Expected Results

The main result of this report is a description of the process done for creating and testing the access to a database. The way of working for arriving to these final results and the way for converting two databases to an ontological representation is explained.

1.8 Outline of the Report

The report is divided in four parts. These parts contain the main sections of this report. Inside of these parts it is possible to find all the work done in this research process.

Part I includes the first four chapters including this one. In Chapter 2 the different languages used for creating the ontologies are described. A brief description of these languages and an introduction to the Semantic Web Services is provided. In Chapter 3 the process of research used is described. In Chapter 4 the three Web ontology languages analysed and an extension for Web services for one of them are described.

Part II includes Chapter 5 and Chapter 6. In Chapter 5 the three Web ontology languages are analysed by using a theoretical framework for evaluating the quality of languages. It is also indicated how the elements of the frameworks affect to our particular case. In Chapter 6 the comparison of the ontology designed within the three Web ontology languages is described. This evaluation has been done applying the languages to our particular study case.

Part III includes from 7 to 9. In Chapter 7 the conversion parsers developed for our case are presented. This parser are based in the solution proposed in Chapter 3. In Chapter 8 the design of the model for the new ontologies and databases is presented. The model of the databases and the new ontology created from them is described. In Chapter 9 the tests that we have been done to the ontologies and the design are presented. It contains an analysis of the transformation done to the initial database model and the comparison with the new ontology in the three languages selected.

Part IV contains the chapter dedicated to the conclusions and appendixes. In Chapter 10 the conclusions of the work done and the future work recommended after our investigation are presented. The Appendix F

Chapter 2

The Semantic Web

Semantic Web is the next generation of the Web. Actually the World Wide Web is made for people, not for machines and thus, in a society with machines and computers present everywhere the Web is not efficient enough. Now, Internet contains an exhaustive amount of bytes of information about seemingly endless amount of different topics but this information is only accessible by humans. Our computers which are made with the last processors technology can not access it. Semantic Web will improve this problem by complementing our data by adding more significative data which an information agent will understand, that is, give a meaning to the information that we are sharing and then an application will comprehend it. The Semantic Web will bring structure to the meaningful content of the web pages. Creating an environment where software agents roaming from page to page can readily carry out sophisticated task for users [4]. In the next pages we will se the most important components of the Semantic Web.

2.1 The Semantic Web

"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation".[4]

Above Semantic Web was described as a technology that can mainly be used on Internet, but this is not true. Semantic Web Technologies can be used in many different situations. It is easy to think about how many types of information a company can manage, e.g. one Business Web Portal, an ERP, one ECM. That means lots of information, and that information with meaning, through Semantic Web Technology, machines will be able to manage it, selecting the information useful for us and discard the others.

The representation of the knowledge that we have, i.e., the stored data that we can access, is one of the key concepts of the development of Semantic Web technologies. The information that should be accessible for the information agents must be structured in an optimal way. It can not be too complex because our agents will find difficulties for processing it and must be structured enough to make the data understandable. Therefore we have to create some logics to our Web, and further, we have to add some inference rules to help to get the results wanted. The challenge of the Semantic Web is to provide a language that formally expresses both the data and the rules for reasoning about this data.

When structuring the information, we must think that the information should be available for all and that it should be structured in an optimal way. Once selecting the structure we must be sure that one machine is capable of understanding it in the right way, e.g., an information agent can interpret that the attribute red of a car means that it is the color of the car but also is possible that the attribute color can have a different meaning.

2.2 The Semantic Web Structure

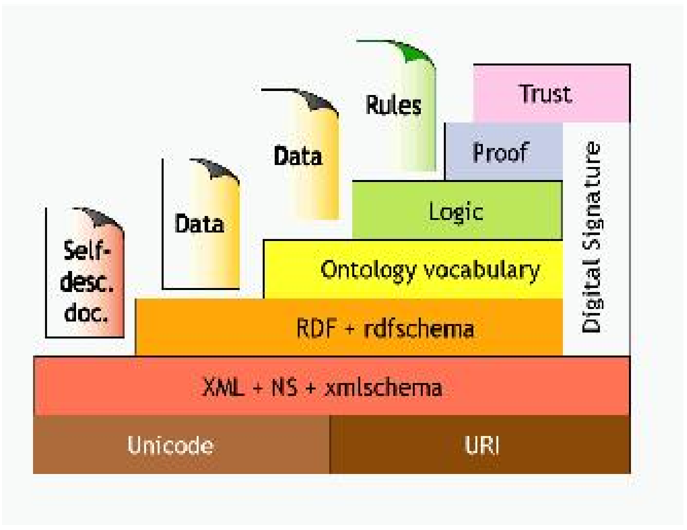


Figure 2.1: Semantic Web - XML2000 by Tim Berners-Lee

Figure 2.1 shows the Berners-Lee tower about Semantic Web (http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html). In the tower the main elements of the Semantic Web are shown and we can distinguish different layers on it. The bottom layer supports the basic coding of characters and the resource identifier. In the next layer, the structure of the data is represented. These are the most basic levels of the Semantic Web. In the next layers we can see the key concepts of the Semantic Web, RDF and the Ontology Vocabulary. These two layers are expected to be the core of the Semantic Web. The last three layers, logic, proof and trust are not in the topic of this report.

2.2.1 Information Carry Layer

The Uniform Resource Locator, URI, is the key for naming resources. These resources are accessible by protocols and an address like http://www.idi.ntnu.no that URL is a concrete example of URI. URI's provide a common syntax for naming a resource regardless of the protocol used to access the resource [31] .

We have to realize that URI is an identifier, not only an address where we can find a resource and then one URI can be used also like a name.

2.2.2 Transfering Layer

XML is the language used to structure the information and it is based in assign one identifier and the value for the identifier. We have seen before that this is not enough for solving the problems of the Semantic Web. The information we use should have a meaning, not only data, and that is the reason because XML is in the lowest levels of the pyramid. XML is the base, and using this base we will build RDF, RDFS and Web Ontology languages.

2.2.3 Metadata Layer

In the metadata layer are two main concepts that support the ontology layer. These concepts are RDF and its extension RDF(S). RDF is a data model and with this model we will represent the information which is formatted in XML to provide the basis for the next layer, in order to provide the structure that will be possible create the rules for reasoning about data.

RDF(S) is the approach of the metadata layer to the ontology lager. RDF(S) is based in RDF and provides elements for modeling data and adds to this data the basic information. Within RDF(S) we are able to create the basic structure to share data and make it available to the most simple information agents.

2.2.4 Ontology Layer

In this layer an Ontology language is provided. This layer provides more expressiveness than the previous layer by adding this new language. With the new language developed by W3C, OWL, the ontology layer provides the elements needed to add enough expressiveness to our data in order to provide the information that the information agents need for reasoning about this data. OWL is the Web Ontology Language and it is builded on top of RDF(S) like the Ontology layer is on top of the metadata layer. OWL is a language for describing data, because an ontology formally defines a common set of terms that are used to describe and represent a domain [31] .

2.2.5 The Other Layers

The layers on top of the Ontology Layer are:

- **The Logic Layer** extends the functionality of the previous layer on writing logics to our data.
- **The Trust Layer** improves the security by adding tools to detect alterations in the documents.

- **The Proof Layer** evaluates the applications to apply the adequate mechanisms when it is necessary to give trust or not.

These three layers are not covered by this report and more works its being doing at the moment to develop the last layers of the Semantic Web.

2.3 Semantic Web Applications

Recall that the Semantic Web is expected to allow us to find the information that we need quickly and with accuracy, obtaining just the answers to our queries that we really want. In short we will forget the current situation where we get hundreds of answers from our queries and that requires a re-search in all these data once we have it.

But this is only the top of the objectives of the Semantic Web activity. With this emerging technology a new kind of approximation to the applications that we use every day will be achieved. We will get more facilities by the integration of the information that our applications are using now, which will allow our machines to communicate between them. The Semantic Web will provide an infrastructure that enables not just web pages, but databases, services, programs, sensors, personal devices, and even household appliances to both consume and produce data on the Web. Software agents can use this information to search, filter and prepare information in new and exciting ways to assist the Web user. New languages, making significantly more of the information on the Web machine-readable, strength this vision and will enable the development of a new generation of technologies and toolkits[14].

2.4 Concluding Remarks

In this chapter, we have seen that the Semantic Web is an emerging technology that will able us to accomplish our objectives in the search and retrieval of information. The Semantic Web, due to its layer division can be accessible for lots of user, from the users that only want to put their information into the Web and make available that information to the community and to the information agents to the companies that want to improve their knowledge about the their data. We have also seen some examples of the use of the Semantic Web Technology.

Chapter 3

Enquiry for the Problem

In this Chapter, we will describe the problem proposed and we will explain the way-of-working to solve it. We will propose several steps to achieve our goal and we will explain similar approaches to our approach.

3.1 Our Problem

The problem is to adapt two databases with a concrete format to an ontological representation by using Web ontology languages. These languages and the case study have been proposed by Invenio AS, the company partner in this project. We want to improve the access to two databases containing the directory service of Norway and Sweden. We will improve the access to these databases by giving more semantics and expressiveness to the data. With this extension an information agent will be able to fulfil the requirements of the users.

3.2 The process of enquiry

The process of enquiry has been detailed in the Picture 3.1. This process has been divided into two main processes of research: one more theoretical approach to the problem where the ontology languages are evaluated by using the Semiotic Quality Framework [19] and another one by using a more technical approach. In this more technical approach a translation process between the databases and the ontological representation has been developed.

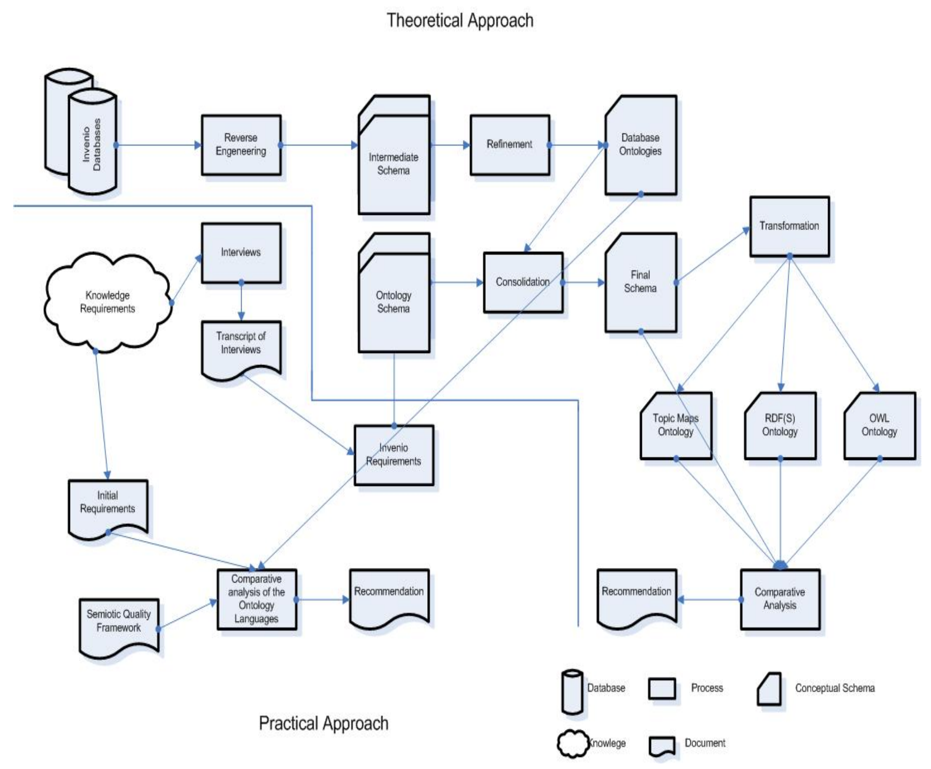


Figure 3.1: The process of enquiry

3.2.1 Theoretical Approach to the problem

This approach for solving our problem contains a strong theoretical base. It is based on the Semiotic Quality Framework [19] and the descriptions of the languages explained in Chapter 4. The following steps have been used in order to accomplish the analysis.

Reverse engineering step. In the first step we extract the model from the databases. These databases are two, one contains the Swedish directory service for persons and the other one contains the Norwegian directory service for companies. The Swedish database is in XML format and the partner provided the DTD model. The Norwegian database is in text format and it does not have any model. We created Swedish the model from the DTD file and the Norwegian model were extracted by using reverse engineering on the database. It was a complex approach and we based the new model in the Swedish database model because both databases had the same provider and the creator of them probably was the same.

Refinement step. In the second step we refine the models extracted to an ontological representation. Once we have the initial design of the databases we are able to translate the database models to the three Web ontology languages. The output was three ontologies for each database in RDF(S), OWL and Topic Maps.

Analysis step. In the third step we analyse the three Web ontology languages by using the Semiotic Quality Framework [19]. We have analysed the quality for modelling following the seven appropriateness specified in the framework adapted to the specifications of the problem and the languages. The requirements of the company in our particular case have also been analysed with the same quality framework because we will obtain the same measure of the requirements of the framework. Therefore will be easier to compare the requirements of the company and the characteristics that the languages offer. Once the comparison has been done we proceeded to recommend the Web Ontology Language (OWL) for the best solution to the proposed problem.

3.2.2 Practical Approach to the Problem

This approach to the enquiry problem has a more practical section because we compare the languages doing queries to the ontology. This time both database designs have been merged and we only have one design. The following steps have been used in order to accomplish the analysis.

First we gathered the requirements of Invenio AS by interviewing the domain expert from the company. He explained us what the databases are, why are designed in this particular way and what the attributes are.

Second we transcript the requirements from the company and with this transcription we design the new databases. The requirements for the two databases, the Norwegian and the Swedish are quite similar and we can merge them. Also, the requirements of the company include a possible extension of the ontology. Initially there will no values related between Persons and Companies but the relationship will exist.

Third, once we have the ontology we consolidate it with the ontologies extracted from the reverse engineering of the databases. We put the requirements and the physical data together to create the final ontology.

Fourth we write this ontology in the Web ontology languages selected and we do the comparative analysis of them. From this analysis we will extract the final recommendation.

3.3 Need for a Parser

A converter is needed in several situations. Our situation is similar to others where a large quantity of data is stored using a relational database technology. An example of parser between a relational representation and an ontological representation is D2R [10]. Relational database technology can not produce enough good answers to our queries. Due to this technology is based on relational algebra it is needed a certain knowledge about it in order to make queries and therefore it can not be done by everyone. A convertor can be used to translate the data in a particular representation to another one. This can be useful, for example, to translate a relational database from one relational representation into another one.

But this will require fitting both representations. The two representations will have different ways to represent one fact and this problem will need to be solved. To solve this situation where two different representations have different ways to represent the data we will need a converter. In this converter we will need to specify the translation rules from one representation to the other one and try to put all the meaning that the original representation has into the other one. These will we the main aim of our converter: to put all the meaning from one representation into the other one and add the representing power of the second one to the converted data.

Our concrete problem is to translate the data from a relational database technology representation into an ontological representation. This last representation will be in three different ontology languages: RDF(S), OWL and Topic Maps. We will develop a converter between these two representations trying not to loose the properties that the data has in the relational schema and add the properties for describing data that the languages have.

3.4 Similar Approaches

The necessity of a parser is also due to the content of a database-driven web sites is not machine understandable. Consequently they are not a part of the Semantic Web and we have to "translate" our data to the proper format. To do this translation we have to select among two representations. We have our data in a Relational database model and we have to change this model to another representation, in Frame Logics or RDF Schema. We have to select the best way to preserve the maximum amount of information possible and as it is done in [3] we will use the frame representation with RDF Schema. In our system, ontologies are conceptual models with an abstraction level higher than the schemas of any database. Ontologies give a homogenous description to different schemas of databases integrated in the system. Also there are several projects like [7] or [23] that uses this approximation. As we explained before and as it is explained in [7] using ontologies will give to our system the advantages of provide logical and physical independence between layers in the system It will increase the scalability of the system, will reduce the changes that have to be done to the original databases if it is necessary to modify it and will make more user friendly the access to the information.

We can also find several projects that extract the information from web sources or unstructured data sources like [18] or [1]. Also these situations require developing parsers for gathering information and translating it to an ontology representation with the difference that they gather unstructured data instead of the structured data of one database.

3.5 Concluding Remarks

In this Chapter, we have seen which will be way for solving our problem. We have made a planning decided two approaches to our problem, one theoretical and another one more practical. We have seen we need some conversion parsers to translate the original data to the new representation models. Finally, we have to remark that our problem is not the only one in the IS community. There are several cases like this one.

Part II

Theoretical Approach to the Case Study

Chapter 4

Overview of languages

In this Chapter, we are going to describe the three Web ontology languages selected and an extension for one of them, OWL. The description of the languages will consist in a short introduction to their functionalities and their instructions.

4.1 Overview of Resource Description Framework (RDF)

The Resource Description Framework is a framework for representing information in the Web. RDF is developed by W3C and provides meaning to data in a machine understandable format allowing for more sophisticated data interchange or searching.

If we look at the W3C web page we can see this definition: "The Resource Description Framework" (RDF) integrates a variety of applications from library catalogs and world-wide directories to syndication and aggregation of news, software, and content to personal collections of music, photos, and events using XML as an interchange syntax. The RDF specifications provide a lightweight ontology system to support the exchange of knowledge on the Web.

RDF will allow us to put information and meaning to our data. RDF is extremely flexible for accomplishing that objective because it will allow us to put the information in one context with enough extra information that an information agent will be capable of process and understand.

If RDF is a way for describing data the RDF Schema is a domain-neutral way of describing the metadata that can then be used to describe the data for a domain-specific vocabulary [31]. RDF Schema provides the resources necessary to describe the objects and properties of a domain-specific schema.

4.1.1 RDF Core

The core of RDF is a set of triples consisting in RDF Subject, RDF Verb or Predicate and RDF Object. The first principal component of RDF is the subject. The subject can be seen as a name or an object. The subject is the resource being described ant it can be identified by an URI. The second principal component is the verb or a property of the subject. The verb is a characteristic of the subject and for example, it can be color, size or another property applicable to a resource. Properties can also be multiple resources, values of properties can be other resources. The third and last component of the RDF triples is the object. This object is the value associated to this resource, for example can be red, big or another value applicable to a defined property. In every RDF triple we can see always:

- Every RDF triple is made of subject, property and object.
- Every triple represents one fact.
- Every RDF triple can be joined with other RDF triples and will not loose their initial meaning.
- A Subject is an URI

RDF can be represented in a graph way, like in the Figure 4.1, a directed labeled graph and is the way that RDF Core Working Group decided as default method for describing RDF data models.

There are three different kinds of nodes in a directed graph for representing RDF data models:

- **Uriref node**. Consist in a Uniform Resource Identifier, that is, an identifier for the node. Can reference to data, not only to Web resources.
- **Blank nodes**. Nodes that do not have URI.
- **Literals**. Formed by three components, a character string, an optional language tag and data type.

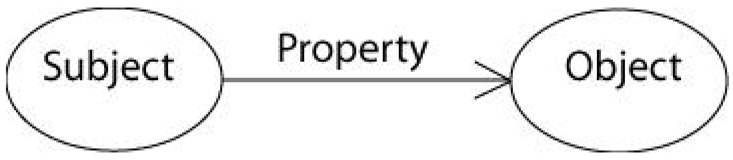


Figure 4.1: RDF Graph

4.1.2 RDF and XML

RDF uses XML for the syntactic expression of model instances [20], it builds a layer on top of it, making interoperable exchange of semantic information possible. A RDF document must be XML well-formed but not XML-style validity and, of course, all the requirements that RDF have. One example of an RDF representation in XML is:

```
<rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

4.1.3 RDF Schema, RDF(S)

RDF Schema defines a simple modelling language on top of RDF. In RDF you can represent the data, with their properties but you can not represent the description of these properties or describe relationships between these properties and other resources. To solve this problem W3C specified RDF Schema [6]. It is introduced as a layer on top of the basic RDF Model.

RDF Schema is a domain-neutral way for describing metadata. This metadata can be used to describe the data for a domain specific vocabulary. RDF(S) helps us to create and define new objects and properties, with RDF(S) we will define classes and properties that may be used to describe classes, properties and other resources[20].

Resources may be divided into groups called classes. The members of a class are known as instances of the class. Classes are themselves resources. They are often identified by RDF URI References and may be described using RDF properties. The `rdf:type` property may be used to state that a resource is an instance of a class.

`rdfs:Resource` This is the main class. In RDF all things described are Resources, all other things in RDF are subclasses of `rdfs:Resource` and also `rdfs:Resource` is a subclass of `rdfs:Class`.

`rdfs:Class` This is the class of resources that are RDF classes. `rdfs:Class` is an instance of `rdfs:Class`.

`rdfs:Literal` This is the main class for types like strings or integers. The literals can be plain or typed (an instance of datatype class).

`rdfs:Datatype` This is the class of datatypes and it is a subclass of `rdfs:Literal`.

`rdf:XMLLiteral` This is the class of XML Literal values, it is an instance of `rdfs:Datatype` and a subclass of `rdfs:Literal`.

`rdf:Property` This is the class of `rdf:Property` and it is an instance of `rdfs:Class`.

`rdfs:range` This property (it is an instance of rdfs:Property) is used to say that the values of the property are instances of one or more classes. Also that property can be applied to it self.

`rdfs:domain` This property (it is an instance of `rdf:Property`) is used to indicate that a resource that have a determinate property is an instance of one or more classes. For example the `rdfs:range` can also be applied to properties using `rdfs:domain`.

`rdf:type` This is a property used to indicate that a resource is an instance of a class.

`rdfs:subClassOf` This property is used to indicate that all instances of one class are instances of another class. This property is transitive.

`rdfs:subPropertyOf` This property is used to indicate that all resources related by one property are also related by another.

`rdfs:label` This property is used to provide a readable version of a resource's name.

`rdfs:comment` This property is used to write comments or descriptions.

4.1.4 Problems in RDF(S)

When you start to design a basic ontology with RDF(S) you will make sense of you can create infinite layers of classes. It is possible to observe that `rdfs:Class` is a subclass of `rdfs:Resource` and `rdfs:Resource` is at the same time an instance of `rdfs:Class`. The problem comes when the next layer, the Logical Layer, tries to extend the previous layer, the metamodel layer. These problems are described in [27] and the result is that RDF(S) has no clear semantics:

1. The class `rdfs:Class` is an instance of itself. That means that you can find the Russell's paradox. The paradox arises when considering the set of all sets that are not members of themselves. Such a set appears to be a member of itself if and only if it is not member of itself, hence the paradox.
2. The class `rdfs:Resource` is a superclass and instance of `rdfs:Class` at the same time, which means that the superset (`rdfs:Resource`) is a member of the subset (`rdfs:Class`).
3. The properties `rdfs:subClassOf`, `rdf:type`, `rdfs:range` and `rdfs:domain` are used to define both the other RDF(S) modeling primitives and the ontology, which makes their semantics unclear and makes very difficult to formalize RDF(S).

4.2 Overview of Web Ontology Language (OWL)

When we start to work with RDF(S) we see that it does not have all the constructs and restrictions we need. For example, RDF(S) has a low expressive power (RDF is a data model), it is not possible in RDF(S) to provide defined classes or provide more complex restrictions or properties like universal or existential quantifiers. The ontology language is the next layer in the Berners-Lee tower and OWL is the language chosen by W3C. OWL extends RDF(S), uses the RDF constructs and adds new constructs, mainly to give more semantic information to the data, adds more vocabulary for describing properties and classes. OWL is a knowledge representation.

But there is an interesting question between RDF(S), OWL and intelligent agents. One intelligent agent can find more conclusions looking into RDF(S) formatted data than looking into OWL formatted data because with OWL we will specify the data with more constructs and we will get more accurate results, but since OWL is based in RDF(S) and put constrains on top of it an intelligent agent that is looking also in RDF(S) data will get more conclusions with this data because will find less restrictions and will process more data. That is one of the main questions about the Semantic Web Tower, you will extend one layer adding constraints to another layer, but in certain cases you will drag some problems from the previous layer as we saw in the previous section 3.1.4.

4.2.1 OWL, Global Overview

OWL's objective is to provide enough constructs and restrictions creating better quality data for our intelligent agents. But since ontologies are business models [31] we need more flexibility in an ontology because there exists different concepts of business and different sizes of business. Hence OWL is divided in three levels: OWL Lite, OWL DL and OWL Full. The relation between the three types of OWL is shown below but the have the same basic constructs. A more detailed specification of the elements explained here can be found at [9]. The figure 4.2 represents the previous sentences.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

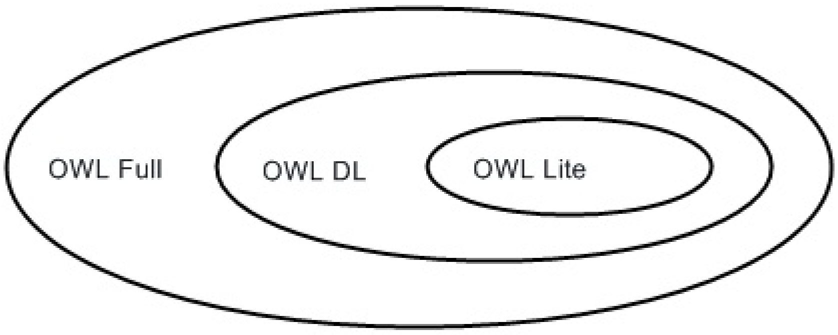


Figure 4.2: OWL Layers

4.2.2 Basic Elements

Namespaces The vocabulary we will use in out ontology is the most important thing. One of the main characteristics of one ontology, or at least one of the principal recommendations for creating an ontology is the ontology interoperability. To facilitate this, the vocabulary used is critical because will allow us to share concepts or at least provide primitives for relating different representations.

OWL Header The ontology header will provide the start and finish point of the ontology, version control, including another ontologies or comments:

```
<owl:Ontology rdf:about="">
    <rdfs:comment>An example OWL ontology</rdfs:comment>
</owl:Ontology>
```

Within OWL the header of the ontology we will manage the version of the ontology, the imported definitions required in this ontology, importing another ontology (this import should be coordinated with the namespace previously indicated). The ontology header declaration will finish with

```
</owl:Ontology>
```

and the next section will start.

4.2.3 Classes and Individuals

OWL defines classes via properties [31] Individuals are an important member of ontologies. The intelligent agents will reason about individuals and the information contained in those individuals. It is important to provide enough restrictions, descriptions and properties to these individual to get a better answers from our agents. Each individual is a member of the class `owl:Thing` or is a member of the empty class `owl:Nothing`. The OWL class taxonomy is detailed in Figure 4.3 [1].

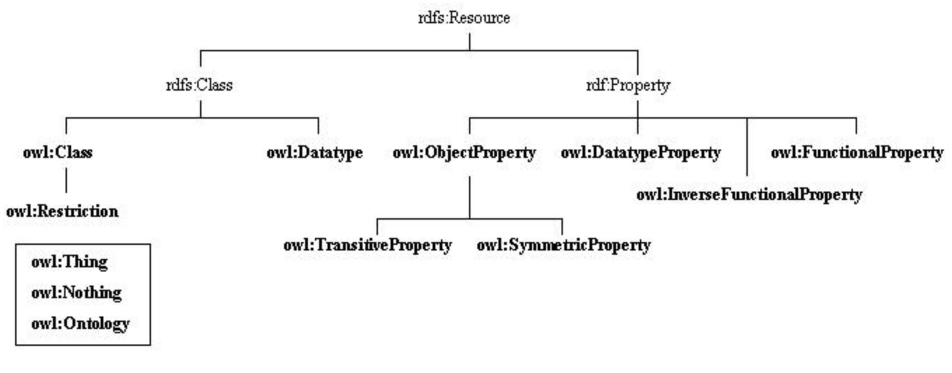


Figure 4.3: OWL Class Taxonomy

[1]http://lists.w3.org/Archives/Public/www-webont-wg/2003Jan/0169.html

Declaration of classes is not complex. It is only needed declaring a class specifying the name. To specify a subclass it is only needed to indicate the name and the superclass as detailed in the following examples:

```
<owl:Class rdf:ID="Winery"/>
 <owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>

<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>
```

These classes are incomplete because we do not know anything about them. In OWL we have to provide restrictions and properties to these classes and its instances. In OWL classes are only names with some properties for describing individuals, then individuals are entities that can be grouped in classes. Example of Individual:

```
<Region rdf:ID="CentralCoastRegion" />
```

For describing the members of a class (the individuals) we have to at least declaring it of a member of a class. `rdf:type` is an RDF property that ties an individual to a class of which it is a member.

4.2.4 Properties

Properties are used to give meaning to our classes and individuals. Without properties classes are useless. A property is a binary relation and we will focus on the object properties, i.e., the properties among instances of two classes. We are not interested in relationships between RDF Literals and instances of classes (datatype properties). Example of Property:

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

Like classes we can define solely a property or define a sub property from another property. This is a specialization from the main property.

Property Characteristics

Properties can be related with other properties by using the instructions indicated below. This provides a good mechanism in order to correctly specify the relationships, by relating them with other properties or by assigning global constraints to these properties.

TransitiveProperty This property allow us to represent the Transitive property for the indicated resources.

SymmetricProperty This property allows us to represent the Symmetric property between two resources.

FunctionalProperty This property allows us to represent the Functional property between two resources.

InverseOf This property allows us to represent the Inverse property between two resources.

InverseFunctionalProperty This property allows us to represent the Inverse Functional property between two resources.

Property Restrictions

We can specify the range of values that we want to apply to the properties. Maybe we are more interested in apply the property hasChild to the individuals that belongs to the class humans but adding the restriction to those humans that have more than 15 years or restrict the number of elements that a class should have.

allValuesForm This property specifies the universal quantifier. Specifies that for all instances of the class that has the property the values of the property are members of the class.

someValuesForm This property specifies the existential quantifier. Specifies that for all instances of the class that has the property at least one of the values of the property are members of the class.

cardinality This property specifies the number of instances in the relation. It is possible to specify a minimum number, a maximum or the exact number of instances.

Example of property restriction:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#madeFromGrape"/>
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinalit
</owl:Restriction>
```

4.2.5 OWL DL

The constructs provided by OWL DL are more powerful and more complex. This extension contains elements of set operations like intersection (intersectionOf), union (unionOf) and complements (complementOf), enumerated classes (one of) or disjoints classes (disjointWith).

4.2.6 OWL Full

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a `owl:DatatypeProperty` can be marked as an `owl:InverseFunctionalProperty`. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.[33]

4.2.7 Problems in OWL

As indicated above one intelligent agent can reason more things in RDF(S) and then obtain more answers than an OWL agent. That implies that you are using the other layers basis to make the new layer, ontology layer extends the RDF layer. Then is possible also that some problems will be extended. Classes with the underlying principles of RDF(S) resulting paradoxes in a same syntax and extended semantics layering of OWL on top of RDF(S) [28].

We have seen that OWL offers many features for modelling a domain, providing classes, relationships, properties or it is also possible to apply restrictions to the elements previously created. It is possible to specify these restrictions with first order predicates that will provide more elements in order to allow the information agents automatic reasoning. An example of most of the elements described in this sections can be found in [33].

4.2.8 Ontology Mapping

We said that one of the principles for designing ontologies should be to share this ontology. The ontology must be public in order to be able to reuse its properties and names. If we share our ontology we will obtain better efficiency from our intelligent agents when they search for information. We can take advantage from the existing ontologies defining some equivalences or relations between them. OWL (DL and Full) provides some constructs in order to achieve this objective.

equivalentClass, equivalentProperty These properties indicate that two classes have equivalent classes or two classes have equivalent properties. When using these constraints you should be careful to not make contradictions in the ontologies. These two properties are within OWL DL, for OWL Full there is another more powerful property (sameAs).

sameAs This property allows us to specify that two classes are identical and causes both arguments of the instruction to be interpreted as individuals. If you use that instruction in you ontology it will be classified like an OWL Full ontology.

differentFrom, AllDifferent This restriction provides the effect of be different from a specified resource or from all resources. It is the opposite effect of sameAs.

In this section we have seen most of the OWL Lite constructs and none from the OWL DL or OWL Full. As indicated above we have to choose between one of these three OWL frameworks for building our ontology. Maybe will be enough for us just use the elements shown in the last pages, if it is not the case, now OWL DL will be described.

4.3 Overview of Topic Maps

XML Topic Maps (XTM) is an ISO standard for describing knowledge structures and associating them with information resources. Is the International Organization for Standardization technology for contributing to the next generation of Web, the Semantic Web.

An example of Topic Maps is the Table of Contents in the beginning of this report. It is possible to see some subjects, the names on the left of the TOC that represent the different sections in this document. In the same table of contents is possible to see the page number where the beginning of the section is. If we translate this example into a topic maps representation we will get that topic will be "Introduction" and the page number will be the occurrence, that is, the place where the Introduction is.

Another good example is given in [29]. Imagine one Shakespeare work, lets say "Hamlet", and we know that "Hamlet" is referenced in lots of places. The topic will be "Hamlet" and the occurrence will be for example "www.hamlet.com" or another URI that references "Hamlet".

Also in the Topic Maps representation we can represent associations. One association can be "Hamlet" was written by Shakespeare, and then, representing this association in Topic Maps we will get the next obvious association Shakespeare wrote "Hamlet". The Topic Maps Model is detailed in 4.4[2] and we can define a Topic Map as:

- **Topic** is a representation of one real concept
- **Occurrences** are the links to the Topic, that is, where the Topic is referenced
- **Association** between Topics (written by)
- **Topic Map** is the collection of these three concepts

[2]http://www.topicmaps.org/xtm/index.html

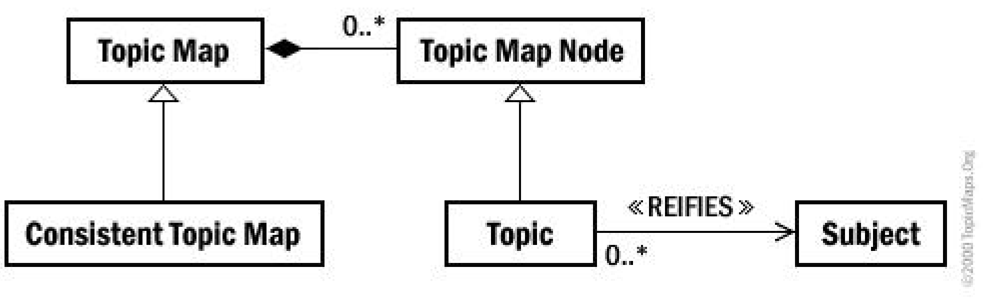


Figure 4.4: Topic Map Model

4.3.1 Main Concepts in Topic Maps

Topic A topic represents one concept abstracted from the reality. A Topic can be "Hamlet", "car" or "Introduction". More formally a topic is the reification (act of creating a topic from one subject) and reification allows assigning topic characteristics (topic name, topic occurrence or role) to that topic. The topic characteristics are only valid within a scope (specifies the validity of the topic characteristics).

Subject subject (identified by a subject indicator) is something about we can reference properties or relations. In a topic map each subject is represented by one topic and more than one topic topic may reify the same subject.

Topic Type A topic type represents the relation between the main class and its instances, the instances will be the topics and the main class the topic type.

Occurrences Occurrences are the relevant cases where we can find the topic. It is a link to the topic. In order to be a valid occurrence it should have one of these two properties:

1. Addressable by reference using URI
2. capable of being placed inline as character data

Occurrence Type An occurrence type represents the main class of an occurrence, the occurrence an instance of Occurrence Type.

Association An association represents a relationship between two or more topics and each topic plays a role (nature of the topic in this association) in the association. The roles are among the characteristics that can be assigned to the topic. There is no directionally inherent in an association, if you associate the topic "Hamlet" with the topic "Shakespeare" you also get the relation "Shakespeare" with "Hamlet".

Association Type The association type represents the main class of association, therefore association is the instance of association type.

Topic Map A Topic Map represents a group of topics with associations between these topics and inside valid scopes. The Topic Map may exist in two forms:

1. A serialized interchange format (e.g. as a topic map document expressed in XTM or some other syntax), or
2. An application-internal form

The purpose of a topic map is to convey knowledge about resources through a superimposed layer, or map, of the resources. A topic map captures the subjects of which resources speak, and the relationships between resources, in a way that is implementation-independent. [29]. A Topic Map Node is an object which contains the internal representation of one topic (with topic, association and scope).

The Topic Map Class Hierarchy is indicated in the Figure4.5 [3]:

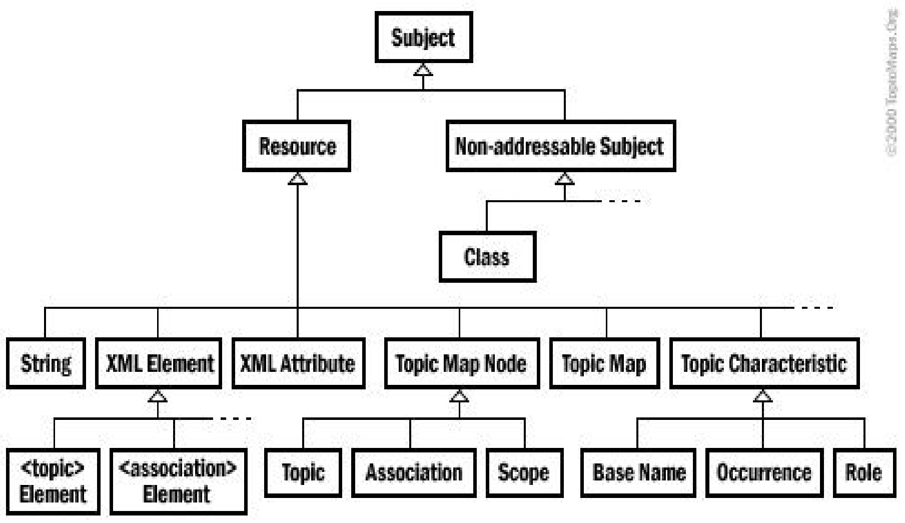


Figure 4.5: Topic Maps Class Hierarchy

4.3.2 Advanced Concepts

Published Subjects The same subject can be represented in more than one topic. We can represent "Spain", "Norway" or we can represent these countries as "España" or "Norge". It is necessary to establish a common identity to share these topics, because if we want to share our knowledge our subjects should be identified for agents that are looking for these subject but with different names. In order to solve that question exist the published subjects. A published subject indicator is any resource that has been published in order to provide a positive, unambiguous indication of the identity of a subject for the purpose of facilitating topic map interchange and merge ability.

[3]http://www.topicmaps.org/xtm/index.html

Facets Facets are the way of Topic Maps to provide metadata to the information resources. A facet provides a mechanism to assign pairs of properties and values of resources. Facet is simply a property with values.

4.3.3 Merging

When two Topic Maps are merged the following situations can happen:

1. When two topic maps are merged, any topics that the application, by whatever means, determines to have the same subject are merged, and any duplicate associations are removed.

2. When two topics are merged, the result is a single topic whose characteristics are the union of the characteristics of the original topics, with duplicates removed.

When two topics are merged exist the possibility that both Topic Maps have the same subject. That happens if:

1. they have one or more subject indicators in common,

2. they reify the same addressable subject, or

3. they have the same base name in the same scope.

4.3.4 XTM Specification

In this section we are going to describe the main constructs for creating an ontology with Topic Maps. These are the principal elements for creating a Topic Map ontology proposed by [29], and more specific elements are detailed in the official specification.

`topicRef` The `topicRef` element specifies an URI reference to a topic. The target of a `topicRef` link must resolve to a topic element child of a `topicMap` document that conforms to this XTM specification. The target topic need not be in the document entity of origin. This element assign a URI reference to a the topic specified topic.

```
<topicRef xlink:href="#thisTopic"/>
```

`scope` The `scope` element specifies the context where the topic characteristics are valid for one or more `topicRef`, `resourceRef` or subjectIndicator. The `scope` element consists of one or more `topicRef`, `resourceRef`, or `subjectIndicatorRef` elements. The union of the subjects corresponding to these elements specifies the context in which the assignment of the topic characteristic is considered to be valid.

```
<scope>
    <topicRef xlink:href="#comedy"/>
    <topicRef xlink:href="#drama"/>
</scope>
```

`instanceOf` the `instanceOf` element specifies an instance of a `topicRef` or `subjectIndicator`.

```
<topic id="Godfather">
<instanceOf>
  <subjectIndicatorRef
      xlink:href="http://www.mybooks.org/puzo.html"/>
</instanceOf>
</topic>
```

`topicMap` The `topicMap`] element specifies the parent of all topic, association, and `mergeMap` elements in the topic map document. `topicMap` element is the root element from which topic map syntactical recognition is performed [29].

```
<topicMap>
  <!-- topics, associations, and merge map directives go here -->
</topicMap>
```

`topic` The `topic` element specifies the name and occurrence characteristics of a single topic. It has a single unique identifier, and the ability to state the class(es) of which it is an instance and the identity of the subject that it reifies. [29].

```
<topic id="Godfather">
<instanceOf>
  <topicRef xlink:href="#book"/>
</instanceOf>
<!-- base names and occurrences go here -->
</topic>
```

`subjectIdentity` The `subjectIdentity` element specifies the subject that is reified by a topic.

`association` the `association` association element creates an association between two or more topics.

```
    <topic id="will-wrote-hamlet-topic">
    <subjectIdentity>
      <subjectIndicatorRef xlink:href="#will-wrote-hamlet"/>
    </subjectIdentity>
    <baseName>
      <baseNameString>Shakespeare’s authorship of
        Hamlet</baseNameString>
    </baseName>
    <!-- occurrences may go here -->
  </topic>
```

`occurrence` The `occurrence` element specifies where is the information resource related to the topic indicated.

```
<topic id="hamlet">
<occurrence>
  <instanceOf>
    <topicRef xlink:href="#date-of-composition"/>
  </instanceOf>
  <resourceData>1600-01</resourceData>
</occurrence>
</topic>
```

`mergeMap` The `mergeMap` element specifies references to an external Topic Map using an URI.

```
<mergeMap xlink:href="http://www.mybooks.org/puzo.xtm">
    <topicRef xlink:href="#mafia"/>
</mergeMap>
```

4.4 Overview of OWL-S

OWL-S is an ontology for describing Web Services based in the Semantic Web Language OWL. A Web service description can be found in Appendix A. OWL-S is build with OWL for creating an ontology that enables the developer to create Semantic Web Services with the requirements and characteristics mentioned in previous chapters. OWL-S is the last release of the DAML-S ontology built with the Ontology language DAML+OIL.

With OWL-S users and software agents should be able to discover, invoke, compose and monitor Web resources offering particular services and having particular properties. [21]

4.4.1 The Ontology for Web Services

OWL-S is located at the application level, just up WSDL, for solving the lacks of this language. As it is explained in Chapter A WSDL has two layers, one abstract layer and another "real" section. At the abstract level WSDL describes the messages that a Web Service sends and receives. These messages can be used to characterize the service but the expressiveness of WSDL is not enough powerful. It is possible to use OWL-S for solving this problem. The essential knowledge about a service that an information agent should know is the same as the answers to these questions:

- What does the service requires of the users or other agents, and provide for them? The answer is given in the ServiceProfile section.
- How does it work? The answer is given in the ServiceModel.
- How is it used? The answer is given in the ServiceGrounding.

In the Figure 4.6 is possible to see the architecture of OWL-S[4]. In this picture it is shown the main modules of the ontology for Web Services. These elements are described in the next sections.

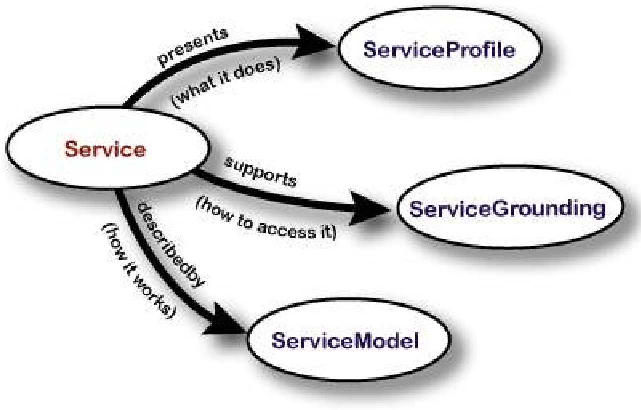


Figure 4.6: The General Process of Engaging a Web Service

The `Service` class depicted in Figure 4.6 is the main class organizer in the OWL-S ontology. This class is provided by the `Resource` class and any of the services, ServiceProfile, ServiceGrounding and ServiceModel will be a Service. The service implements the cardinality restrictions of a service can be described at most by one service model and one grounding must be associated to only one service.

4.4.2 The Service Profile

The service profile provides the mechanisms to describe the participants in a Service: the service requester and the service provider. The Profile allows the user

[4]http://www.w3.org/Submission/OWL-S/

to create definitions and to add semantic information to the service requester needs. The service requester can be a user asking for a determinate service and the service provider can be a seller, the one who provides any service the customer.

The main objective of the Service Profile is to describe the characteristics of the process and what does the process really need. This can also be done by using OWL subclassing. The ServiceProfile provides a representation based on three profile definitions:

- The provider of the service information consists of contact information referred to the entity that provides the service.
- Functional description of the service expressed in term of the transformations done by the service. The functional description of the service specifies the inputs and outputs required in the service. Also describes the preconditions and post conditions of the service.
- Descriptions of the host properties. These descriptions are used to describe the features of the service. Specifies the category of a given service (buy books), the quality rating of the service and a list of parameters that can contain any type of information.

Once the Service Profile is located it starts to be useless. This is because we do not need anything more from it, now the Process Model will be used. But there are also some more questions about the service profile. The service profile specifies a certain type of requirements of the service and the Process Model specifies what the service really needs. These two specifications can be completely different and it can generate some inconsistencies. This situation can happen when a service tries to execute something and the parameters needed are not the specified initially or the result of the service is not the expected result. This is because OWL-S does not specify any constraint between Profile and Process Model and both play different roles during the transaction between Web Services. Also, the Profile allows the user to define several functionalities of the same service. This allows other services to select among different services.

4.4.3 Profile Properties

The OWL Services Coalition has divided the properties of the ProfileService in four groups. These groups classify the properties in properties which link the ServiceProfile class with the Service class and Process Model class; the properties which describe the contact information and the Description of the profile; the properties which describe functional representation in function of the Input, Output, Precondition and Effects (IOPEs); and the properties which describe the attributes of the Profile.

1. The class ServiceProfile provides a superclass of every type of high-level description of the service. It mandates the basic information to link any instance of profile with an instance of service.

`presents` Describes the relation between an instance of a service and an instance of a profile which describes it.

`presentedBy` Inverse of `presents`.

2. The following properties describe the contact information and the Description of the profile. A profile may have at most one service name and text description, but as many items of contact information as the provider wants to offer.

   `serviceName` Represents the name of the service offered.

   `textDescription` : Provides a short description of the service.

   `contactInformation` : Provides information about the human responsible for the service.

3. The following properties describe the parameters needed by the service, the IOPEs.

   `hasParameter` Describes some parameters different from Inputs and Outputs.

   `hasInput` Describes the input of the process.

   `hasOutput` Describes the output of the process.

   `hasPrecondition` Describes the preconditions of the process.

   `hasResult` Describes the expected result of the process.

4. The following properties describe some additional attributes to guarantee the quality of the service.

   `serviceParameterName` Parameter name (a literal or an URI).

   `sParameter` Value of the parameter in an OWL ontology.

4.4.4 Modeling Services as Processes

A Service can be viewed as a process. OWL-S provides the class Process that is a subclass of ServiceModel. A process is not a program to be executed. It is a specification of the ways a client may interact with a service. An atomic process is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. A composite process is one that maintains some state; each message the client sends advances it through the process[21].

The processes can have any number of inputs (representing the information needed for executing the process), any number of preconditions and any number of effects. The effects can be divided in two groups: effects that generate information as output or effects that generate a change in the world state. Input and outputs are subclasses of the general class Parameter and the preconditions are modeled as logical formulas. A extended description of the elements of a process is shown in the next lines.

- **Participants**: A process requires two participants, `TheClient` who will invoke the process and `TheServer`, who provides the service. If there are others can be listed by using the property `hasParticipant`.
- **Inputs and Outputs**: Inputs and outputs specify the data transformation produced by the process. The inputs specifies the data required for the process and the outputs are the result. Inputs can come from the user or from another processes. Both have their properties, `hasInput` and `ConditionalOutputs`, subclasses of `hasParameter`.
- **Preconditions and Results**: It is the mandatory condition to execute a process and the results are the changes that may perform the process.
- **Conditioning outputs and effects**: OWL-S does no assume that outputs and effects are the same for every execution of the process. OWL-S provides the classes ConditionalOutput and ConditionalEffect to associate conditions to the outputs and the effects.

AtomicProcess

The atomic processes are directly invocable and have no subprocesses. Are executed in a single step from the perspective of the service requester. Each atomic process requires a grounding service.

SimpleProcess

Simple processes are not invocable and are not associated with a grounding but they have single-step executions. Simple processes are used as elements of abstraction. They can provide a view of some atomic processes or a simplified representation of some composite processes. The simple process is `realizedBy` one atomic process or the simple process `expandsTo` a composite process.

CompositeProcess

Composite processes are decomposable into other (non-composite or composite) processes. These processes contain several flow control units similar to the traditional programming languages. As an example in the next lines are shown the most common and only indicated the others. A CompositeProcess must have a composedOf property by which is indicated the control structure of the composite, using a ControlConstruct.

```
<owl:ObjectProperty rdf:ID="composedOf">
  <rdfs:domain rdf:resource="#CompositeProcess"/>
  <rdfs:range rdf:resource="#ControlConstruct"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="ControlConstruct">
</owl:Class>
```

Each control construct, in turn, is associated with an additional property called components to indicate the nested control constructs from which it is composed, and, in some cases, their ordering.

```
<owl:ObjectProperty rdf:ID="components">
 <rdfs:domain rdf:resource="#ControlConstruct"/>
</owl:ObjectProperty>
```

The OWL-S control constructs are: Sequence, Split, Split + Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until.

Dataflow and parameter Bindings

Like in all processes sometimes a process needs to access to a parameter which have to be collected from the output of another process. This is the typical dataflow management. In a common programming language this situation is managed by using different variables as function arguments but in OWL this is not possible. Using OWL you can only define the inputs and outputs of processes as properties with range restrictions representing the classes of allowed values, independent of any context [21].

For solving this question we adopt the convention that the source of a datum is identified when the user of the datum is declared. If step 1 feeds step 3, we specify this fact in the description of step 3 rather than the description of step 1. We call this a consumer-pull convention, as opposed to a producer-push alternative. We implement this convention by providing a notation for arbitrary terms as the values of input or output parameters of a process step, plus a notation for subterms denoting the output or input parameters of prior process steps.

4.4.5 Grounding Service

The Grounding Service is the service for specifying the details of how to access the service, a mapping from the abstract to the concrete description of this service. The ServiceProfile and the ServiceModel are abstract representations, the concrete representation is found in ServiceGrounding.

OWL-S does not provide any mechanism for defining messages between services, only the input and output properties of the processes. The GroundingService main objective is to convert the processes' inputs and outputs to concrete messages, and this is done by using WSDL. The relation between OWL-S grounding service and WSDL is shown in the Figure 4.7 [5]. With this complementary use of OWL-S and WSDL it is combined the expressiveness of the OWL ontology for services for defining processes and the reuse of WSDL.

[5]http://www.daml.org/services/owl-s/1.1/overview/

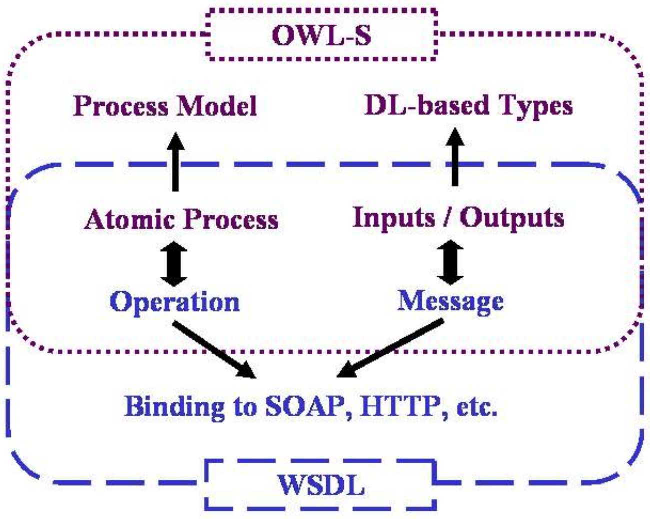


Figure 4.7: Relation Between OWL-S and WSDL

An OWL-S/WSDL grounding is based upon the following three correspondences between OWL-S and WSDL.

1. An OWL-S atomic process corresponds to a WSDL (1.1) operation. Different types of operations are related to OWL-S processes as follows:
    - An atomic process with both inputs and outputs corresponds to a WSDL request-response operation.
    - An atomic process with inputs, but no outputs, corresponds to a WSDL one-way operation.
    - An atomic process with outputs, but no inputs, corresponds to a WSDL notification operation.
    - A composite process with both outputs and inputs, and with the sending of outputs specified as coming before the reception of inputs, corresponds to WSDL's solicit-response operation.
2. The set of inputs and the set of outputs of an OWL-S atomic process each correspond to WSDL's concept of message. More precisely, OWL-S inputs correspond to the parts of an input message of a WSDL operation, and OWL-S outputs correspond to the parts of an output message of a WSDL operation.
3. The types (OWL classes) of the inputs and outputs of an OWL-S atomic process correspond to WSDL's extensible notion of abstract type (and, as such, may be used in WSDL specifications of message parts).

To construct an OWL-S/WSDL grounding one must first identify, in WSDL, the messages and operations by which an atomic process may be accessed, and then specify correspondences (1) - (3).

4.5 Other Languages

The Ontology Inference Layer OIL is a proposal for a web-based representation and inference layer for ontologies, which combines the widely used modelling primitives from frame-based languages with the formal semantics and reasoning services provided by description logics. It is compatible with RDF Schema (RDF(S)), and includes a precise semantics for describing term meanings (and thus also for describing implied information).

OIL presents a layered approach to a standard ontology language. Each additional layer adds functionality and complexity to the previous layer. This is done such that agents (humans or machines) who can only process a lower layer can still partially understand ontologies that are expressed in any of the higher layers.[25]

DAML is a language created by DARPA as an ontology and inference langauge based upon RDF. DAML takes RDF Schema a step further, by giving us more in depth properties and classes. DAML allows one to be even more expressive than with RDF Schema, and brings us back on track with our Semantic Web discussion by providing some simple terms for creating inferences.[26]

4.6 Concluding Remarks

In this section we have seen three different ontology languages and an extension for one of these languages. Every language has it purpose and fits for some determinate requirements and are more suitable for solving determinate problems. RDF(S) fits better in simples cases, OWL is better to develop business ontologies. Topic Maps fits in the middle of both languages. With the OWL-S extension we have seen a very powerful ontology language that offers tools not only to describe data, also to describe web services and make accessible our services to the information agents that are populating our networks.

Chapter 5

Comparative Evaluation of Ontologies

In this chapter, the analysis of the three Web ontology languages is described. We based the analysis in the framework specified in Appendix B, a Semiotic Quality Framework and the adaptations indicated also in this Appendix B.

5.1 RDF(S) Evaluation

In this section RDF(S) will be evaluated using the Semiotic Quality Framework described in Appendix B. The original framework for evaluating languages is [19] and modification for the Domain Appropriateness is [35].

5.1.1 Domain Appropriateness

In this subsection we analyze if the language has enough expressiveness power for representing the domain of our problem. RDF provides three main elements, Subject or resource, a property of this resource and the object or value of the property. With this main element within RDF you can specify classes, resources, datatypes, range of the properties, subclasses and other similar constructs. In order to analyze this appropriateness these perspectives will be used:

- **Structural Perspective**: RDF(S) provides the basic constructs needed for representing the schema of the Invenio's database. In the schema we need classes, attributes and relationships. Basic relationships can be added to the classes using slots with an instance of the related class as datatype. We can also specify inverse slots, the cardinality and some small restrictions on this slot. These are the basic constructs needed for modelling the database, but in order to obtain more expressiveness in the corresponding ontology more constructs are needed, like constraints or complex relationships.

- **Functional Perspective**: RDF(S) is a description language and since for describing data we do not need any kind of process, activities or transformations therefore RDF(S) do not provides any kind of mechanism for that. Moreover, this perspective represents the states and the transitions between them. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Behavioral Perspective**: RDF(S) does not model the behavior of any model, RDF(S) is used to give meaning to static data therefore RDF(S) does not provides constructs for specifying states or transitions between the states. Moreover, this perspective represents the states and the transitions between them. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Rule Perspective**: Rule perspective is about the rules that we can create with the language. RDF(S) does not provide these constructs, either any kind of predicates for inferring rules. These constructs are useful in our case. We have to access the data from one database in order to get quality answers from the queries and if we specify certain kind of rules like "the full name of one person is the composition of the first name and last name" we will get better quality answers from our queries. This kind of rules can not be represented within RDF(S).

- **Object Perspective**: RDF(S) offers some constructs related with objects. Objects are classes with attributes and processes and RDF(S) offers mechanisms in order to create classes and attributes for these classes but does not provide processes facilities. In our case RDF(S) is good enough in this perspective because in our database we do not have any process to implement but we have classes and attributes and we can represent it within RDF(S).

- **Communication Perspective**: RDF(S) does not offer any construct for supporting the communication perspective. For supporting it should offer the possibility for representing the illocutionary context (hearer, speaker, time location and circumstances). In the description of the data we will know nothing about the hearer (who is using the data), the speaker (the server) or other elements of the illocutionary context. For representing the illocutionary force it is needed to represent the reasons and the goals of the data. With RDF(S) we can not describe which are the goals or the reasons of our data because it will be different in different contexts. We can not have one representation for each execution of a query.

- **Actor and Role Perspective**: RDF(S) does not cover this perspective, also in our case we do not need it. We can no specify any kind of actor or role within RDF(S). In the Invenio case this is not a problem, we do not have to represent any actor or role in the ontology, just model it.

5.1.2 Participant Knowledge Appropriateness

All the statements in the language models of the languages used by the participants are part of the explicit knowledge of this participant. Underlying basis should correspond to the way individuals perceive the reality. RDF(S) with

Protégé gives enough facilities in order to satisfy the condition for this appropriateness. The database has been designed in RDF(S) by reproducing the same schema of the database. The database model and the RDF(S) model have the same classes and the same relationships. Concepts like class, relate two classes or attributes are easily created within RDF(S) and Protégé.

The external representation of the participant knowledge appropriateness represents how intuitive is the language in order to express the situation that we are modeling. RDF(S) by itself does not offer any facility to satisfy this property but combined with Protégé it is possible to view the diagrams and the hierarchy of the model. Even this it is necessary a basic knowledge about RDF(S).

In our case this appropriateness it is not important. The solution for Invenio will be an ontology to access the database. It is not required any RDF(S) knowledge by the end user because it only will search at the database.

5.1.3 Knowledge Externalizability Appropriateness

This appropriateness indicates how relevant knowledge may be represented in the modeling language. Also, the knowledge of the developer about the language and his skill designing models will be key concepts in this appropriateness. Within RDF(S) it is possible to model classes, attributes and the common constructs. Within RDF(S) we have a great freedom for modelling everything, for example OWL is defined using the RDF(S) but it does not offers the constructs needed for modelling a complex domain. For example, RDF(S) does not offer the possibility of create first order logic axioms, hence we can not create logical rules for constraining determinate structures.

In the Invenio case it is a problem but a problem with limits since the solution is limited to express the data from the databases using the schema provided or extracted. The solution will be based in represent these models, therefore we do not need maximum expressiveness from our language. We do not need maximum expressiveness but more expressiveness than RDF(S) is needed. We will need to create constraints in order to define special kinds of relations like "neighbor" and RDF(S) does not provide the tools needed for that.

5.1.4 Comprehensibility Appropriateness

This appropriateness describes that all the possible statements of the language are understood by the participants, in other words, the participants know everything of the language. This use to be difficult in any language and RDF(S) is not an exception but it is not difficult read RDF(S) formatted data.

The underlying basis specifies that the phenomena of the language should be easily distinguished from each other. RDF(S) classes are identified within the code by `rdfs:class`, resources are identified with `rdfs:Resource`, instances with `rdfs:type` or properties with `rdfs:Property`, therefore the main elements in RDF(S) are easily distinguished from each other. The number of the phenomena is reasonable (the key concepts is a triples composed by resource, property and the value of the property) and RDF(S) is flexible in level of detail but you have to specify all with the same constructs. We can separate the RDF(S) concepts by dividing in different ontologies or resources our problem and reference from our main ontology these concepts. Also RDF(S) provides expressive economy for expressing just the things we want.

The external representation indicates that the symbol discrimination should be easy. RDF(S) offers an easy way to see the differences between the symbols of the languages providing different constructs for each symbol. The external language is consistent, every symbol represents one phenomenon, but different combinations of resources will produce inconsistencies in our model.

In RDF(S) it is possible to emphasize the important phenomena by adding comments or the adequate properties and navigation it is not difficult. Protégé allow us to navigate easily among the structure in RDF(S) created. The structure of the language is consistent enough. The language provides enough elements in this appropriateness in order to satisfy the Invenio case requirements because in this case it is not necessary a high level of comprehensibility of the participants. The participants does not need to know every element of the language because we are in a specific domain case study and all elements of the language are not needed.

5.1.5 Technical Actor Appropriateness

For the technical actors is important that the language lends it self to automatic reasoning. RDF(S) is a language for describing data and the main motivation for describing this data is that a technical actor, for example an information agent, be allowed to access the data and process it in an intelligent way. RDF(S) provides basic information for automatically reasoning, offers mainly descriptions but not logical rules which can be used by information agents for reasoning properly about the data.

The Invenio case need to achieve the maximum level of Technical Actor Appropriateness. The main goal of this case is to provide the maximum amount of information to the information agents in order to facilitate their automatic reasoning and provide better answers to the queries of the Directory Service. Therefore RDF(S) does not provide all the elements needed.

5.2 OWL Evaluation

In this section the Ontology Web Language will be evaluated using the Semiotic Quality Framework. For the analysis we will use the Framework proposed by [19] and modification of the Domain Appropriateness proposed by [35].

5.2.1 Domain appropriateness

In this subsection we analyze if the language has enough expressiveness power in order to represent the domain of our problem. OWL provides to the user classes, datatype properties which represents the attributes of a determinate class, object properties which represents the relationships between two classes, individuals which represent the instances of the classes previously defined, descriptions for all the elements described, mapping between ontologies. These are the facilities that OWL offers for design:

- **Structural Perspective**: OWL provides the necessary constructs for creating the structure we need. In our relational model extracted from the databases and presented in Chapter 8 we mainly need classes and relationships between these classes. OWL provides these constructs with the constructs for creating classes and object and datatype properties. Also, we can specify easily with Protégé inverse relationships (which initially we do not need) or disjoint classes (which we need it for giving more meaning and make easier the task of the agent that will access to the data).

- **Functional Perspective**: OWL is a description language and since for describing data we do not need any kind of process, activities or transformations, OWL does not provide any kind of mechanism to do that. Moreover, the functional perspective is referred to the process, activities and transformations. In our case, a static database which does not implement any kind of process, activities or transformations because is used for consulting information about people, not for transforming it. We do not need to analyze this perspective.

- **Behavioral Perspective**: OWL does not model the behavior of any model, OWL is used to give meaning to static data. Therefore OWL does not provide constructs for specifying states or transitions between the states. Moreover, this perspective represents the states and the transitions between them. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Rule Perspective**: Rule perspective is about the rules that we can create with the language. OWL provides constructs for creating first order predicates in order to achieve the goal of put restrictions into the relationships. Examples of these constructs are the existential quantifier, universal quantifier, cardinality, logical and, logical and so forth. These constructs are useful in our case. We have to access the data from one database in order to get quality answers from the queries and if we specify certain kind of rules like "the full name of one person is the composition of the first name and last name" we will get better quality answers from our queries. This kind of rules can be represented within OWL.

- **Object Perspective**: OWL offers some constructs related with objects. Objects are classes with attributes and processes and OWL offers mechanisms for creating classes and attributes for these classes (datatype properties) but does not provide processes facilities. In our case OWL is good enough in this perspective because in our database we do not have any process to implement but we have classes and attributes and we can represent it within OWL.

- **Communication Perspective**: OWL does not offer any construct for supporting the communication perspective. For supporting it should offer the possibility for representing the illocutionary context (hearer, speaker, time location and circumstances). In the description of the data we will know nothing about the hearer (who is using the data), the speaker (the server) or other elements of the illocutionary context. For representing the illocutionary force it is needed to represent the reasons and the goals of the data. With OWL we can not describe which are the goals or the reasons of our data because it will be different in different contexts. We can not have one representation for each execution of a query.

- **Actor and Role Perspective**: OWL does not cover this perspective, also in our case we do not need it. The most similar to an actor perspective can be the "client" class or something similar. We can add roles to this "actor" by adding relationships and restrictions to these relationships, like adding the "role" neighbour with restrictions in determinate relations. But naturally OWL does not cover this perspective. In the Invenio case this is not a problem, we do not have to represent any actor or role in the ontology, just model it.

5.2.2 Participant Knowledge Appropriateness

All the statements in the language models of the languages used by the participants are part of the explicit knowledge of this participant. Underlying basis should correspond to the way individuals perceive the reality. OWL with Protégé gives enough facilities in order to satisfy the condition for this appropriateness. The database has been designed in OWL by reproducing the same schema of the database. The database model and the OWL model have the same classes and the same relationships. Concepts like class or attribute are easily created within OWL and Protégé. Maybe the most difficult is to understand that the object properties are the relationships (it takes few minutes). Once you understand these concepts and create correspondent classes and relationships are easy to figure how to extend these elements with more constraints. Protégé allows the user to create easily restrictions with its editor, which enables the user to navigate for all the classes and relations in order to create the restrictions with precision.

The external representation of the participant knowledge appropriateness represents how intuitive is the language for expressing the situation that we are modeling. OWL for itself does not offer any facility to satisfy this property but combined with Protégé it is possible to view the diagrams and the hierarchy of the model. Even this it is necessary a basic knowledge about OWL.

In our case this appropriateness it is not important. The solution for Invenio will be an ontology to access the database. It is not required any OWL knowledge by the end user because it only will search at the database.

5.2.3 Knowledge Externalizability Appropriateness

This appropriateness indicates how relevant knowledge may be represented in the modeling language. Also, the knowledge of the developer about the language and his skill designing models will be key concepts in this appropriateness. Within OWL it is possible to model classes, attributes and the common constructs. It is obvious we can not model everything within OWL. OWL is designed for describing data and it has three different layers of complexity, OWL Lite, OWL DL and OWL Full indicating that are things impossible to model within OWL Lite. OWL Full offers more constructs for modelling but with no computational guarantees[33]. This sentence implies that OWL can not model everything because if we do not get any result from our model the model is wrong.

In the Invenio case it is a problem but a problem with limits since the solution is limited to express the data from the databases using the schema provided or extracted. The solution will be based in represent these models, therefore we do not need maximum expressiveness from our language. The constructs provided for OWL are very good and can express the model extracted from the database easily.

5.2.4 Comprehensibility Appropriateness

This appropriateness describes that all the possible statements of the language are understood by the participants, in other words, the participants know everything of the language. This use to be difficult in any language and OWL is not an exception but it is not difficult read OWL formatted data.

The underlying basis specifies that the phenomena of the language should be easily distinguished from each other. OWL classes are identified within the code by `owl:class`, properties are identified with `owl:ObjectProperty` and restrictions with `owl:Restriction`, therefore the main elements in OWL are easily distinguished from each other. The number of the phenomena is reasonable (the key concepts are four) and OWL is flexible in level of detail since provides three versions of OWL and the use of the phenomena is uniform. Also OWL provides expressive economy to express just the things we want using one of the three sublanguages.

The external representation indicates that the symbol discrimination should be easy. OWL offers an easy way to see the differences between the symbols of the languages providing different constructs for each symbol. The external language is consistent, every symbol represents one phenomenon, but different combinations of properties will produce inconsistencies in our model.

In OWL it is possible to emphasize the important phenomena by adding comments or the adequate properties and navigation it is not difficult. The structure of the language is consistent enough. The language provides enough elements in this appropriateness in order to satisfy the Invenio case requirements because in this case it is not necessary a high level of comprehensibility of the participants. The participants does not need to know every element of the language because we are in a specific domain case study and all elements of the language are not needed.

5.2.5 Technical Actor Appropriateness

For the technical actors it is important that the language lends it self to automatic reasoning. OWL is a language for describing data and the main motivation to describe this data is that a technical actor, for example an information agent, be allowed to access the data and process it in an intelligent way. OWL achieves its main goal that is the technical actor appropriateness by allowing the information agents to reason about what is described with OWL.

The Invenio case need to achieve the maximum level of Technical Actor Appropriateness. The main goal of this case is to provide the maximum amount of information to the information agents in order to facilitate their automatic reasoning and provide better answers to the queries of the Directory Service. Therefore OWL provides enough elements to satisfy this requirement.

5.3 Topic Maps Evaluation

In this section Topic Maps will be evaluated using the Semiotic Quality Framework described in Appendix B. The original framework for evaluating languages is [19] and modification for the Domain Appropriateness is [35].

5.3.1 Domain Appropriateness

In this subsection we analyze if the language has enough expressiveness power for representing the domain of our problem. Topic Maps contains several elements, Topic Map, Topic, Association and so forth that will allow us to create complex models specifying classes, attributes or relationships easily. In this section we are going to see the quality of these elements within Topic Maps ontology language. In order to analyze this appropriateness these perspectives are studied:

- **Structural Perspective**: Topic Maps provides classes (Topic Types), relations between classes (Associations), attributes in Topic Types, Topics which can be viewed as relevant cases of topic types, occurrences (links to topics) or instances of these topics. These elements are enough for representing the databases models in Topic Maps and also offers more elements for specifying better expressiveness.

- **Functional Perspective**: Topic Maps is a language designed for describing data, not to process anything, therefore Topic Maps does not offer any element to manage processes, activities or transformation. The main goal of Topic Maps is to represent information and to make this information available. Moreover, this perspective represents the states and the transitions between them. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Behavioral Perspective**: Topic Maps objective is to give meaning to the data not to model the behaviour of this data. Also the data represented within Topic Maps does not have any transition or state, is statically data, therefore Topic Maps does not provide any mechanism for modelling the behaviour of the data. Moreover, this perspective represents the states and the transitions between them. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Rule Perspective**: Rule perspective is about the rules we can create with the language. Topic Maps does not provide these constructs, either any kind of predicates for inferring rules.

  Instead of offer explicit mechanism for creating rules or define logical predicates, Topic Maps offers a more complex way for defining relations. Within Topic Maps it is possible to define more exactly the elements of one relation, therefore we can obtain better accuracy with the answers of our queries. These constructs are useful in our case. We have to access the data from one database in order to get quality answers from the queries and if we specify certain kind of rules like "the full name of one person is the composition of the first name and last name" we will get better quality answers from our queries. This kind of rules can not be represented within Topic Maps but can be substituted for complex relationships.

- **Object Perspective**: Topic Maps offers some constructs related with objects. Objects are classes with attributes and processes and Topic Maps offers mechanisms for creating classes and attributes for these classes but does not provide processes facilities.In our case Topic Maps is enough good in this perspective because in our database we do not have any process to implement but we have classes and attributes and we can represent it within Topic Maps.

- **Communication Perspective**: Topic Maps does not offer any construct for supporting the communication perspective. For supporting it should offer the possibility for representing the illocutionary context (hearer, speaker, time location and circumstances). In the description of the data we will know nothing about the hearer (who is using the data), the speaker (the server) or other elements of the illocutionary context. For representing the illocutionary force it is needed to represent the reasons and the goals of the data. With Topic Maps we can not describe which are the goals or the reasons of our data because it will be different in different contexts. We can not have one representation for each execution of a query.

- **Actor and Role Perspective**: Topic Maps does not cover this perspective, also in our case we do not need it. We can no specify any kind of actor or role within Topic Maps. In the Invenio case this is not a problem, we do not have to represent any actor or role in the ontology, just model it.

5.3.2 Participant Knowledge Appropriateness

All the statements in the language models of the languages used by the participants are part of the explicit knowledge of this participant. Underlying basis should correspond to the way individuals perceive the reality. Topic Maps with Protégé does not give enough facilities in order to satisfy the condition for this appropriateness. It is necessary to download the Topic Maps Tab from http://www.techquila.com/tmtab/index.html and the latest version is 0.4. This is an early version that should be improved in the next months, and then the quality of this appropriateness will be improved. The database has been designed in Topic Maps by reproducing the same schema of the database. The database model and the Topic Maps model have the same classes and the same relationships. Concepts like class, relate two classes or attributes are easily created within Topic Maps and Protégé if we have a clear view about the main Topic Maps concepts.

The external representation of the participant knowledge appropriateness represents how intuitive is the language for expressing the situation that we are modeling. Topic Maps for it does not offer any facility in order to satisfy this property but combined with Protégé it is possible to view the hierarchy of the model. Even this it is necessary a basic knowledge about Topic Maps.

In our case this appropriateness is not important. The solution for Invenio will be an ontology to access the database. It is not required any Topic Maps knowledge by the end user because it only will search at the database.

5.3.3 Knowledge Externalizability Appropriateness

This appropriateness indicates how relevant knowledge may be represented in the modelling language. Also, the knowledge of the developer about the language and its skill designing models will be key concepts in this appropriateness. Within Topic Maps it is possible to model classes, attributes and the common structures. Also, Topic Maps offers more elements to detail relations and classes.

In the Invenio case we find useful the elements offered by Topic Maps for creating relations, Topics, Topic Types, Occurrences and so forth and these elements will allows us to build a good model from the database also specifying complex relations.

5.3.4 Comprehensibility Appropriateness

This appropriateness describes that all the possible statements of the language are understood by the participants, in other words, the participants know everything of the language. This use to be difficult in any language and Topic Maps is not an exception but it is not difficult read Topic Maps formatted data.

The underlying basis specifies that the phenomena of the language should be easily distinguished from each other. The Topic Maps elements are easily identified from each other due to their specification: `<topicMap> topics, associations...</topicMap>`, `<association> ...</association>`, all elements start with `<name>` and end with `</name>` and inside of each element can be another elements identified with the same method.

The number of the phenomena is reasonable: the main concepts are Topic Map, Topic Type, Topic, Association and Occurrences. Also we can merge different Topic Maps from other specifications. Also Topic Maps provides expressive economy for expressing just the things we want.

The external representation indicates that the symbol discrimination should be easy. Topic Maps offers an easy way to see the differences between the symbols of the languages providing different constructs for each symbol. The external language is consistent, every symbol represents one phenomenon, but different combinations of resources will produce inconsistencies in our model.

In Topic Maps it is possible to emphasize the important phenomena by adding comments or the adequate properties and navigation it is not difficult. Protégé allow us to navigate easily among the structure in Topic Maps created.

The structure of the language is consistent enough. The language provides enough elements in this appropriateness in order to satisfy the Invenio case requirements because in this case it is not necessary a high level of comprehensibility of the participants. The participants does not need to know every element of the language because we are in a specific domain case study and all elements of the language are not needed.

5.3.5 Technical Actor Appropriateness

For the technical actors is important that the language lends it self to automatic reasoning. Topic Maps is a language for describing data and the main motivation for describing this data is that a technical actor, for example an information agent, be allowed to access the data and process it in an intelligent way. Topic Maps provides an efficient way of indexing and formatting data which will allow the technical actors to reason with efficiency about the data.

The Invenio case need to achieve the maximum level of Technical Actor Appropriateness. The main goal of this case is to provide the maximum amount of information to the information agents in order to facilitate their automatic reasoning and provide better answers to the queries of the Directory Service. Therefore Topic Maps provides enough elements to satisfy this requirement.

5.4 OWL-S Analysis

In this section the Ontology Web Language for Services OWL-S will be evaluated using the Semiotic Quality Framework. For the analysis we will use the Framework proposed by [19] and modification of the Domain Appropriateness proposed by [35].

5.4.1 Domain Appropriateness

In this subsection we analyze if the language has enough expressiveness power in order to represent the domain of our problem. OWL-S is an ontology for services created with OWL. Therefore OWL-S provides the same features than OWL and these features are classes, datatype properties which represents the attributes of a determinate class, object properties which represents the relationships between two classes, individuals which represent the instances of the classes previously defined, descriptions for all the elements described, mapping between ontologies. Also, OWL-S adds the extension for managing services and processes. These are the facilities that OWL-S offers for designing:

- **Structural Perspective**: OWL-S provides the same necessary constructs for creating the structure that we need as OWL. In our relational model extracted from the databases and presented in Chapter 8 we mainly need classes and relationships between these classes and now, with OWL-S we can add processes. OWL-S provides these constructs with the constructs for creating classes and object and datatype properties. Also, we can specify easily with Protégé inverse relationships (which initially we do not need) or disjoint classes (which we need it for giving more meaning and make easier the task of the agent that will access to the data) or define processes and their parameters.

- **Functional Perspective**: As we said in the previous evaluation of OWL, OWL is a description language and to describe data we do not need any kind of process, activities or transformations. Therefore OWL does not provide any kind of mechanism for that. Moreover, the functional perspective is referred to the process, activities and transformations. But with OWL-S is provided an extension which allows the user to define services, publish them and access to these services by other services. In our case, a static database which does not implement any kind of process, activities or transformations because is used for consulting information about people, not for transforming it, we do not need to analyze this perspective. But if the owner of the database, for example wants to provide a service for accessing to certain information of the database OWL-S will provide the necessary constructs for accessing this information.

- **Behavioral Perspective**: As we said in the previous evaluation OWL does not model the behavior of any model, OWL is used to give meaning to static data therefore OWL does not provide constructs for specifying states or transitions between the states. Moreover, this perspective represents the states and the transitions between them. But one of the OWL-S motivating task is to provide automatic Web service execution and monitoring but it is still is project, the main motivating task are web service discovery, invocation and composition. In our case, the database has not states or transitions, hence it is not necessary to apply the analysis to this perspective.

- **Rule Perspective**: Rule perspective is about the rules that we can create with the language. OWL provides constructs for creating first order predicates in order to achieve the goal of put restrictions into the relationships therefore OWL-S provides the same constructs as OWL because it is just an ontology based on OWL. Examples of these constructs are the existential quantifier, universal quantifier, cardinality, logical and, logical and so forth. These constructs are useful in our case. We have to access the data from one database in order to get quality answers from the queries and if we specify certain kind of rules like "the full name of one person is the composition of the first name and last name" we will get better quality answers from our queries. This kind of rules can be represented within OWL and OWL-S.

- **Object Perspective**: OWL offers some constructs related with objects and OWL-S extends OWL to provide more object constructs. Objects are classes with attributes and processes and OWL offers mechanisms for creating classes and attributes for these classes (datatype properties) and OWL-S provides the constructs to create processes and define them. In our case OWL is enough good in this perspective because in our database we do not have any process to implement but we have classes and attributes and we can represent it within OWL. If we want to extend the possibilities of the database we can use OWL-S to provide different services accessible from Internet, for example.

- **Communication Perspective**: OWL-S as an extension of OWL has the same problems but in this perspective can solve some of them. With OWL-S does support some of the characteristics of the communication perspective. It offers elements for describing part of the illocutionary context (hearer, speaker, time location and circumstances). With OWL-S we can represent the hearer (who is using the data), the speaker (the server) but not the other elements of the illocutionary context. For representing the illocutionary force it is needed to represent the reasons and the goals of the data. With OWL-S we can describe which are the goals because are the outputs of the services (not always) but we can not describe the reasons of our data because it will be different in different contexts. We can not have one representation for each execution of a query.

- **Actor and Role Perspective**: OWL does not cover this perspective, also in our case we do not need it. OWL-S does not extend this perspective. The most similar to an actor perspective can be the "client" class or something similar. We can add roles to this "actor" by adding relationships and restrictions to these relationships, like adding the "role" neighbor with restrictions in determinate relations. But naturally OWL does not cover this perspective. In the Invenio case this is not a problem, we do not have to represent any actor or role in the ontology, just model it.

5.4.2 Participant Knowledge appropriateness

All the statements in the language models of the languages used by the participants are part of the explicit knowledge of this participant. Underlying basis should correspond to the way individuals perceive the reality. OWL with Protégé gives enough facilities in order to satisfy the condition for this appropriateness and also there is a plug-in for creating Semantic Web Services with OWL-S. The database has been designed in OWL by reproducing the same schema of the database but now there is no service added to the databases because it is not in the scope of the problem. The database model and the OWL model have the same classes and the same relationships. Concepts like class or attribute are easily created within OWL and Protégé. The modeling process in OWL-S and Protégé is the same as with OWL and Protégé. This is due to the OWL-S plug-in because for using the plugging first is mandatory to create an OWL project and later configure Protégé for using the plugging. The model of the database must be created with the OWL plugging first. Then, the pros and cons analyzed before for OWL are present in this analysis of OWL-S. Maybe the most difficult is to understand that the object properties are the relationships (it takes few minutes). Once you understand these concepts and create correspondent classes and relationships are easy to figure how to extend these elements with more constraints. Protégé allows the user to create easily restrictions with its editor, which enables the user to navigate for all the classes and relations in order to create the restrictions with precision.

The external representation of the participant knowledge appropriateness represents how intuitive is the language for expressing the situation that we are modeling. The external representation is the same for OWL-S than for OWL because we use the same tool for modelling and the language is basically the same with a new plug-in. OWL for itself does not offer any facility to satisfy this property but combined with Protégé it is possible to view the diagrams and the hierarchy of the model. Even this it is necessary a basic knowledge about OWL. In our case this appropriateness it is not important. The solution for Invenio will be an ontology to access the database. It is not required any OWL-S knowledge by the end user because it only will search at the database.

5.4.3 Knowledge Externalizability Appropriateness

This appropriateness indicates how relevant knowledge may be represented in the modeling language. Also, the knowledge of the developer about the language and his skills designing models will be key concepts in this appropriateness. Within OWL it is possible to model classes, attributes and with the OWL-S ontology it is possible to model processes. OWL is designed for describing data and it has three different layers of complexity, OWL Lite, OWL DL and OWL Full. With OWL-S we have the same layers and the extra ontology for modelling services. Then, we have the same restrictions as in OWL. OWL Full offers more constructs for modelling but with no computational guarantees [33]. This sentence implies that OWL can not model everything because if we do not get any result from our model the model is wrong. Therefore, OWL-S as an extension of OWL has the same problems.

In the Invenio case it is a problem but a problem with limits since the solution is limited to express the data from the databases using the schema provided or extracted. The solution will be based in represent these models, therefore we do not need maximum expressiveness from our language. The constructs provided for OWL and OWL-S are very good and can express the model extracted from the database easily and also we can add services.

5.4.4 Comprehensibility Appropriateness

This appropriateness describes that all the possible statements of the language are understood by the participants, in other words, the participants know everything of the language. We can apply the analysis of OWL to OWL-S because OWL-S is just a new ontology based on OWL. This use to be difficult in any language and OWL-S is not an exception but it is not difficult read OWL-S formatted data.

The underlying basis specifies that the phenomena of the language should be easily distinguished from each other. OWL-S classes are identified within the code by `owl:class`, properties are identified with `owl:ObjectProperty` and restrictions with `owl:Restriction`, therefore the main elements in OWL-S use the same classes as OWL and they are easily distinguished from each other. The elements of the processes are also easily distinguished, for example the input parameters `process:Input` or the Grounding service `grounding:WsdlAtomicProcessGrounding`. The number of the phenomena is reasonable (the key concepts are four) and OWL is flexible in level of detail since provides three versions of OWL and the use of the phenomena is uniform. Also OWL provides expressive economy to express just the things we want using one of the three sublanguages. As an extension of OWL, OWL-S inherits all its characteristics.

The external representation indicates that the symbol discrimination should be easy. OWL-S offers the same easy way to see the differences between the symbols of the languages than OWL. OWL-S provides different constructs for each symbol. The external language is consistent, every symbol represents one phenomenon, but different combinations of properties will produce inconsistencies in our model.

In OWL-S it is possible to emphasize the important phenomena by adding comments or adequate properties and navigation is not difficult. The structure of the language is consistent enough. The language provides enough elements in this appropriateness in order to satisfy the Invenio case requirements because in this case it is not necessary a high level of comprehensibility of the participants. The participants do not need to know every element of the language because we are in a specific domain case study and all elements of the language are not needed.

5.4.5 Technical Actor Appropriateness

For the technical actors is important that the language lends it self to automatic reasoning. OWL-S is a language for describing processes and OWL is a language for describing data. The main motivation for describing this data is that a technical actor, for example an information agent, be allowed to access the data and process it in an intelligent way.

OWL-S and OWL achieves their main goal that is the technical actor appropriateness by allowing the information agents to reason about what is described with OWL-S and OWL (processes and data). The Invenio case needs to achieve the maximum level of Technical Actor Appropriateness. The main goal of this case is to provide the maximum amount of information to the information agents in order to facilitate their automatic reasoning and provide better answers to the queries of the Directory Service. Therefore OWL-S provides enough elements to satisfy this requirement.

We have seen in the analysis of OWL-S that it is an extension of OWL for processes. It fully covers the Functional, Behavioral and Object perspectives of the Domain Appropriateness and OWL did not cover fully these perspectives, just some cases. Also, OWL-S improves some characteristics of OWL as we can see in the other Appropriateness. It covers better the Participant Knowledge Appropriateness due to the knowledge of processes is easy to understand to one developer and also it is available a plugging of OWL-S for Protégé. The other appropriateness's are not improved or worsen due to OWL-S is just an extension of OWL to fix some of the lacks of a description language, but it still remains some of the problems found before.

5.5 Concluding Remarks

In this section we have evaluated the three ontology languages described in Chapter 4, RDF(S), OWL, OWL-S and Topic Maps. We have presented the advantages and disadvantages of each languages and we have arrived to the conclusion that in their basic conception all of them provides the same constructs. The main difference between them is the purpose of each language. RDF(S) due to that does not provide elements for creating restrictions to the relationships has the lowest expressiveness of the languages analyzed. OWL provides the best expressiveness by adding first order predicates to the restriction and Topic Maps is a mid term between OWL and RDF(S) due to Topic Maps has more elements for specifying relations between classes but does not provide first order predicates like OWL and is more difficult to understand than the other languages.

We have also seen the requirements needed in order to solve the Invenio Case. Combining the results of the analysis of the languages and the requirements needed for representing the databases we have arrived to the conclusion that the most appropriated language is OWL due to that it gets the highest marks for satisfying the case in the final comparison of weights. OWL-S is a good extension of OWL but it is not needed in this case. We the company considers to upgrade the system for offering this service, OWL-S will be the best election.

Chapter 6

Comparison of languages

In this chapter, we are going to compare the three Web ontology languages and the OWL-S extension described in the Chapter 4. In the Chapter 5 we have the analysis of these languages and the problem proposed by Invenio. This chapter will show the comparison of all elements described.

6.1 Comparison of the languages

In order to make the final comparison of the languages a weighted classification has been used. This classification is based in [12] and the weights assigned to the elements of the Semiotic Quality Framework are the following:

Let *CF* be a classification framework such that *CF* has a fixed set Ç of categories ç, where Ç = ç1, ç2, ç3, ç4, ç5 and ç Ç, where ç is a quadruple <id, descriptor, C, cw>, id is the name of the category, descriptor is a natural language description, C is a set of selection criteria c, and *cw* defines a function of S that return 0, 1, or 2 as coverage weight, where S is a set of satisfied elements c in the selection criteria C of each Ç. One modification has been done in this criteria. The most important for a language is the *Domain Appropriateness* due to the language has to proportionate a solution to a determinate problem into a domain, hence the value of this appropriateness has been increased and now the range goes from 0 to 6 depending on the number of perspectives covered for the language. The categories are as follows.

ç1: *Domain Appropriateness* can be ç1c1) the language covers the structural perspective, ç1c2) the language covers the Functional perspective, ç1c3) the language covers the Behavioral perspective, ç1c4) the language covers the Rule perspective, ç1c5) the language covers the Object perspective, ç1c6) the language covers the Communication perspective, ç1c7) the language covers the Actor and Role perspective. The weight is calculated by the next function where n=1 ç1çi satisfies the perspective or 0 if it not.

$$cw_1(S_1) = \sum_{k=1}^{7} n$$

ç2: *Participant Knowledge Appropriateness* can be ç2c1) the language covers fully this appropriateness, i.e. the language covers the underlying basis and the external representation, ç2c2) the language covers partially the underlying basis and the external representation of this appropriateness, ç2c3) the language does not cover this appropriateness.

$$cw_1(S_1) = \begin{cases} 0 & \text{if } \text{ç2c3} \\ 1 & \text{if } \text{ç2c1} \\ 2 & \text{if } \text{ç2c2} \end{cases}$$

ç3: *Participant Knowledge Externalizability Appropriateness* can be ç3c1) the language covers fully this appropriateness, i.e. the language covers the underlying basis and the external representation, ç3c2) the language covers partially the underlying basis and the external representation of this appropriateness, ç3c3) the language does not cover this appropriateness.

$$cw_1(S_1) = \begin{cases} 0 & \text{if } \text{ç3c3} \\ 1 & \text{if } \text{ç3c1} \\ 2 & \text{if } \text{ç3c2} \end{cases}$$

ç4: *Participant Knowledge Comprehensibility Appropriateness* can be ç4c1) the language covers fully this appropriateness, i.e. the language covers the underlying basis and the external representation, ç4c2) the language covers partially the underlying basis and the external representation of this appropriateness, ç4c3) the language does not cover this appropriateness.

$$cw_1(S_1) = \begin{cases} 0 & \text{if } \text{ç4c3} \\ 1 & \text{if } \text{ç4c1} \\ 2 & \text{if } \text{ç4c2} \end{cases}$$

ç5: *Technical Actor Appropriateness* can be ç5c1) the language covers fully this appropriateness, i.e. the language covers the underlying basis and the external representation, ç5c2) the language covers partially the underlying basis and the external representation of this appropriateness, ç5c3) the language does not cover this appropriateness.

$$cw_1(S_1) = \begin{cases} 0 & \text{if } \text{ç5c3} \\ 1 & \text{if } \text{ç5c1} \\ 2 & \text{if } \text{ç5c2} \end{cases}$$

In the tables 6.1, 6.2, 6.3 and 6.4 the comparison of the three languages is presented:

	Domain Appropriateness	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Technical Actor appr.
RDF(S)	RDF(S) covers structural perspective and object perspective is partially covered. Does not cover rule, behavioural, functional, actor and role perspectives	Appr. dependent on the designer experience. RDF(S) with Protege are intuitive and covers the external representation.	Domain Dependent. Posibility of model determinated situations.	RDF(S) elements are easily distinguished. The number of the phenomena is reasonable. The structure of RDF(S) is partially consistent.	RDF(S) covers partially this perspective. Provides basic elements for automatic reasoning.
Weight	2	1	1	1	1

Table 6.1: RDF(S) Analysis

In the Table 6.1 we can see the main features provided by RDF(S) by after analysing the language using the Semiotic Quality Framework. The Table shows that RDF(S) provides the basic elements in order to satisfy the *Domain Appropriateness* requirements, because it covers the *Structural Perspective* fully and partially the *Object Perspective* and gets a weight of 2. The other appropriateness are covered by RDF(S) only providing the basic requirements of these appropriateness and therefore RDF(S) have the same weight.

	Domain Appropriateness	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Technical Actor appr.
Web Ontology Language	OWL covers structural and rule perspectives. Object perspective is partially covered. Does not cover behavioural, functional, communication actor and role perspectives	Dependent on the designer experience. The phenomena are easily distinguished in OWL and Protege	Domain dependent. Posibility of model main databases concepts.	OWL elements are easily distinguished. The number of the phenomena is reasonable. Symbol discrimination is not fully covered. The structure of owl is consistent	OWL covers this perspective. It is designed for automatic reasoning.
Weight	3	1	1	1	2

Table 6.2: Web Ontology Languages Analysis

In the Table 6.2 we can see the main features provided by OWL by after analysing the language using the Semiotic Quality Framework. The Table shows that OWL provides the basic elements in order to satisfy the *Domain Appropriateness* requirements, because it covers the *Structural Perspective* and *Rule Perspective* fully and partially the *Object Perspective* and gets a weight of 3. The *Technical Actor Appropriateness* is fully covered because OWL is designed to support this appropriateness and gets a weight of 3. The other appropriateness are covered by OWL only providing the basic requirements of these appropriateness and because of that OWL gets the same weight in these sections.

	Domain Appropriateness	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Technical Actor appr.
Topic Maps	Topic Maps covers structural perspective an partially Object perspective. Does not cover Functional, Behavioural, Rule, Object, Communication and actor and role perspective.	Appr dependent on the developer experience. In combination with Protege is not enough intuitive.	Domain dependent.	Topic Maps diferentiates each symbol from others. The number of the phenomena is reasonable, but less reasonable than OWL or RDF(S). The structure is partially consistent. Not enough expressive economy. The structure is consistent.	Topic Maps covers this appr. It s designed to facilitate automatic reasoning.
Weight	2	1	1	1	2

Table 6.3: Topic Maps Analysis

In the Table 6.3 we can see the main features provided by Topic Maps by after analysing the language using the Semiotic Quality Framework. The Table shows that Topic Maps provides the basic elements in order to satisfy the *Domain Appropriateness* requirements, because it covers the *Structural Perspective* fully and partially the *Object Perspective* and gets a weight of 2. Topic Maps also covers *Technical Actor Appropriateness* because it provides more elements to detail data and facilitate the automatic reasoning by the information agents and gets a weight of 2. The other appropriateness are covered by Topic Maps only providing the basic requirements of these appropriateness.

	Domain Appropriateness	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Actor and Role appr.
Web Ontology Language for Services	OWL-S covers fully structural, functional, rule and object perspectives. It does not cover communication, behavioral and actor and role perspectives.	Dependent on the designer experience. The phenomena are easily distinguished in OWL-S and Protege	Domain dependent. Posibility of model main databases concepts. OWL-S adds the possibility of add processes.	OWL-S elements are easily distinguished. The number of the phenomena is reasonable. Symbol discrimination is not fully covered. The structure of owl is consistent	OWL-S covers this perspective. It is designed for automatic reasoning.
Weight	4	1	2	1	2

Table 6.4: Web Ontology Languages Analysis - Services

In the Table 6.4 we can see the main features provided by OWL by after analysing the language using the Semiotic Quality Framework. The Table shows that OWL provides the basic elements in order to satisfy the *Domain Appropriateness* requirements, because it covers the *Structural Perspective* and *Rule Perspective* fully and partially the *Object Perspective* and gets a weight of 3. The *Technical Actor Appropriateness* is fully covered because OWL is designed to support this appropriateness and gets a weight of 3. The other appropriateness are covered by OWL only providing the basic requirements of these appropriateness and because of that OWL gets the same weight in these sections.

6.1.1 Final comparison

After this comparison of languages it is possible to compare with the previous requirements of Invenio showed in Table 6.5 by adding the weight needed for each language appropriateness.

	Domain Appr.	Participant Knowledge Appr.	Knowledge Externalizability Appr.	Comprehensibility Appr.	Technical Actor Appr.
Invenio Case	In the case full structural perspective is needed and partially Rule perspective. Functional, Behavioral, Communication and Actor and Role are not needed	High requirement. A tool with enough functionalities is needed.	Partially needed. The reality are two databases which have to been modelled.	Medium/high needed. Phenomena should be clear for a better understanding of the participants.	Highly needed. The case is based on automatic reasoning.
Importance	High	High	Medium	Medium	High

Table 6.5: Invenio requirements

	Domain Appropriateness	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Technical Actor appr.
Weight	3	2	1	1	2

Table 6.6: Invenio Requirements

The weights assigned to the Table 6.6 are based in the requirements specified in table 6.5. In this table the requirements needed are specified in natural languages. We have translated these requirements to a more proper representation in order to make a comparison with results obtained from the languages analysis. The final comparison of the languages with the Invenio requirements is shown in the Table 6.7.

	Domain Appr.	Participant Knowledge Appr.	Knowledge Externalizability appr.	Comprehensibility appr.	Actor and Role appr.	Total Weight
RDF(S)	2x3	1x2	1x1	1x1	1x2	12
OWL	3x3	1x2	1x1	1x1	2x2	17
OWL-S	5x3	1x2	2x1	1x1	2x2	19
Topic Maps	2x3	1x2	1x1	1x1	2x2	14

Table 6.7: Comparison of the Web ontology languages and the Invenio Requirements

In Table 6.7 we have added a new column where it is specified the final weight of the languages. We have multiplied the results obtained for each language by the weight of the appropriateness given in the Table 6.5. With this new column and the new weights added we can distinguish more easily which is the most adequate language.

We can see in this final table that is the most appropriated language in our case, is the Web Ontology Language for Services, OWL-S. This is due to the elements that it provides for creating first order predicates and constructs for model services. This service feature it is no fully needed in a database but it adds more functionality to the solution of our problem. The other languages do not offer this feature because RDF(S) is not designed to offer these facilities and Topic Maps is oriented to create better quality relationships between its elements but does not provide enough elements for constraining these relations.

The difference between OWL-S and OWL is really big because the Invenio case is based in a static representation of the data. It is not really needed to add services or processes to the solution of the problem but it can be a good improvement.

6.2 Concluding Remarks

The comparison of the previous languages has been done and the conclusions are that OWL is the most suitable language to solve our case due to the Invenio's requirements. If we want to extend the requirements to make available the service then OWL-S will be the best alternative in our case. The following sections will contain the practical approach to the problem where the languages are analysed. OWL-S is not included in this analysis due to a service provider is not included in the company requirements.

Part III

Practical Approach to the Case Study

Chapter 7

Conversion Parsers

In this chapter, we are going to explain the development of three conversion parser. These three conversion parsers have been created due to the necessity of translating the format of the databases to another representation that the information agents will be able to understand. This new format is the format the ontologies have in the three Web ontology languages selected. Then it is necessary to develop three parsers, one for RDF(S), one for OWL and the last one for translating the databases to a Topic Maps representation.

7.1 The conversion parser

As it is explained in Chapter 3 we have two relational databases and we want to transform them to an ontological representation. We will do this by developing three conversion parsers. We will transform the data to this new representation and our tools for managing ontologies will access to the data in the new format. This data will be semantically enriched and an information agent will be able to process it.

Before start the development of the parsers we have been looking at similar projects. We can see in [3] and [34] there are several ways for converting data stored in relational databases to semantically enriched representations. These two articles explain how to convert data stored in relational databases to a representation in RDF.

7.1.1 Conversion Rules

For converting our data stored in relational databases we have been following several rules. This rules are useful for mapping data in different representation. Our rules for translating the database to an ontological representation have been based on the relational schema designed. This schema was extracted from the databases, directly from the data in one case and from the DTD specification from the other case. Once we had the initial schemas we translated them to a relational representation. This makes a clear difference between our approach and the other cases. We had freedom for modelling the initial schema from we will extract the ontology.

For creating the conversion rules we looked at the conversion rules specified in [34]. In this approach rules are defined more concretely but there is no description for fields that are not in 1NF or more complex situations. In this article is explained the mapping process between a database and a ontology:

- Capture the information from a relational schema.
- Analyse the obtained information to construct ontological entities. Rules to translate from one representation to another can be created. Recommendations for creating this rules are specified below.
- Create the ontology ("schema translation") by using the rules specified before. With this translation all ontological entities will be created.
- Evaluate, validate and refine the ontology.
- Form a knowledge base ("data migration").

These steps have been followed in our research process but adding some differentiations. Once we obtained the information from the relational schema (we have created this schema) we proceeded to construct the ontology. The rules for creating the new ontology are based in the following recommendations:

- Each class in the database is an entity in the ontology.
- Each relationship in the database is a relationship in the ontology.
- M:N relationships in the database require a special treatment.
- We do not have primary and foreign keys in the ontology.

If the relational schema has not enough expressiveness power (with different representations this may happen) a extension of the ontology following the developer criteria will be made. This will happen if we have to deal with associative relationships. We will have to model in an ontology three or more classes with an M:N relationship and in the ontology may appear loops. We will have to model our new ontology following the specifications of the problem and the schema of the database.

There is available Database to RDF, D2R MAP [10]. D2R MAP is a declarative language to describe mappings between relational database schemata and OWL/RDFS ontologies. The mappings can be used by a D2R processor to export data from a relational database into RDF. The problem with this language is that we still have problems with not normalized data and our databases are not normalized.

7.2 Structure of the Parsers

The three parsers have a similar structure because they obtain the data from the same source. Therefore, the method for reading the data from these databases is the same for all the three parsers. These parsers read each line of the files and then they process the. The database containing the directory service for Sweden has a XML format and each field of the database is clearly defined. The different elements of the database are easily identifiable and they are easy to process. Due to that, every time a line from the file is read is possible to know which field of the database are we reading and it is not difficult to transform it into the ontology representation. It is also important not to forget the peculiarities of each language. RFD(S) and OWL are quite similar meanwhile Topic Maps has some particularities in the way of organizing the data and that makes the parser developing more difficult.

In the three languages, RDF(S), OWL and Topic Maps the data is divided in two different parts. In the first part the data types, attributes and the fields of the database are defined and in the other part of the Web ontology language file the data is represented. In the files formatted with RDF(S) and OWL the definitions of the attributes and the fields of the database are located at the beginning of the file and the data is stored at the end of the file. The data only begins when the definitions are finished. In the file with XTM format this is different. Are two different parts inside the file. The definitions of the classes can be in any place of the file, for example between the data of two instances, but it is possible to organize the file manually and put all the definitions at the beginning of the file. The relationships can be placed in any place too. They use their identifier to create the relationship. Within OWL and RDF(S) the relationships are indicated inside the class.

7.3 Common concepts

We analysed the data stored in the databases and from this analysis we extracted the particular format of both databases. The way for reading the original data is the same. Therefore we developed the same way for reading the data for all conversion parsers and developed a specific output format for each one. The time needed to develop the first conversion parser was higher than the time needed for developing the others.

The parser for converting one database to an ontological format consists in a loop that reads every line of the database, that is, the XML files and it process them. The fields of the databases are clearly identified by the structure of the XML file. One example of this database is the next one:

```
<Publications>
<Comment>TelAdress initial</Comment>
<NewFile>TeleAdrALLJNN820040131112601.xml</NewFile>
<Publication>
<Id>12457</Id>
<Terminate>no</Terminate>
<Placement>
<RegionCode>8</RegionCode>
<RegionName>Gotland</RegionName>
<DirectoryArea>0498</DirectoryArea>
```

```
<SectionCode>1</SectionCode>
<ClassifiedCode>0</ClassifiedCode>
<Internet>yes</Internet>
</Placement>
<Row>
<Number>1</Number>
```

The simple attributes of each field are at the beginning of the structures and these fields can contain more structures inside them. Therefore the database is not in First Normal Form. One structure "Publication" can contain inside zero or more "Row" and this "Row" can contain zero or more "Name", "ExtraName" or "Address" for example.

The output format is the same for the three ontology languages but with the specific formatting for each language. All three languages will represent the same structure. This structure is explained in the Chapter 8 and it is the new design of the databases in an ontology. Here we will explain the specific format of the files.

The program is a loop that reads a "Publication" once at a time. It stores the useful attributes and it discards the others. Some attributes like the number of publication are not needed in the new design. Once we have read the entire publication we write into the output files the read data. This data will we written in the format of the new ontology, fulfilling the new structures created. The relationships will be added at the same time. We only have to read one time the file due to the tools used for accessing the databases. We will use the specific way for reading data of the tool used to store it. With Protégé we will read the OWL and RDF(S) files and we will use Ontopia Omnigator to read the Topic Maps format.

7.4 Differences between conversion parsers

Due to the particularities of the different plug-in used with Protégé to model the ontologies in the three languages we had to use different query methods. Neither OWL plug-in nor TM Tab plugging for Topic Maps do not provide any querying method. Therefore we had to seek alternative ways to do the querying process. Within Topic Maps we had another problem: it is not possible to import an XTM file to the TM Tab plug-in, it is only possible to export the model and the instances created with Protégé. This problem causes the impossibility to use Protégé to query our data and we found Omnigator, a Topic Maps management environment. We were able to load our data into the Omnigator and query it.

Also, all plugging have their particularities when the data is loaded into the ontology. Protégé offers more freedom to load the data. This freedom is due to the reading Protégé does of the whole ontology before show it on the program screen. When the data is translated into the ontological format it is not completely related. The translator program reads from the database sequentially and puts the identifications to the classes depending on the data. When Protégé reads these identifiers it knows if the same identifier has been read before. If it was read before Protégé relate this identifier to the new one. For example, we have a two side relationship between Person and Title. When we read a Person we have to relate it to a Title and from one title we should be able to see all the persons who have this title. If we have one person "Carlos" our translator will relate it to the title "Student" and automatically Protégé will relate "Student" to "Carlos". We only have to create one side of the relationship. This situation does not happen directly when using the Ontopia Omnigator but it is possible to access to an option for deleting repeated values. It is possible even to export the ontology to RDF format.

7.5 Concluding Remarks

In this chapter, we have seen the structure our conversion parses have. We shown in which way we have created our programs and we have seen there are similar problems in other projects. Our approach for developing the conversion parsers has been based in a combinations of the solutions proposed by [3] and [34] and the requirements of our databases. We have seen that our data once converted will have the same structure for the three languages, but the functionalities will depend of the ontology design.

Chapter 8

Design of the Models

In this chapter, we are going to present the new design of the databases. We will explain the motivation of our designs the way of working and the result of the design. The main classes, attributes and relationships will be explained.

8.1 Need of a new design

We arrived to the conclusion in the article [13] that a new model was needed for our databases. This was needed due to the old design was not improved for a proper access to the databases. Also, we made an interview to the expert of the company and we got a clear view of the databases' purpose. The full interview can be found in the Appendix E. One of the main conclusions of the interview was that the databases are designed for printing "Yellow Pages". The expert explained us that the organization of the data is the most suitable for printing directory services.

Internally the information is organized in "Publication" and each "Publication" can contain one "Placement" that describes one region and one "Publication" can contain * "Row" that contains the information about one telephone number. This information can be several names or addresses. This description of the data indicates that the database is not in first normal form and should be improved if we want an intelligent and quick access to our data.

The design of the new models of the databases and the ontologies has been divided in three steps. First, we created a new design for the databases because the old design is not the most accurate design that we can obtain from these databases. With the new design we will create the base ontologies. Second, we improved the ontologies by adding relationships and modifying the initial ontology design to obtain a better conceptualization of the proposed domain. Third, we tested the ontologies and we added the necessary modifications to our ontology to obtain the best results. This design has been done by using Protégé, a tool for designing knowledge representations described in Appendix C.

8.2 Extracted Model

We are going to show how the model has been extracted. We will explain the reasons for this specific model and the classes and relationships created. The model is designed in UML from the previous schemas extracted from the databases.

We have based the extraction of the databases' model in the existing data. We can not create more attributes in the schema than the existing data. We started designing the new models for the databases and we based these new models on the interview to the expert of the company. For designing the Swedish database we had the help of the original database design but for designing the Norwegian new schema we had no initial design.

The core class of the new Swedish database model is "TelephoneNumber". This class contains one telephone number because is the core of a Directory Service, all is around this number. One person can have several telephone numbers in one or more addresses. The main question here is to access the information around a telephone number. We have centered the design of the database on this class, the telephone numbers are the key concept in a Directory Service. The class Address is another important class. Around this class we can find the information about the people who lives in this place, the region and the country. We have decided to divide the data from the original database in other classes for a better structuring and accessing to this information.

The Norwegian database is a "Yellow Pages" database, a directory service containing only companies and the data for locating them. It contains telephone numbers of companies in Norway. In this case we do not have any initial model just the data and we made an approximation to the model. We have made the same than with the Swedish database. We have centered all the information on the "TelephoneNumber" class and we have organized all the information in the classes "Company" instead of "Person". The structure for organizing the addresses will follow the same way.

In the pictures 8.1 and 8.2 we can see the initial model of both databases. These are the models extracted from the data available and the interview described at E.

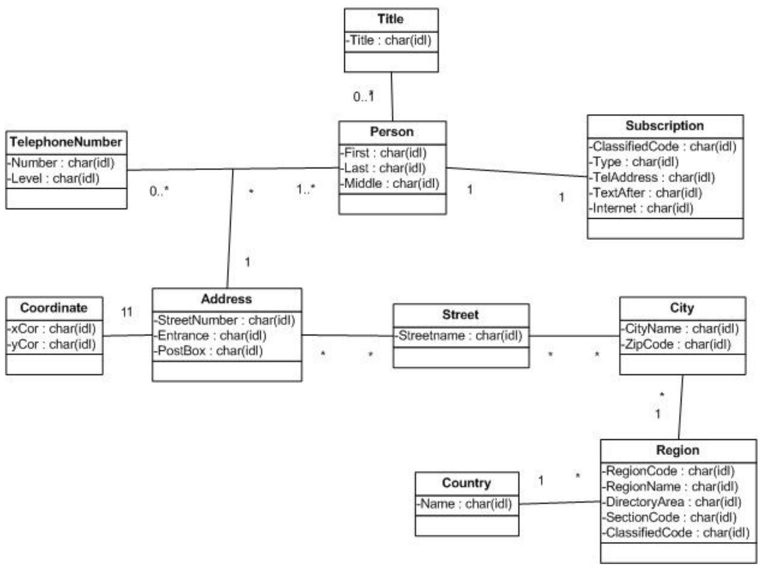


Figure 8.1: Swedish Schema

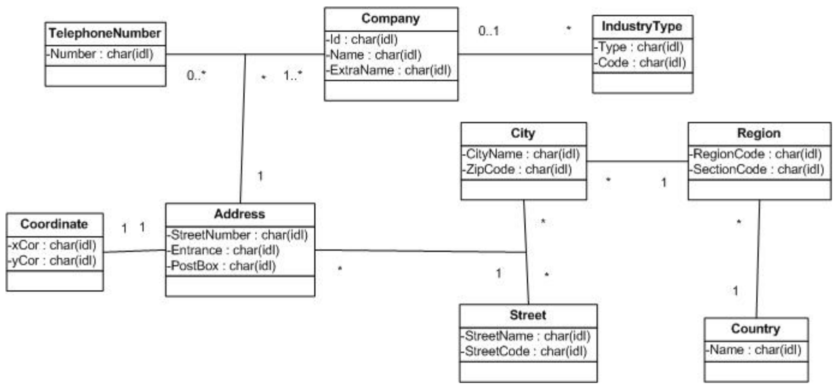


Figure 8.2: Norwegian Schema

Looking at both models we can see that they are quite similar. Mainly, the classes are the same and the relationships are also similar. This is because they are representing the same fact, a directory service but with different data. At this point we decided to merge both databases in one common ontology. Merging both databases in one ontology we will get several advantages. For the company will be easier to manage one database instead of one. It is more simple to design just one general ontology than two and it will be easier to upgrade this merged ontology if we want to add more data to it.

We have created the new design for the databases and we have seen that this model can be merged into one more general model. This will improve the performance of the queries to the databases and will make easier the design of the ontologies.

8.3 New Ontology Design

The new design of the ontology is presented in this section. The ontology will be created from the merged database schema and it will be adapted to the needs of the Web ontology languages used.

The main design has been based in three classes: Human, Information, Organization and Place. All our concepts can be placed in one of these three sets. Human will only have one subclass: Person. This is because in our domain we should separate a Person from the other concepts of the ontology. Organization will have two subclasses. These subclasses are Company and PublicAdministration. We have this division for differentiating both types of organization available in the data. Information will have several subclasses. These subclasses are Title, VisitCard, StreetNumber, Coordinate, Subscription and TelephoneNumber. We have classified these classes as subclasses of Information because all of them only provide information about one place. By themselves these classes are nothing. Place will have as subclasses several concepts. The subclasses of Place will be City, Street, Region and Country. It is easy to see that all these classes represent a physical space where a person can be in a certain moment.

Human this class describes all the human beings that can be in our domain.

This class has no attributes and it is used to organize the ontology.

Information this class describes the information that can identify a place. This information is used to identify a concrete place and a concrete person. Without this information will be impossible to locate a person or a place.

Place this class describes the physical space of a place. This class is used to describe where a person can be in one determinate moment. In combination with the previous class it can be determined where a person has a tenement.

Organization this class describes the concept of organization. It has the attributes for identifying the organization and it has two subclasses, Company and PublicAdministration.

Person this class describes a person. This class will represent all the people of our directory service. The class person has the attributes Full, First, Middle, Last and the relationships hasTitle, isSubscribed and hasVisitCard. The attributes First, Middle and Last represent the components of common name and the attribute Full represents all the three attributes together. The relationship hasTitle represents that a Person has achieved one or more title in his live. A person can have one or more title assigned to it. The relationship hasVisitCard represents the information needed to find the place where this person lives. One person can have several places for living therefore one person can have more than one VisitCard. The relationship isSubscribed represents the subscription every person has. It is a subscription to the company provider of the telephone number. The relationship worksFor represents the companies a person is currently working for.

VisitCard this class describes the information about the place where a person can live. With this class it is possible to know what the telephone number of one person is and where this person lives. This class contains only relationships to other classes. The relationships are: hasTelephoneNumber, belongsPerson, hasPostalCode, hasStreetNumber, hasStreet and hasCity. The class is always related to one person and will identify where this person live, including the telephone number of the people who live in this place. hasTelephoneNumber indicates what telephone number has the card. hasPostalCode indicates the postal code of the person's building. hasStreet indicates the street where the person lives. hasStreetNumber indicates the number of the street where the person lives. hasPostalCode indicates the postal code of the building where the person lives. hasCity indicates the city where the person lives.

PostalCode this class represents a typical postal code. It only contains one attribute and it is used to represent the postal code number. This class contains two relationships: isInCity and isInStreet. The relationship isInCity represents in what cities this postal code is and the relationship isInStreet represents in what streets this postal code is.

StreetNumber this class represents a street number. It contains an attribute, StreetNumberAttr and contains a number. This class has the relationship isInStreet to indicate that this street number is in a street. The street number is the number of a building inside this street.

TelephoneNumber this class represents a telephone number. It only contains the attribute Number that is used to represent the value of the telephone number.

Subscription this class represents the subscription what a person has with the provider of the directory service. All persons have one subscription at least and this subscription contains all the data necessary for the company. The attributes are TelAddress which represents the telephone number of the person, Type which represents the type of TelAddress, ClassifiedCode used for internal identification, TextAfter used to add comments to the subscription and Internet used to know if this person has an internet subscription. This class has the relationship hasSubscriptors which contains all the persons with a subscription.

Title this class represents the different titles that a person can have. It can represent for example student or architect. It has only one attribute, TitleName for representing the name of this title. It has one relationship with Person, hasPerson and it represents the persons who own this title.

Coordinate this class represents the coordinates of the building where the person lives. It contains the attributes xCor and yCor for representing the coordinates of the building but in the source databases where no data associated to these attributes. This class is related to Person and in a future work will contain the necessary data.

Street this class represents the physical place of a street. It contains only one attribute, StreetName and it is used to store the name of this street. It contains the relationship hasStreetNumbers for knowing what buildings are in the street. A street can be located in different cities, for example the street Dronningensgate is in Oslo and Trondheim.

City this class represents the physical space of a city. It contains only one attribute, CityName and it is used to store the name of this city. It contains the relationship isInaRegion that represents that this city is located in a concrete region.

Region this class represents the physical space of a region. It contains the attributes RegionName used for specifying the name of the region, RegionCode that represents the code of this region, SectionCode used for representing the code of a section of this region, ClassifiedCode used for internal control and DirectoryArea used for representing the area inside the region. It has the relationships hasCities that represents the cities that are in this region and isInaCountry that represents in which country is this region. It contains the relationship hasCities for representing what cities are in this region.

Country this class represents the physical space of a country. It only contains one attribute, CountryName and it is used to store the name of this country. It contains one relationship, hasRegions that represents the regions that this country has. In our data there are only two countries: Sweden and Norway.

Company this class represents the concept of a company. It has the attributes inherited from Organization Id, Name, ExtraName and Subdivision. It has the relationships inherited from Organization hasWorkers and hasVisitCard.

PublicAdministration this class represents the concept of a public administration like a town hall or the postal service of a country. It has the attributes inherited from Organization Id, Name, ExtraName and Subdivision. It has the relationships inherited from Organization hasWorkers and hasVisitCard.

The relationships will be created in the ontologies and will fix previous mistakes and will improve also the access to the data by creating new relationships and restrictions between classes. The following relationships are the initial ideas for the ontologies:

These relationships have been based in the database of the New Zealand Census Data Model and Dictionary. The ontology designed is shown in Figure 8.3.

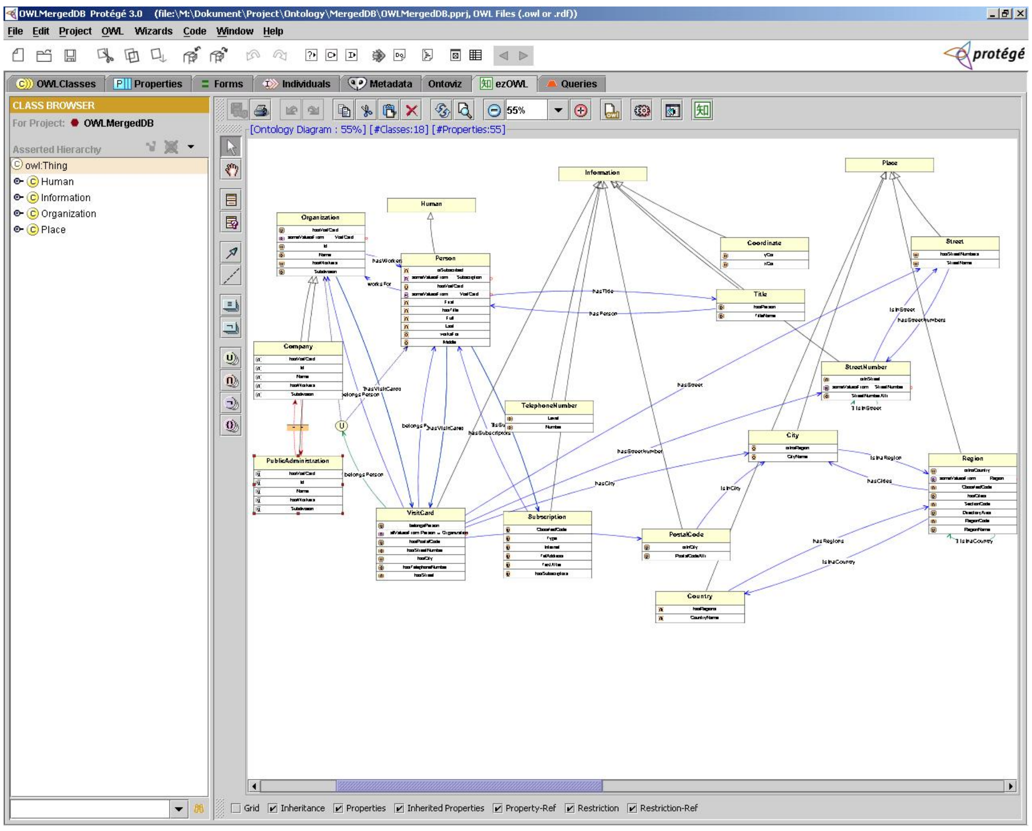

OWLMergedDB Protégé 3.0 (file:\M:\Dokument\Project\Ontology\MergedDB\OWLMergedDB.pprj, OWL Files (.owl or .rdf))
File Edit Project OWL Wizards Code Window Help
protégé
OWLClasses Properties Forms Individuals Metadata Ontoviz ezOWL Queries
CLASS BROWSER
For Project: OWLMergedDB
Asserted Hierarchy
owl:Thing
Human
Information
Organization
Place
55%
[Ontology Diagram : 55%] [#Classes:18] [#Properties:55]
Information
Place
Human
Organization
Person
Coordinate
Street
Title
Company
StreetNumber
TelephoneNumber
City
PublicAdministration
Region
VisitCard
Subscription
PostalCode
Country
Grid Inheritance Properties Inherited Properties Property-Ref Restriction Restriction-Ref

Figure 8.3: Initial Ontology

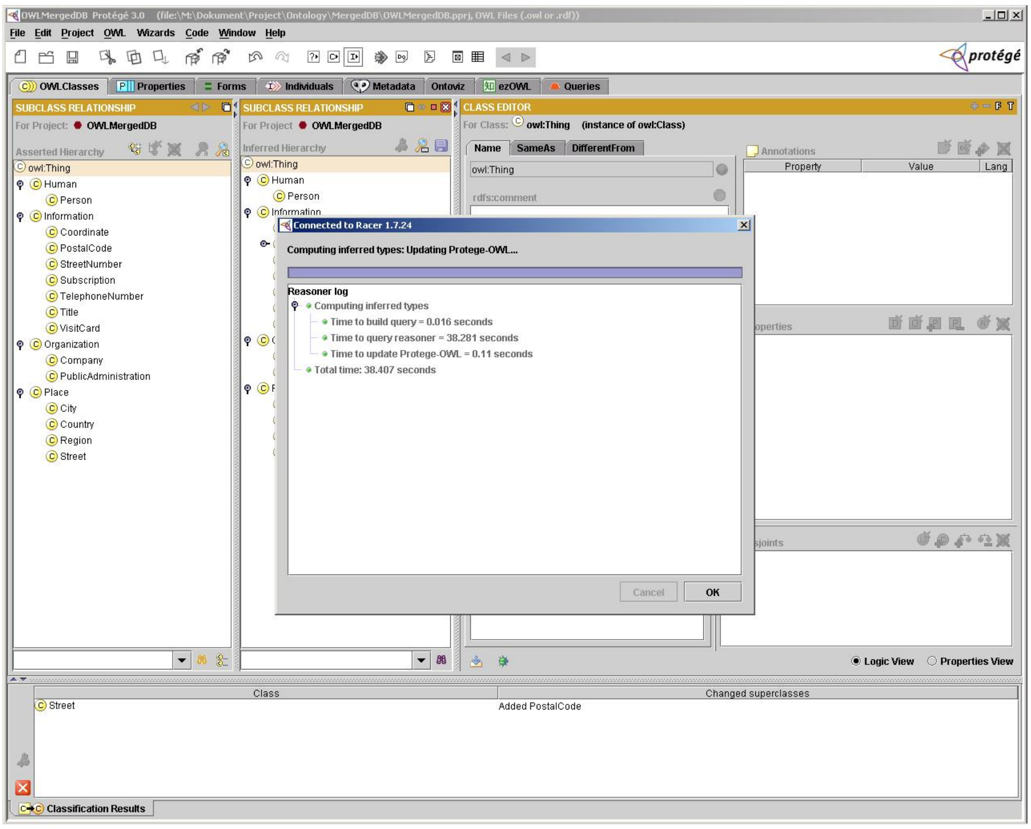
8.4 Restrictions and OWL

The main difference between OWL and the other languages are the restrictions that are possible to add to the ontology. As it is explained in [16] property domains and ranges should not be viewed as constraints to be checked. They are used as axioms in reasoning. If we want to restrict the the individuals that belong to a class we have the restrictions provided by OWL. We can create these restrictions by using quantifiers, cardinality or has value restrictions. This is the main advantage of OWL. We have created the following restrictions:

- A necessary condition to be a Person is to have VisitCard.
- A necessary condition to be an Organization (Company or PublicAdministration) is to have Visitcard.
- A necessary condition to be a TelephoneNumber is to have a VisitCard.
- A necessary condition to be a StreetNumber is to be in a Street.
- A necessary condition to be a Street is to be in a City. We can infer the PostalCode attribute.
- A necessary condition to be a Region is to be in a Country.

- A necessary condition to be a VisitCard is to belong to a Person or to an Organization.

It is important to compute the the class hierarchy. In a big ontology with thousands of instances this is almost mandatory to have an ontology in a maintainable and logical state. To automate this process it is useful to use a reasoner. Protégé can interact with RACER, a reasoner, to compute this hierarchy. We have done this classification but with a few instances. With the hole database, or even a thousand of instances our computer needed so much time. The results of applying these restrictions are shown in the Picture 8.4. The OWL Sub-language is OWL DL as it is shown in the Picture 8.5.



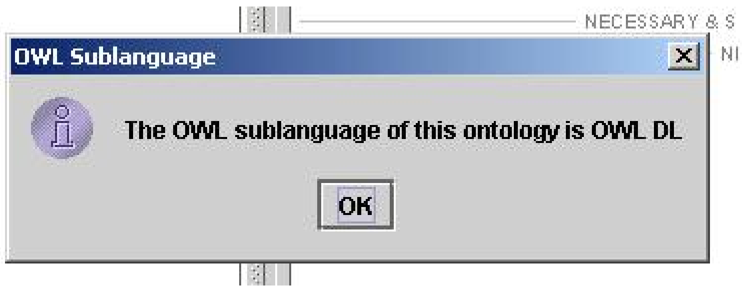
Connected to Racer 1.7.24

Computing inferred types: Updating Protege-OWL...

```
Reasoner log
  Computing inferred types
    Time to build query = 0.016 seconds
    Time to query reasoner = 38.281 seconds
    Time to update Protege-OWL = 0.11 seconds
  Total time: 38.407 seconds
```

Class	Changed superclasses
Street	Added PostalCode

Figure 8.4: Inferred types in our ontology



The OWL sublanguage of this ontology is OWL DL

Figure 8.5: OWL Sub-language used

8.5 Concluding Remarks

In this chapter, we have seen how we have modelled the new databases and ontology. We merged the databases because we realized that they contain mainly the same type of data. Also, merging both databases will allow the extension of databases. Now we do not have enough data to get the best efficiency from it.

Chapter 9

Analysis of the Transformation

In this chapter, we are going to see the analysis and comparison of the three Web ontology languages for representing our domain model. We will compare the transformations done to the original UML design with the final ontology in the three languages. We have created several tables which are in Appendix D. This tables contain the results of the analysis of our models. We have not included in this analysis OWL-S due to it is an extension of OWL. In this case study Web services are not needed and to model the case study in OWL-S will produce the same output as OWL.

9.1 Information need based evaluation

Once we have done the transformation of the data we have three databases containing the same data but with a different representation. Now we will proceed to the evaluation of this transformation. This analysis of the transformation will be one of the main results of the thesis. As we have commented previously this representation is in three different ontology languages: RDF(S), OWL and Topic Maps. For evaluating the three languages we will propose the following method. We will have two sets of data, one with "control data" that we will use to know if the answers of the languages are correct and another one with "observed data". The database schema resulted from the case study above is here used as observed data and we will use it fulfil the three databases. The observed data are compared with a set of control schemas in an empirical experiment. This data comes from the original databases with a transformation to the three ontology languages. We will use several selected queries to different sets of this data to produce an optimal evaluation. These sets will contain different amounts of data, and they will be increasing the amount of data in every iteration.

The answers obtained from the queries will be compared to the answers provided by the UML model and the control data. Combining the two answers, the one from the languages and the observed data and the one from the UML model and the control data we will calculate the precision and recall measures.

Once we have calculated these precision and recall measures for each iteration of the amount of data we will be able to calculate an average result of successful answers to our queries from the ontological databases. This average result will be the recommendation of one language.

Once we have set the parameter of the evaluation in the lines above we will proceed to do the evaluation. This evaluation will be based in the previous indications and it will be applied to the ontology designed in the three Web ontology languages.

9.1.1 Test queries

We are going to evaluate the data stored in our ontologies. To perform a correct evaluation we should compare the data in the new representation with the data in the old representation. To do this several queries will be done and the results will be shown.

Once we have the data stored in our ontologies we will proceed to evaluate it. We can not compare the results of our queries because we do not have any way to consult the original databases. These databases are XML and TXT files and we do not have the proper tool to consult them. We can look at the model created in the previous chapter and try to suppose the answers. This is not difficult because in a relational database the language used is based in relational algebra. Therefore it is possible to make the queries in paper or say if it is possible to do or not.

The queries to the ontology are done by using the engine provided by Protégé. This engine is a tab that provides a simple interface to query the data stored. It is possible to do several types of queries but it has limitations. For example it is not possible to count the elements returned from a query or return attributes from a query. We will query the databases in OWL and RDF(S) using Protégé and for Topic Maps we will use Ontopia Omnigator. We are using different engines because we can not load in Protégé the XTM format of Topic Maps. We used Protégé and the TMTab plugging for designing and export the ontology but it is not possible to import the ontology. The example of one of these queries is shown in Figure 9.1 and Figure 9.2. The initial queries are the following:

Q1 In these queries we are going to consult the people who have a fixed surname. These queries can be done in SQL but the complexity of the is increasing.

- **Q1.1** Return the people who live in my street, in my city and have the same surname as me.
- **Q1.2** Return the people who live in my street, in my city, have the same surname as me and are librarians.
- **Q1.3** Return the telephone numbers of the people who live in my street, in my city are librarians and have the same surname as me.

Q2 In these queries we are going to consult the people who have a specific title. These queries can be done in SQL but the complexity of the is increasing.

- **Q2.1** Return the people that have the same title as me.

Q2.2 Return the people that have the same title as me and live in the same street as me.

Q2.3 Return telephone numbers of the people that have the same title as me and live in the same street as I.

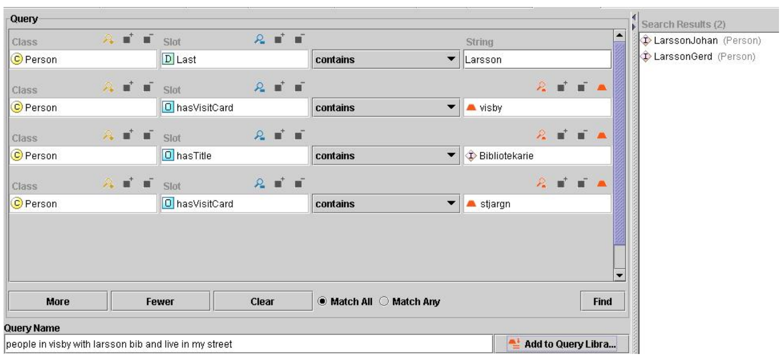
Q3 In these queries we are going to consult the people who live in a specify building in an specific street. These queries can be done in SQL but the complexity of the is increasing.

Q3.1 Return the people that live the same building as me.

Q3.2 Return the people that have the same title as me and live in the same building as me.

Q3.3 Return telephone numbers of the people that have the same title as me and live in the same building as me.

We can modify these queries by querying the people who live in the same building as me (we have to ask for the street number), we can define "neighbor" and with the given results of a query decide if the answer fits in the definition of neighbor, etc. We can not get automatically an attribute as answer counting the cardinality of the answers but we have this number as information of the query. If we try to do these queries with SQL we will need enough knowledge and experience using this language. The initial queries are not complicated but to get answers from Q1.3, Q2.3 and Q3.3 is quite complicated and it will require several consuming time operations. This is one of the advantages of the sematic Web languages, are simple to use if you have the right tool and can retrieve the information quickly.


Query

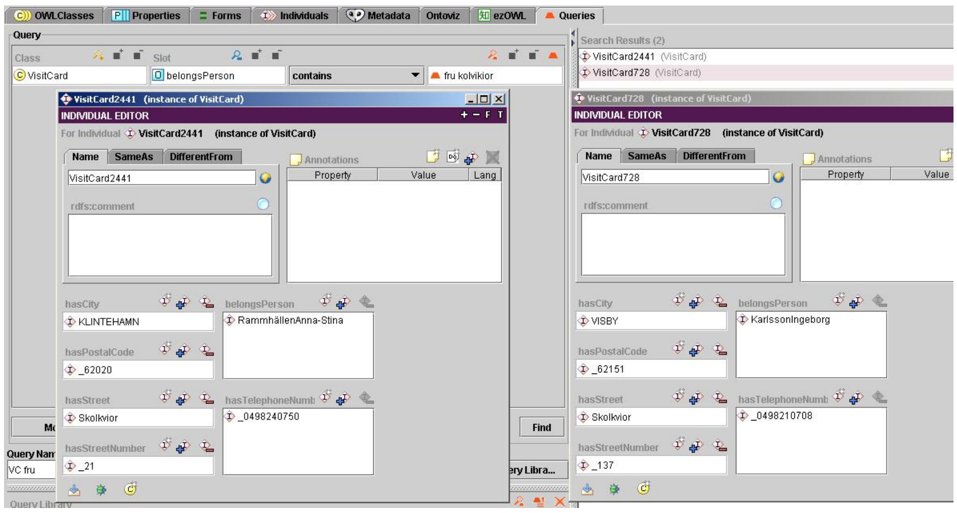
Class	Slot		String
Person	Last	contains	Larsson
Person	hasVisitCard	contains	visby
Person	hasTitle	contains	Bibliotekarie
Person	hasVisitCard	contains	stjargn

[✓ Match All] Match Any

Query Name: people in visby with larsson bib and live in my street

Search Results (2)
- LarssonJohan (Person)
- LarssonGerd (Person)

Figure 9.1: Query 1



Query
Class: VisitCard | Slot: belongsPerson | contains | fru kolvikior

Search Results (2)
VisitCard2441 (VisitCard)
VisitCard728 (VisitCard)

VisitCard2441 (instance of VisitCard)
INDIVIDUAL EDITOR
For Individual VisitCard2441 (instance of VisitCard)
Name: VisitCard2441
hasCity: KLINTEHAMN
belongsPerson: RammhällenAnna-Stina
hasPostalCode: _62020
hasStreet: Skolkvior
hasTelephoneNumb: _0498240750
hasStreetNumber: _21

VisitCard728 (instance of VisitCard)
INDIVIDUAL EDITOR
For Individual VisitCard728 (instance of VisitCard)
Name: VisitCard728
hasCity: VISBY
belongsPerson: KarlssonIngeborg
hasPostalCode: _62151
hasStreet: Skolkvior
hasTelephoneNumb: _0498210708
hasStreetNumber: _137

Query Nam: VC fru

Figure 9.2: Query 2

These queries can be done easily with Protégé by using the queries Tab. We have only tested these queries with the ontologies in OWL and RDF(S) and we have the same results. These results are the same because we have the same ontology and the restrictions are controlled in the translation process. An information agent need to control these restrictions and this is where OWL can perform a better efficiency.

9.1.2 Problems with the queries

As we have seen in the line above we can not do a proper comparison of the languages. First, we do not have a way for accessing the original databases. We need a proper way for accessing them in order to do a good comparison with the original models. Theoretically, as we have seen, the Web ontology languages are quite better than the relational, at least in this concrete use case. Second, we do not have the same tools for comparing the results of our queries. This may have a significant weight in our evaluation because the ways for reasoning may be different and the final conclusions from the reasoning tool can be different. Third, the ontologies in this case are similar. We have modelled the ontologies in order to substitute the original design improving it and translating it to the Web ontology languages. The final ontologies are similar and the results from the queries are similar as well. Fourth, the engine used for comparing the languages is not the proper engine. The languages are designed to facilitate the reasoning of information agents and the tool has not the proper environment for reasoning. In the next section we will proceed with other way for comparing the languages.

We tried to access to the databases by using a more complex tool based in Description Logics. The tool was RACER. We found very complicated to write the queries is the specific language of RACER and we discarded it. We had with RACER the same problem for comparing the results with the other languages.

We have seen in this section that we need another way for comparing the ontologies. In the future work will be necessary to improve the access to the data stored in the ontology but now we will continue with other way for analysing the representations.

9.2 Meaning based evaluation

The comparison of the transformation has been divided in three sections. We will compare what features can be represented in RDF(S), OWL and Topic Maps against the features represented in UML. In UML the new database is represented and in RDF(S), OWL and Topic Maps the ontology is represented. For each comparison we will analyse if the concepts in the languages are represented or not in UML, if the concepts are specified or not in UML, if the cardinality is the same in both representations and if the restrictions are specified or not in UML.

9.2.1 Description of the transformation

In the previous chapters we have shown the transformation we have applied to the original data. This data was two databases, one containing the Swedish directory service and the other one the Norwegian directory service. We had the database schema from the Swedish database but nothing from the Norwegian database. The databases were formatted in XML and with an unknown format respectively. We applied a process of redesign to these databases and in the following lines we analyse this process of redesign and transformation.

We will compare the final ontology obtained and represented in the three Web ontology languages with the final UML model of the database. We will not use the initial databases' schema due to it is quite simple. In the initial model (only the Swedish database had any kind of model) we can not query anything. We do not have the tools for accessing two databases in such format. The model is not for querying, it is just for printing the directory service. Therefore, the comparison between the base model and the ontology model obtained from the UML design of the database would be worthless.

We will use the merged database model to do the analysis of the transformation. This analysis of the transformation will consist in compare what can the ontology in the three Web ontology languages represent that the UML design can not. The concepts to do this analysis will be the classes, attributes and relationships of the designs. We will see if the ontology in each language can represent these concepts in UML. We will analyse if for each concept it is represented or specified in UML and the cardinality of each of them.

The comparison has been done in the tables present in the Appendix D. These tables contain the comparison done to the languages. The rows represent if a concept is represented or not, specified or not, what cardinality does the concept have and if this concept has the same restrictions in both languages or not. The columns represent the concepts in the ontology in each Web ontology language. These concepts are classes, attributes and relationships. In each cell we can find 1 if the concept fulfils the property indicated in the column, 0 if not, same if it has the same restrictions,more if the analysed language has added more restrictions to the representation, - if it is no applicable and the type of cardinality.

9.2.2 RDF(S) and UML comparison

The comparison between UML and RDF(S) is represented in the Table D.1. The rows represent the UML classes, attributes and relationships between classes. The notation is specified in the Appendix D and here we will use the coded names. In the columns are represented the concepts we are going to analyse: if the ontology in UML is represented in RDF(S) or not, if the concept in UML is specified in RDF(S) or not and if the concept in UML has restrictions or not.

Representation

As we can see in the Table D.1 66.6% the classes that are represented in RDF(S) are not represented in UML. This is due to the expressivity of RDF(S). We have modelled our ontology to provide the maximum expressiveness we can get. In UML we have modelled a database, optimized for SQL queries. The difference between these two approaches is who will do the queries. Within UML the queries will be done by a person who will reason about the relationships we have in the model. Within RDF(S) the queries will be managed by an agent and due to this the classes in RDF(S) should be more richly specified. Therefore we have more classes in the RDF(S) representation to permit a better reasoning.

As we can see in the Table D.1 93.33% attributes in the RDF(S) representation are in the UML representation. The only attributes that are not in the UML representation is due to we are using them as help attributes to fulfil some classes in RDF(S). Almost all the attributes are present because we have the same data in both representations. Here the most important is to maintain the data, the attributes have less expressiveness than the classes.

As we can see in the Table D.1 none of the relationships present in the RDF(S) representation are in the UML representation. This is because we have different classes and to relate them we need different relationships. Also the same reason we gave to explain the differences between the classes representation within RDF(S) and UML can be applies here. We have different reasoning necessities. In UML the queries will be done by a person and in RDF(S) by an agent, therefore we need different ways for representing data.

Specification

As we can see in the Table D.1 11% of the RDF(S) classes are specified in the UML representation. The classes which are not represented can be divided in two groups. The first group of classes which have no representation in UML but in RDF(S) are those classes created to improve the ontological design. These classes are Human or Information, for example. We do not have these classes in UML because we do not need them, but we need them in the ontology. The second group of classes not specified in the UML model but present in the RDF(S) model are those classes which we need for representing special features for reasoning. An example is the VisitCard class. We do not have it in UML but in RDF(S) because we use it to relate all the information needed for a person to locate it. In UML we do not need it because we will do this by queries using the existing relationships.

As we can see in the Table D.1 6.6% of the RDF(S) attributes are specified in UML. This is because almost all attributes are represented and we do not need to specify them. The only attributes that are not represented are the attributes that are specified.

As we can see in the Table D.1 83.3% of the RDF(S) relationships are specified in UML. This is because we have similar relationships in both models. In the RDF(S) model we have almost all these relationship represented but they do not exist in UML. But these relationships are specified in UML. We have similar relationships like hasTitle to indicate a Person can have zero or more Titles. This relationship is represented within UML. No the relationships which are neither specified nor represented are those relationships which relate classes that are not in the UML model but in the RDF(S) model. The relationship between Person and VisitCard is an example. We do not have the relationship in UML because we use different relationships to retrieve the information.

Restrictions

The restrictions are based on the restrictions specified in the DTD document where it is specified the Swedish database. Initially we will maintain these restrictions because we will find them in the data we read and we store. Once we designed the new UML model we translated most of the restrictions to this model but we added more. We added more restrictions to maintain the coherence of the model. We will chose which data is useful and we will chose also which kind of restrictions we want in this data.

As we can see in the Table the Table D.1 in all classes we can apply restrictions in both UML and RDF(S) we can apply the same restrictions. The restrictions specified in RDF(S) are in the UML representation. We can model easily the cardinality restrictions within UML. The restrictions like disjoint are not present in the UML representation. This representation does not offer them. We added these restrictions to groups of classes to facilitate the reasoning of the information agents. These types of restrictions are a particular way for modelling frames. They are not applies in databases design because the databases are not designed for reasoning.

The restrictions in the relationships and in the attributes are the same. This is because relationships and attributes are managed in the same way in RDF(S). For specifying a relationship we have to create an attribute and specify its type. The type will be an instance of a class. Therefore we have a relationship.

The restrictions we can apply to the attributes are a few. Mainly type of the attribute, cardinality, if it is required or initial values. All the attributes have the same restrictions. We can apply this reasoning to the relationships because they are a special kind of attribute.

9.2.3 OWL and UML comparison

The comparison between UML and OWL is represented in the Table D.2. The rows represent the UML classes, attributes and relationships between classes. The notation is specified in the Appendix D and here we will use the coded names. In the columns are represented the concepts we are going to analyse: if the ontology in UML is represented in OWL or not, if the concept in UML is specified in OWL or not and if the concept in UML has restrictions or not.

Representation

We use the same classes as OWL because both offer the same constructs for creating classes. Both languages RDF(S) and OWL are similar when designing the basic structure of an ontology. They provide the same basic constructs because OWL is based on RDF(S) and we will get a similar analysis. As we can see in the Table D.2 66% that are represented in OWL are not represented in UML. This is due to the expressivity of OWL. We have modelled our ontology to provide the maximum expressiveness we can get. In UML we have modelled a database, optimized for SQL queries. The difference between these two approaches is who will do the queries. Within UML the queries will be done by a person who will reason about the relationships we have in the model. Within OWL, like in RDF(S) the queries will be managed by an agent and due to this the classes in OWL should be more richly specified. Therefore we have more classes in the OWL representation to permit a better reasoning.

We have the same representation here then in RDF(S). OWL is an extension of RDF(S) and we have the same data than before. Then, similar attributes will be finding here. As we can see in the Table D.2 93.3% of the attributes in the OWL representation are in the UML representation. The only attributes that are not in the UML representation is due to we are using them as help attributes to fulfil some classes in OWL. Almost all the attributes are present because we have the same data in both representations. Here the most important is to maintain the data, the attributes have less expressiveness than the classes.

Here happens the same than with the classes and attributes. We have to specify the same schema and we have the same constructs to do it. As we can see in the Table D.2 none of the relationships present in the OWL representation are in the UML representation. This is because we have different classes and to relate them we need different relationships. Also the same reason we gave to explain the differences between the classes representation within OWL and UML can be applies here. We have different reasoning necessities. In UML the queries will be done by a person and in OWL by an agent, therefore we need different ways for representing data.

Specification

Here we will have the same concepts specified than in RDF(S). We have the same situation than in the Representation section and we have a similar analysis. As we can see in the Table D.2 11.1% of the OWL classes are specified in the UML representation. The classes which are not represented can be divided in two groups. The first group of classes which have no representation in UML but in OWL are those classes created to improve the ontological design. These classes are Human or Information, for example. We do not have these classes in UML because we do not need them, but we need them in the ontology. The second group of classes not specified in the UML model but present in the OWL model are those classes which we need for representing special features for reasoning. An example is the VisitCard class. We do not have it in UML but in OWL because we use it to relate all the information needed for a person to locate it. In UML we do not need it because we will do this by queries using the existing relationships.

As we can see in the Table D.2 6.6% of the OWL attributes are specified in UML. This is because almost all attributes are represented and we do not need to specify them. The only attributes that are not represented are the attributes that are specified.

As we can see in the Table D.2 83.3% of the OWL relationships are specified in UML. This is because we have similar relationships in both models. In the OWL model we have almost all these relationship represented but they do not exist in UML. But these relationships are specified in UML. We have similar relationships like hasTitle to indicate a Person can have zero or more Titles. This relationship is represented within UML. No the relationships which are neither specified nor represented are those relationships which relate classes that are not in the UML model but in the OWL model. The relationship between Person and VisitCard is an example. We do not have the relationship in UML because we use different relationships to retrieve the information.

Restrictions

The restrictions are based on the restrictions specified in the DTD document where it is specified the Swedish database. Initially we will maintain these restrictions because we will find them in the data we read and we store. Once we designed the new UML model we translated most of the restrictions to this model but we added more. We added more restrictions to maintain the coherence of the model. We will chose which data is useful and we will chose also which kind of restrictions we want in this data.

The restrictions are different than in RDF(S). Here we can put more restrictions because OWL offers more constructs to do it. As we can see in the Table D.2 almost all the restrictions specified in OWL are in the UML representation. We can model easily the cardinality restrictions within UML. The restrictions like disjoint are not present in the UML representation. This representation does not offer them. We added these restrictions to groups of classes to facilitate the reasoning of the information agents. These types of restrictions are a particular way for modelling frames. They are not applied in databases design because the databases are not designed for such reasoning. Restrictions can be mainly applied to classes. Here we can specify first order logic predicates, rules, cardinalities, etc. This is because in the Table D.2 we can see that there are not the same restrictions. We have added more restrictions in OWL. We added first order predicates in classes to restrict what instances can belong to a determinate class. This column is different from the other languages because OWL is the only languages which offers this kind of restrictions.

9.2.4 Topic Maps and UML comparison

The comparison between UML and Topic Maps is represented in the Table D.3. The rows represent the UML classes, attributes and relationships between classes. The notation is specified in the Appendix D and here we will use the coded names. In the columns are represented the concepts we are going to analyse: if the ontology in UML is represented in Topic Maps or not, if the concept in UML is specified in Topic Maps or not and if the concept in UML has restrictions or not.

In the two previous languages the Representation, Specification and cardinality sections were the same. Here we will find again the same question. Topic Maps provides the same constructs as RDF(S) and OWL and we have used the same tool for designing the ontologies. Therefore we will get the same analysis as before in the initial sections.

Representation

As we can see in the Table D.3 33.3% of the classes that are represented in Topic Maps are not represented in UML. This is due to the expressivity of Topic Maps. We have modelled our ontology to provide the maximum expressiveness we can get. In UML we have modelled a database, optimized for SQL queries. The difference between these two approaches is who will do the queries. Within UML the queries will be done by a person who will reason about the relationships we have in the model. Within Topic Maps, like in OWL the queries will be managed by an agent and due to this the classes in Topic Maps should be more richly specified. Therefore we have more classes in the Topic Maps representation to permit a better reasoning.

As we can see in the Table D.3 93.3% of the attributes in the Topic Maps representation are in the UML representation. The only attributes that are not in the UML representation is due to we are using them as help attributes to fulfil some classes in Topic Maps. Almost all the attributes are present because we have the same data in both representations. Here the most important is to maintain the data, the attributes have less expressiveness than the classes.

Here happens the same than with the classes and attributes. We have to specify the same schema and we have the same constructs to do it. As we can see in the Table D.3 none of the relationships present in the Topic Maps representation are in the UML representation. This is because we have different classes and to relate them we need different relationships. Also the same reason we gave to explain the differences between the classes representation within Topic Maps and UML can be applies here. We have different reasoning necessities. In UML the queries will be done by a person and in Topic Maps by an agent, therefore we need different ways for representing data.

Specification

As we can see in the Table D.3 11.1% of the Topic Maps classes are specified in the UML representation. The classes which are not represented can be divided in two groups. The first group of classes which have no representation in UML but in Topic Maps are those classes created to improve the ontological design. These classes are Human or Information, for example. We do not have these classes in UML because we do not need them, but we need them in the ontology. The second group of classes not specified in the UML model but present in the Topic Maps model are those classes which we need for representing special features for reasoning. An example is the VisitCard class. We do not have it in UML but in Topic Maps because we use it to relate all the information needed for a person to locate it. In UML we do not need it because we will do this by queries using the existing relationships.

As we can see in the Table D.3 6.6% of the Topic Maps attributes are specified in UML. This is because almost all attributes are represented and we do not need to specify them. The only attributes that are not represented are the attributes that are specified.

As we can see in the Table D.3 83.3% of the Topic Maps relationships are specified in UML. This is because we have similar relationships in both models. In the Topic Maps model we have almost all these relationship represented but they do not exist in UML. But these relationships are specified in UML. We have similar relationships like hasTitle to indicate a Person can have zero or more Titles. This relationship is represented within UML. No the relationships which are neither specified nor represented are those relationships which relate classes that are not in the UML model but in the Topic Maps model. The relationship between Person and VisitCard is an example. We do not have the relationship in UML because we use different relationships to retrieve the information.

Restrictions

The restrictions are based on the restrictions specified in the DTD document where it is specified the Swedish database. Initially we will maintain these restrictions because we will find them in the data we read and we store. Once we designed the new UML model we translated most of the restrictions to this model but we added more. We added more restrictions to maintain the coherence of the model. We will chose which data is useful and we will chose also which kind of restrictions we want in this data.

As we can see in the Table the Table D.1 almost all the restrictions specified in Topic Maps are in the UML representation. We can model easily the cardinality restrictions within UML. The restrictions like disjoint are not present in the UML representation. This representation does not offer them. We added these restrictions to groups of classes to facilitate the reasoning of the information agents. These types of restrictions are a particular way for modelling frames. They are not applies in databases design because the databases are not designed for reasoning. Here we can apply similar restrictions as in RDF(S). This language does not offer logic predicates to specify restrictions and hence we are in the same level as RDF(S). We see in the Table D.3 we have the same restrictions as in UML or RDF(S).

We have seen in this section that the languages in their basic functions are quite similar. The main difference between them are the tools for creating restrictions they have. We can see many differences with the UML representation. There are classes that are not needed in the UML representation, attributes better represented as classes in the ontological format and different relationships in both representations. One thing is common for all, the basic concepts are represented or specified in both schemas.

9.3 Concluding Remarks

We have seen in this section the analysis of the representation of our languages compared to UML. We have to conclude after this chapter OWL is the best language for representing our domain problem. It offers more constructs for specifying restrictions than the others. We can observe after the analysis that in the other sections like representing classes or attributes all languages are quite similar. This is because they offer the basic elements to model our domain and these elements are present in all three languages. The main difference between all three Web ontology languages are the restrictions that can be applied.

Part IV

Conclusions and Appendixes

Chapter 10

Conclusions and future work

In this chapter, we are going to present the extracted conclusions from the research process done and the future work recommended. In the conclusions we will describe the results of our analysis of the case study and in the future work we will present the possible ways for improving and continuing the research.

10.1 Conclusions

In this thesis we have analysed the suitability of the semantic Web technologies for a concrete domain. These technologies have demonstrated a very good performance and they suppose a good technology for using in knowledge representation problems.

This analysis will provide to Invenio a clear view of the most suitable technology to solve its case. The analysis has covered the necessities of Invenio for directory services and it provides clear theoretical and practical recommendations.

This analysis can be used for analysing the requirements of this case study can be applied to others. The way for analysing the suitability of the semantic Web languages can be applied to other case studies by using the same tools and framework. It will provide a recommendation of the most suitable technology if it is needed.

The technical conclusions we can obtain from the research process are the following. First, the design of the databases is a primary concept. We need a clear view of what the database is representing. Without this view it is very difficult to create the appropriate ontology, even if we only want the data. Second, without data it is not possible to get proper results. We have thousands of instances but instances from the same group of classes, like person or street. An agent can reason about this data but without relationships and more classes can not get the best performance. Third, the evaluation of the languages gives a clear winner, OWL. It offers more constructs to design a good ontology and elements to reason about. The other two languages offer fewer constructs and are not oriented to enterprise solutions. Also, with OWL-S we can extend our services and be more competitive. Fourth, the tool for storing and accessing the data in the ontological format is another key concept. It makes a huge difference when you have to load a database in memory and to consult this data with speed. If you do not want to use an information agent to consult the tool will be a key concept. Fifth, conversion parsers are just translators, but they are necessary. They can be less accurate than other things but it is needed to use a large amount of time developing them.

10.2 Future Work

One important thing to improve is the tools used in this study. We have used Protégé 3.0 for OWL, RDF(S) and the design of Topic Maps and Ontopia Omnigator Eight for accessing the data in Topic Maps. These tools offer enough functionalities with a certain amount of data but if we want to load thousands of instances of each class we will need a more powerful tool. For example, when we want to load an OWL project the told and the computer need time and in the same way if we want to save our project. With Ontopia Omnigator we had problems when browsing the data. If there are so many instances and we want to access to the master index we will not see al the instances, only a few of them.

The next step in this project is to extend the ontology by adding more relationships and restrictions. This ontology has been based on the data available and can be extended by adding new constructs to it. These constructs should be filled in by new data. The ontology is prepared to add this data in the case of companies and persons and it can be done in more cases if the data is available.

One of the most interesting objectives for a future is to offer a service with the directory service of the two countries. This can be done by using OWL-S, the semantic Web services extension for OWL, and only will be needed to develop these services. The core ontology is already created. In a more exhaustive analysis of the case study the evaluation can be refined by adding to the analysis Web Ontology Languages such DAML+OIL or the new version of the Web Ontology Language OWL-S or analyzing more ontology creation tools. These improvements will create a more concrete evaluation of the case study and will provide better results, but also will require more time.

The results obtained with the evaluation framework can be improved by using a better by going into more detail when evaluating. The results obtained in the Table 6.7 are close between them but are more differences between the three languages selected.

Bibliography

[1] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt, *Automatic ontology-based knowledge extraction from web documents*, IEEE Intelligent Systems **18** (2003), no. 1, 14–21.

[2] D. Austin, A. Barbir, N. Networks, C. Ferris, and S. Garg (eds.), *Web services architecture requirements*, http://www.w3.org/TR/wsa-reqs/, W3C, February 2004.

[3] J. Barrasa, O. Corcho, and A. Gómez-Pérez, *Fund finder wrapper: A case study of database-to-ontology mapping*, October 2003, ISWC2003 Workshop on Semantic Integration, Sanibel Island, Florida.

[4] T. Berners-Lee, J. Handler, and O. Lassila, *The semantic web*, Scientific American (2001), 35–43.

[5] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard (eds.), *Web services architecture*, http://www.w3.org/TR/ws-arch/, W3C, February 2004.

[6] D. Brickley and R.V. Guha (eds.), *Rdf vocabulary description language 1.0: Rdf schema*, Brian McBride, Hewlett-Packard Laboratories, http://www.w3.org/TR/rdf-schema/, W3C, February 2004.

[7] N.R. Brisaboa, M.R. Penabad, A.S. Places, and F.J. Rodriguez, *Ontologies for database federation*, UPGRADE - cepis-upgrade.org (2002).

[8] C. Buil-Aranda, *Development of semantic web solutions for directory services*, December 2004, Autumn project at IDI, Norwegian University of Science and Technology.

[9] M. Dean and G. Schreiber (eds.), *Owl web ontology language reference*, http://www.w3.org/TR/owl-ref/, W3C, February 2004.

[10] C. B. Freie, *D2r map – a database to rdf mapping language*, 2003.

[11] T.R. Gruber, *A translation approach to portable ontology specifications*, Knowledge Acquisition (1993).

[12] S. Hakkarainen, D. Strasunskas, L. Hella, and S. Tuxen, *Weighted evaluation of web-based ontology building method guidelines*, The 24th International Conference on Conceptual Modeling (ER2005), Springer Verlag, 2005, To appear.

[13] S. E. Hakkarainen, A. Kofod-Petersen, and C. Buil-Aranda, *Situated support for choice of representation for a semanticweb application*, September 2005, In the IEEE/WIC/ACM International Conference on Web Intelligence, To appear.

[14] J. Hendler, T. Berners-Lee, and E. Miller, *Integrating applications on the semantic web*, Journal of the Institute of Electrical Engineers of Japan **122(10)** (2002), 676–680.

[15] J. Hjelm, *Creating the semantic web with rdf: Professional developer's guide*, John Wiley & Sons, Inc., New York, NY, USA, 2001.

[16] M. Horridge, *A practical guide to building owl ontologies with the protege-owl plugin and co-ode tools 1.0*, The University of Manchester, August 2005.

[17] H. Knublauch, *Protege owl plugin*, Stanford Medical Informatics, http://protege.stanford.edu/plugins/owl/index.html.

[18] J. Korhonen and P. Isto, *Practical experiences in developing ontology-based multi-agent system*, Business Information Systems, Proceedings of BIS 2003 (Gary Klein Witold Abramowicz, ed.), 2003.

[19] J. Krogstie and A. Solvberg, *Information systems engineering - conceptual modeling in a quality perspective*, Norwegian University of Science and Technology, Trondheim, Norway (1999).

[20] F. Manola and E. Miller (eds.), *RDF Primer, W3C Recommendation*, W3C, 2004.

[21] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *Owl-s: Semantic markup for web services*, http://www.w3.org/Submission/OWL-S/, Nobember 2004.

[22] N. Mitra (ed.), *Soap version 1.2 part 0: Primer*, http://www.w3.org/TR/soap12-part0/, W3C, June 2003.

[23] V. Morocho and L. Perez-Vidal, *Database schema detection and mapping on mobile applications: An ontology-based approach.*

[24] N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen, *Creating semantic web contents with protégé-2000*, IEEE Intelligent Systems **16** (2001), no. 2, 60–71.

[25] Ontoknowledge.org, *Description of oil*, http://www.ontoknowledge.org/oil/.

[26] S. B. Palmer, *The semantic web: An introduction*, http://infomesh.net/2001/swintro/, September 2001.

[27] J. Z. Pan and I. Horrocks, *Metamodeling architecture of web ontology languages*, International Semantic Web Working Symposium (SWWS) (California, USA), Stanford University, July 2001, pp. 131–149.

[28] P. F. Patel-Schneider and D. Fensel, *Layering the semantic web: Problems and directions*, ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web (London, UK), Springer-Verlag, 2002, pp. 16–29.

[29] S. Pepper and G. Moore (eds.), *Xml topic maps (xtm) 1.0 specification*, http://www.topicmaps.org/xtm/1.0/, International Organization for Standardization ISO, TopicMaps.org, 2001.

[30] S. A. Petersen, *Web services: Architectures and standards*, Trial Lecture, November 2003, NTNU.

[31] S. Powers, *Practical rdf*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[32] C. Robson, *Real world research*, Blackwell Publishers, Oxford, UK ; Cambridge, Mass., USA, 2002.

[33] M. K. Smith, C. Welty, and D. L. McGuinness, *Owl web ontology language guide*, http://www.w3.org/TR/owl-guide/, February 2004, W3C Recommendation.

[34] L. Stojanovic, N. Stojanovic, and R. Volz, *Migrating data-intensive web sites into the semantic web*, SAC '02: Proceedings of the 2002 ACM symposium on Applied computing (New York, NY, USA), ACM Press, 2002, pp. 1100–1107.

[35] X. Su and L. Ilebrekke, *A comparative study of ontology languages and tools*, CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering (London, UK), Springer-Verlag, 2002, pp. 761–765.

[36] IBM T. Bellwood (ed.), *Uddi version 2.04 api specification*, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, UDDI Committee Specification, July 2002.

[37] M. Uschold and M. Gruninger, *Ontologies: Principles, methods and applications*, Knowledge Engineering Review **11** (1996), no. 2.

[38] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, *Ontology-based integration of information a survey of existing approaches*, IJCAI–01 Workshop: Ontologies and Information Sharing (2001).

Appendix A

(Semantic) Web Services

In this chapter, a brief description of Semantic Web Services is provided. It is also shown what a Web Service is, what structure doe the Web Service have and what are the recommendations that W3C provides for building Web Services.

A.1 Web Services

In the W3C Web page it is possible to see the following definition of Web Service from the Web Services working group:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [2].

Web Services constitute a new model for using the Web which allows the publishing of business functions to the Web and enables universal access to these functions[30]. The main functions that a Web Service should support are Web Service Creation, Description, Publishing, Discovery, Invocation and Unpublishing. The services offered to the customer via the Web page of the company offers the business processes to the customer. In these main tasks the human interaction is required.

A.1.1 Web Service Layers

The Web Services Architecture is composed by four layers and they can be seen in Picture A.1 [1]. These layers are the Processes Layer (the most important technology is the Universal Description, Discovery and Integration, UDDI), the descriptions layer (the most important technology related to this layer is the Web Services Description Languages, WSDL), the Messages layer (the most important technology related to this layer is the Simple Object Access Protocol, SOAP) and the layer communication built with Web Standards.

[1]http://www.w3.org/TR/2003/WD-ws-arch-20030808/

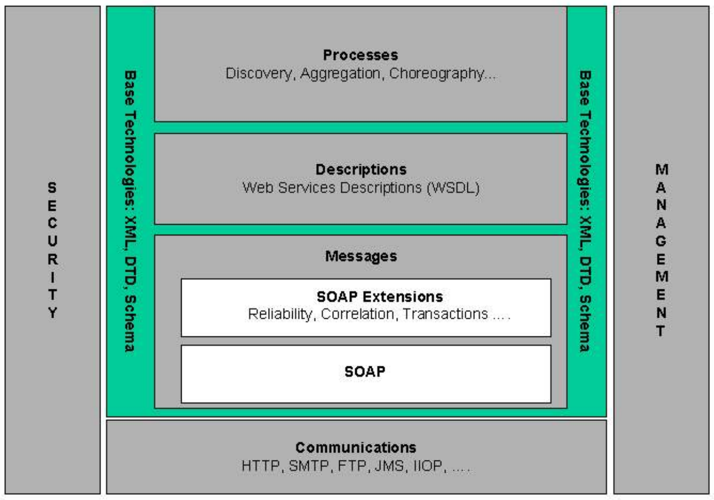

S E C U R I T Y
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
M A N A G E M E N T
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure A.1: Web Services Architecture Layers

WSDL is the web Service Description Language and its function is to describe a Web Service. Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered. The main concept of WSDL is that WSDL divides the description into two different sections, one abstract section and another "real" section. At the abstract level WSDL describes the messages that a Web Service sends and receives using these messages for characterize the service and At a concrete level, a binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. And finally, a service groups together endpoints that implement a common interface.

SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information [22]. Provides the way to send and receive messages between applications.

Universal Description, Discovery and Integration, or UDDI, is the name of a group of web-based registries that expose information about a business or other entity and its technical interfaces (or API's)[36]. UDDI is used for service discovery and defines a common means to publish information about businesses and services. UDDI specification consists of an XML schema for SOAP messages and a description of the UDDI API's specification.

The W3C consortium has defined the main requirements that a Web Service should achieve. These requirements are accessible at [2]:

- Interoperability: The WSA should enable the development of interoperable

Web services across a wide array of environments.

- Reliability: The WSA must be reliable and stable over time. " Integration with the World Wide Web The WSA must be consistent with the current and future evolution of the World Wide Web.
- Integration with the World Wide Web: The WSA must be consistent with the current and future evolution of the World Wide Web.
- Security: The WSA must provide a secure environment for online processes.
- Scalability and Extensibility: The WSA must enable implementations that are scalable and extensible.
- Team Goals: The Web Services Architecture Working Group will work to ensure that the Architecture will meet the needs of the user community.
- Management and Provisioning: The standard reference architecture for Web Services must provide for a manageable, accountable environment for Web Services operations.

A.1.2 Web Service Architecture

The Web Services Architecture model is specified in [5] and it contains four sub-architecture models. The main model is the following show in the Picture A.2 which indicates the main concepts and the relationships between them.

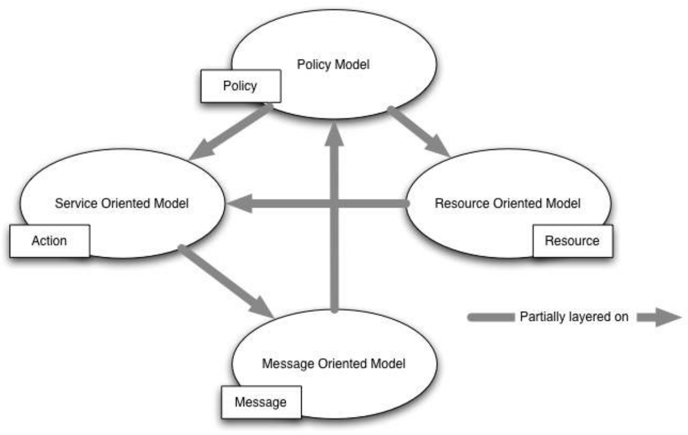


Figure A.2: Meta Model of the Web Services Architecture

The Message Oriented Model main task is to manage the messages, message structure, message transport and so on. This model it is not interested in any other question different. The key concepts are the agent who sends and receives the messages and the structure of the messages.

The Service Oriented Model main task is to manage the service. The model is the means of messages and use to be an agent. One agent on the receiver's side and other agent on the provider's side.

The Resource Oriented Model main task is to manage the resources that exist and have owners.

The Policy Model main task is to manage the policies of access and constraints on the behavior of agents and services. This is the security level of the services, to grant the security of both systems.

A.1.3 Composition Cycle of Web Services

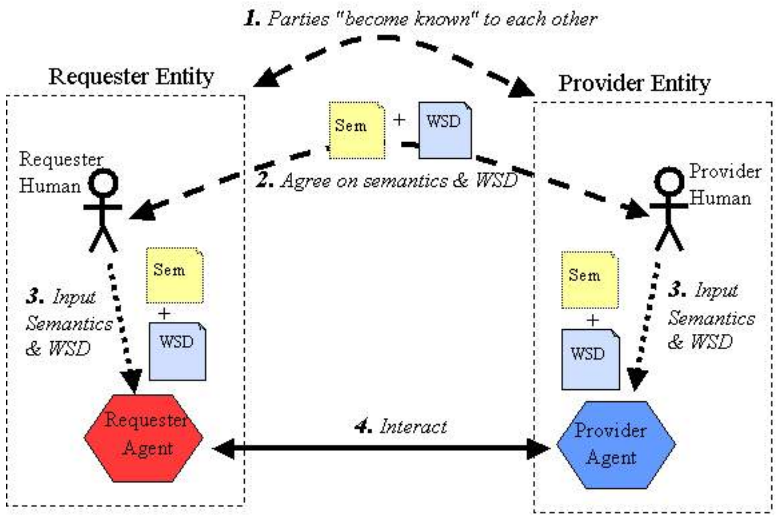


Figure A.3: The General Process of Engaging a Web Service

A Web service use to need three parties: the requester of the service (a client), one or more provider (the server) of the service and a registry that supports the transactions. The Figure A.3shows how the Web service is executed.

First both, the requester and the provider must become known to each other and second the interaction among both will happen. The provider should first advertise itself in the registry and then it is available for the requesters. The provider and the requester should be agree about the semantics and the service description and when they are agree in the way the communication will be the actors start to exchange messages in order to perform the service.

A.2 Semantic Web Services

As we have seen in the previous section Web Services are program-to-program interaction that offers dynamic integration contents but in the main activities that a Web Service should support human interaction is required. Also, Web

Services architecture does not provide any mechanism to extract the meaning of the messages that have been exchanged by the Services involved in the transaction. Web Services architecture provides the technology for understanding the messages but not the content of these messages. With Semantic Web Services these lacks will be fixed.

Semantic Web Services are Web services with semantic information added. With Semantic Web Services we join together two of the most important technologies in the actual Web, the Web Services explained in the previous section and the Semantic Web, the next generation of Web, explained in Chapter 2.

A.2.1 Semantic Web Services Language Requirements

The SWSI language Committee has elaborated a list of requirements that a Semantic Web Language needs to support the Semantic Web Services properties. In the next lines a brief description of these requirements or recommendations is presented. The following recommendations are present in [21] that is still a draft waiting for the final approval.

The committee specifies two different requirements: functional requirements and formalization. Functional requirements are these requirements that are needed to support the interaction between different Web Services. Also, the functional requirements are divided in three sections:

- Advertising and matchmarking: A key concept for the use of a Service is to discover this service. If the machine does not know anything about the service this service is useless. The Semantic Web Language should allow to publish and advertising this service in order to make it available for other services. Also, the description should include the goals of the service, both in the client side and in the server side. This is for not produce mistakes between the services, to make sure that the services will get what they need.

- Negotiation and contracting: The process of matchmarking could not be enough and will not guarantee that the client can use the service. Several problems like the customer needs certain type of requirements in order to complete the transaction may appear. To solve this problem it is needed to make sure that the client can indeed use the service and this is the purpose of the negotiation ad contracting requirements. The contracts may require behavioral, process-like aspects which should describe what the service will do in response to certain actions.

- Process modeling: One of the key concepts within Web Services is the composition of several Web Services inn order to create a new service. The Semantic Web Service Language should provide elements to facilitate this process modeling, the construction of a new service based in others.

- Process enactment: The client must be able to monitor what is happening when the process is running, if the process will finish in a successful way and if will return the desired solution. The language should provide elements to track the services.

Functional requirements are intended to help identify the formalisms that can adequately support the functional requirements. Also, some requirements are needed like general language requirements and process modeling requirements.

Appendix B

The Semiotic Quality Framework

In order to evaluate the Web representation languages, the Semiotic Quality Framework (SQF) [19], [35], a model quality framework consisting of five semiotic factors of quality modelling languages is chosen. The framework has three main characteristics that make it well-suited as an evaluation instrument 1) it distinguish between goals and means separating what to achieve from how, 2) it is closely related to linguistics and semiotic concepts, and 3) it is based on a constructivist world-view, the framework recognizes that models are build from interaction between the designer and the user. The main model of the semiotic quality framework is as follows.

***A* - Audience** refer to the individual, A_i, organisational, A_s, and technical actors, A_t who relate to the model. This includes both human participants and artificial actors.

***K* - Participant knowledge** is the explicit knowledge that is relevant for the audience A. This is the combined knowledge of all participants in the project.

***L* - Language extension** is what can be represented according to the graphical symbols, vocabulary and syntax of the language; the set of all statements that may be informal A_i, semi-formal A_s, or formal A_f.

***M* - Model externalization** is the set of all statements in an actorŠs model of a part of a perceived reality written in a language L.

***I* - Social actor interpretation** is the set of all statements which the externalised model consists of, as perceived by the social audience A_i and A_s.

***T* - Technical actor interpretation** is all the statements in the conceptual model L as they are interpreted by the technical audience A_t.

***D* - Modelling domain** is the set of all statements that can be stated about a particular situation.

The framework evaluates the physical, empirical, syntactic, semantic, pragmatic, perceived semantics, social and knowledge quality; it evaluates the quality of conceptual models, modelling environments, and modelling languages. This work focuses on the evaluation of the Web representations as modelling languages.

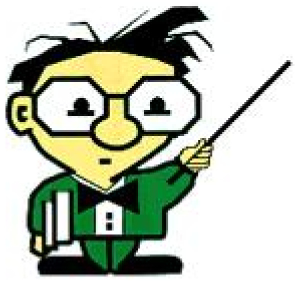
B.1 Adapted appropriateness of languages

The Semiotic Quality Framework consists of five quality factors, called appropriateness namely, Domain Appropriateness (DA), Participant Knowledge Appropriateness (PAK), Knowledge Externalizability Appropriateness (KEA), Comprehensibility Appropriateness (CA), and Technical Actor Interpretation Appropriateness (TAIA). Here we modify the DA as in [14], as follows. DA covers seven perspectives for languages: 1) Structural Perspective refers to the static structure, classes and properties, 2) Functional Perspective refers to the processes, activities, and transformations, 3) Behavioural Perspective refers to the states and transitions between them, 4) Rule Perspective refers to the rules for certain processes, activities, and entities, 5) Object Perspective refers to the resources, processes and classes, 6) Communication Perspective refers to the language actions, meaning and agreements, and 7) Actor and Role Perspective refers to the actor, role, society and organisation.

With the modification of the DA we acquire the elements needed for analysing the most practical features of the languages. With the PKA we measure the knowledge of the user. With the KEA we analyse if the language provides enough elements to represent the domain model specified. With CA we analyse if the language is consistent enough and provides clear elements for modelling the domain, and with TAIA we analyse if the language provides enough features for allowing automatic reasoning, the key concept in our investigation. The quality factors will be further developed in the next section.

Appendix C

The Tool: Protégé 2000



The tool used to design the different ontologies in the three languages mentioned in the previous chapters is Protégé 2000. This tool enables the user the creation of ontologies and also can be used as knowledge-based editor. Protégé is developed by the Stanford Medical Informatics at the Stanford University School of Medicine.

The tool is available at http://protege.stanford.edu as free software under the open source Mozilla Public License. Also, you can download various plugging that will enable Protégé to develop different kinds of projects. Protégé 2000 has been developed in Java.

C.1 Why Protégé 2000

We have selected the tool Protégé 2000 partly due to it is very easy to use. This is mainly because of its screen interface and also because it is highly configurable and you can download many plug-in from the Protégé web site. For example, Protégé 2000 easily supports editing all the three languages that are part of the study.

The easy of use originates from the graphic interface. One you run Protégé 2000 you will se the initial screen that offers you the possibility of creating a new project, Figure C.1 (also you can select the format to save this project RDF(S), OWL, database or just plain text) or open a previously created project. This screen doesn't have any particular option but once initiated the tool we will see that it can offer many options, also we will be capable of creating our own ontologies and administrating knowledge-based systems.

Once Protégé 2000 is initiated with a new project or with an existing project we can see a tab division of Protégé that includes a Class tab, Slots Tab, Forms Tab, Instances Tab and Queries Tab.

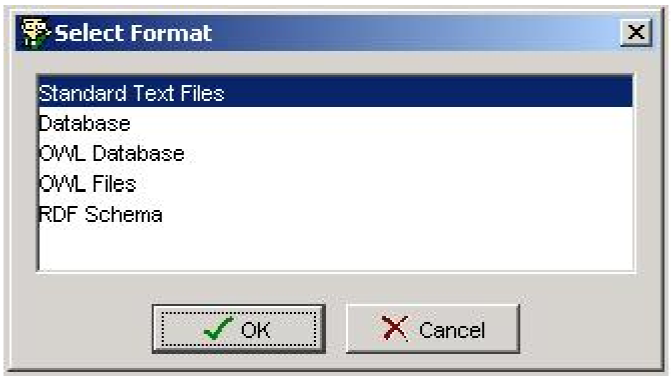


Figure C.1: Create Project

C.1.1 Class Tab

Class Tab The process of create a class it is easy as illustrated in Figure C.2. You only have to click in the super class THING (on the left of the screen) and the click the C button situated on the left of the screen, on top of the classes screen. Then we will see on the right of the screen that a new screen is open and we will proceed to full the properties of the class.

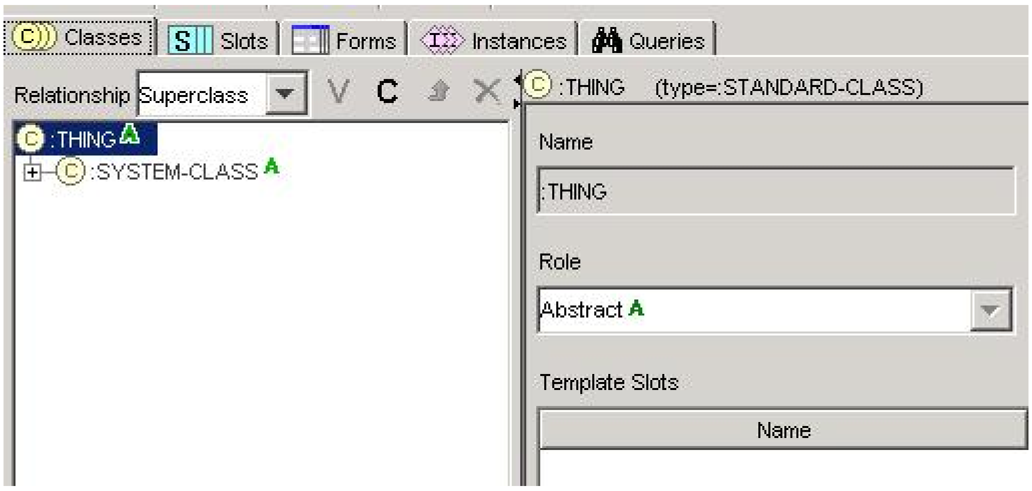


Figure C.2: Classes Tab

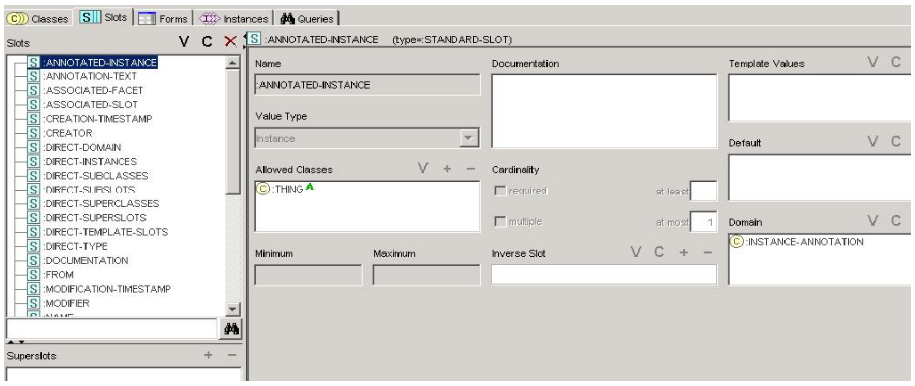
It is also possible create classes with inheritance from a previously created class from the THING metaclass. The process is the same as create a class from THING. The super classes of one class are indicated in a dialog situated on the left down of the main screen.

The easy of creation classes is extended to the modification of the relations created because if it is necessary to make any modification to an inheritance relationship the only thing needed to make that is to drag the class to the class that will its parent, for example THING.

It is also possible to add to the class's information about them like some documentation, constraints or assign a different role. To make that the only necessary is to check the dialogs on the top right of the main screen having selected one class.

C.1.2 Slots Tab

In order to create a slot the same process as the used for creating a class is used. Once selected the class in the classes tab it is only necessary to click on the C button on the left screen of the main window of Protégé 2000. Then it will appear a screen with the attributes of the slot as depicted in Figure C.3.



:ANNOTATED-INSTANCE (type=:STANDARD-SLOT)

Slots:
:ANNOTATED-INSTANCE
:ANNOTATION-TEXT
:ASSOCIATED-FACET
:ASSOCIATED-SLOT
:CREATION-TIMESTAMP
:CREATOR
:DIRECT-DOMAIN
:DIRECT-INSTANCES
:DIRECT-SUBCLASSES
:DIRECT-SUBSLOTS
:DIRECT-SUPERCLASSES
:DIRECT-SUPERSLOTS
:DIRECT-TEMPLATE-SLOTS
:DIRECT-TYPE
:DOCUMENTATION
:FROM
:MODIFICATION-TIMESTAMP
:MODIFIER

Name: :ANNOTATED-INSTANCE
Value Type: Instance
Allowed Classes: :THING
at least:
at most: 1
Domain: :INSTANCE-ANNOTATION

Figure C.3: Slots Tab

There is also a tab on the right of the classes tab and there is a list of all slots created. You can also create the attributes there and assign super slots of one attribute like with the classes, the domain where the slot will be applied. We can add facets into the slots for expressing information about those slots.

C.1.3 Forms Tab

When the Form tab is selected it is possible to see on the left of the screen the list of classes that have been created with a different icon. When a class is selected it is possible to see on the right of the screen the picture that represents the fields of the slot that had been created before. It is possible to select each field and modify its different visual properties and put them into the best place.

C.1.4 Instances Tab

In the instances tab Protégé 2000 allow us to create instances from the classes created before. For creating instances it is only necessary to select the main class of the instance of it and then click the C button on the middle of the screen.

Once the instance is created it is possible to fill in the different slots of the class, also creating new instances if one slot is an instance. Then a new window will be opened with the slots of the new instance. The new instance will be stored in the instances screen.

C.1.5 Queries Tab

In the last tab of Protégé 2000 we can make queries to the knowledge base created. It is possible to access some fields for configuring the query. It is possible to specify the class, the slots and some properties. The result will be shown on the results screen, on the left of the screen. The queries also can be stored in a database of queries. An example of this feature is shown in Figure C.4.

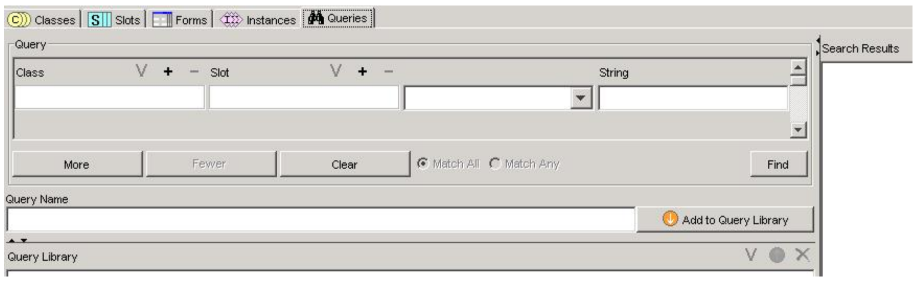


Figure C.4: Queries Tab

C.2 Core of Protégé 2000

We have seen the basics functionalities of Protégé 2000, creating our knowledge model from the initial constructs of Protégé but it is also possible to extend these basic constructs for generating out own metaclasses or our own metaslots.

Metaclasses are templates for classes in the same way that classes are templates for instances [24]. In order to define our own metaclasses we have to create a new class from the abstract class META-CLASS and define the properties that the new class should have. To create metaslots the same process is needed, just select the META-CLASS and create a new class for the new slot template.

For creating a new editor for our language it is necessary to define the new kinds of classes and attributes in Protégé. In order to do that there are some general considerations about the similarities and differences between the tool and the new language. Four categories are identified [24].

- Concepts that are exactly the same in the two languages
- Concepts that are the same but expressed differently in the two languages
- Concepts in the language of choice that do not have an equivalent in Protégé 2000
- Concepts that Protégé 2000 support and the language of choice does not.

Depending of the concepts of the language that we will define in Protégé we should create the elements in the tool. When it is necessary to save the work Protégé will save it in the Protégé format, but if one wants to save the work in another format one should to create a back-end plug-in. An example of back-end plug-gin is the TM-Tab plug-in used for creating of the ontology in Topic Maps. This plug-in adds one tab for extracting the ontology created with Protégé to the XTM format.

C.3 Protégé 2000 Plug-in

You can create plug-in using the steps specified before, but there are many plug-in created for users. These plug-in are available at the Protégé Web site and are classified in the Topics related. The examples of plug-in are:

OWL Support Load, save, and edit Web Ontology Language (OWL) ontologies in Protégé.

ezOWL Tab Widget Visual OWL (Web Ontology Language) editor for Protégé.

RDF Storage Backend Create, import, and save RDF(S) files in Protégé.

Dublin Core Representation of Dublin Core metadata in Protégé.

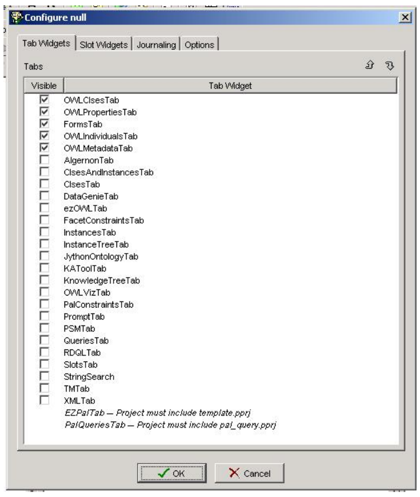
DAML+OIL Storage Backend Create and edit DAML+OIL ontologies with Protégé.

UML Storage Backend Store a Protégé knowledge base in UML.

XMI Storage Backend Store a Protégé knowledge base as XMI files. XMI is a standard format for metadata exchange supported by OMG, the group that is responsible for standards such as UML, CORBA and the Common Warehouse Metamodel.

UMLS Tab Widget Search the Unified Medical Language System (UMLS) and annotate your current Protégé ontology with terms, concept ids, synonyms, relations, and other information from UMLS.

In order to use any of the Protégé 2000 Plugging you have to go activate it in the screen File-Configure. Figure C.5 shows the Plug-in selection screen.



Visible	Tab Widget
✓	OWLClsesTab
✓	OWLPropertiesTab
✓	FormsTab
✓	OWLIndividualsTab
✓	OWLMetadataTab
☐	AlgernonTab
☐	ClsesAndInstancesTab
☐	ClsesTab
☐	DataGenieTab
☐	ezOWLTab
☐	FacetConstraintsTab
☐	InstancesTab
☐	InstanceTreeTab
☐	JythonOntologyTab
☐	KAToolTab
☐	KnowledgeTreeTab
☐	OWLVizTab
☐	PalConstraintsTab
☐	PromptTab
☐	PSMTab
☐	QueriesTab
☐	RDQLTab
☐	SlotsTab
☐	StringSearch
☐	TMTab
☐	XMLTab
	EZPalTab — Project must include template.pprj
	PalQueriesTab — Project must include pal_query.pprj

Figure C.5: Plugging activation

C.3.1 OWL Plugging

OWL plug-in enables the user to create easily with Protégé 2000 ontologies in the Ontology Web Language. The plug-in can be downloaded from http://protege.stanford.edu/plugins/owl/ and offer many features. The Protégé 2000 OWL plug-in enables you to[17]:

- Load and save OWL and RDF ontologies
- Edit and visualize OWL classes and their properties
- Define logical class characteristics as OWL expressions
- Execute reasoners such as description logic classifiers and
- Edit OWL individuals for Semantic Web markup

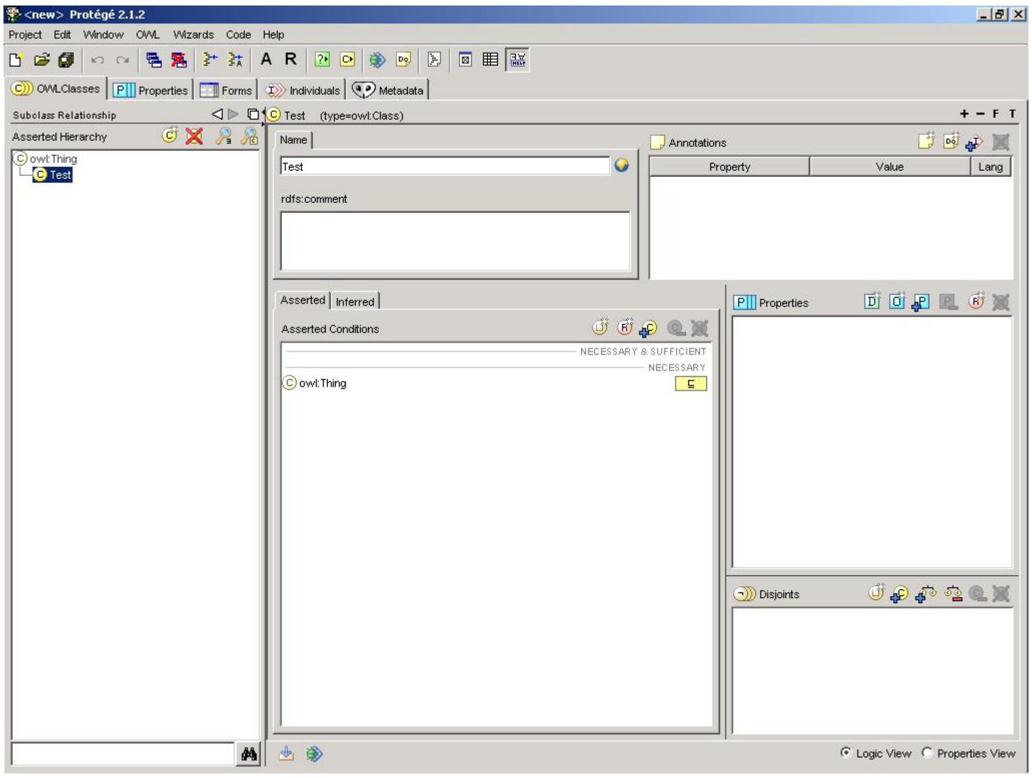


Figure C.6: OWL Plugging for Protege

In the Figure C.6 the main screen of Protégé 2000 with OWL is shown. We can see more features than in the original screen of Protégé 2000. These features are concentrated in the classes tab. In the middle section, we can create new restrictions in the selected class domain. There are two kinds of restrictions, Necessary and Sufficient and Necessary. We can select one of these two and click the R button for creating one new restriction or C button to add one restriction previously created. The restrictions will be applied to the properties of the class and the plug-in offers one editor in order to create them. With this editor we can create easily restrictions to our properties. The editor allows us to navigate between classes or properties and create new properties if necessary. We can also create the existential or universal restrictions pushing a button. The restriction that has been created is parsed in every moment.

In the OWL section in the main menu we can access to more OWL features. It is possible to check which OWL sublanguage (Lite, DL or Full) are we using in the ontology, run ontology test to check if our ontology language is correct, configure the parameters of the test, check the consistency and so forth.

Other new feature is in the properties tab, we can create object properties for creating relationships between classes. Also, in the plug-in there is a new tab, Metadata Tab that will allow us to specify the namespaces used in our ontology.

C.3.2 ezOWL Plug-in

ezOWL Plug-in is a Visual OWL (Web Ontology Language) Editor for Protégé 2000. This plug-in enables the user to create graphically ontologies with OWL. In combination with the OWL extension for Protégé 2000 we can create easily the ontologies. The plug-in can be downloaded from http://iweb.etri.re.kr/ezowl/index.html.

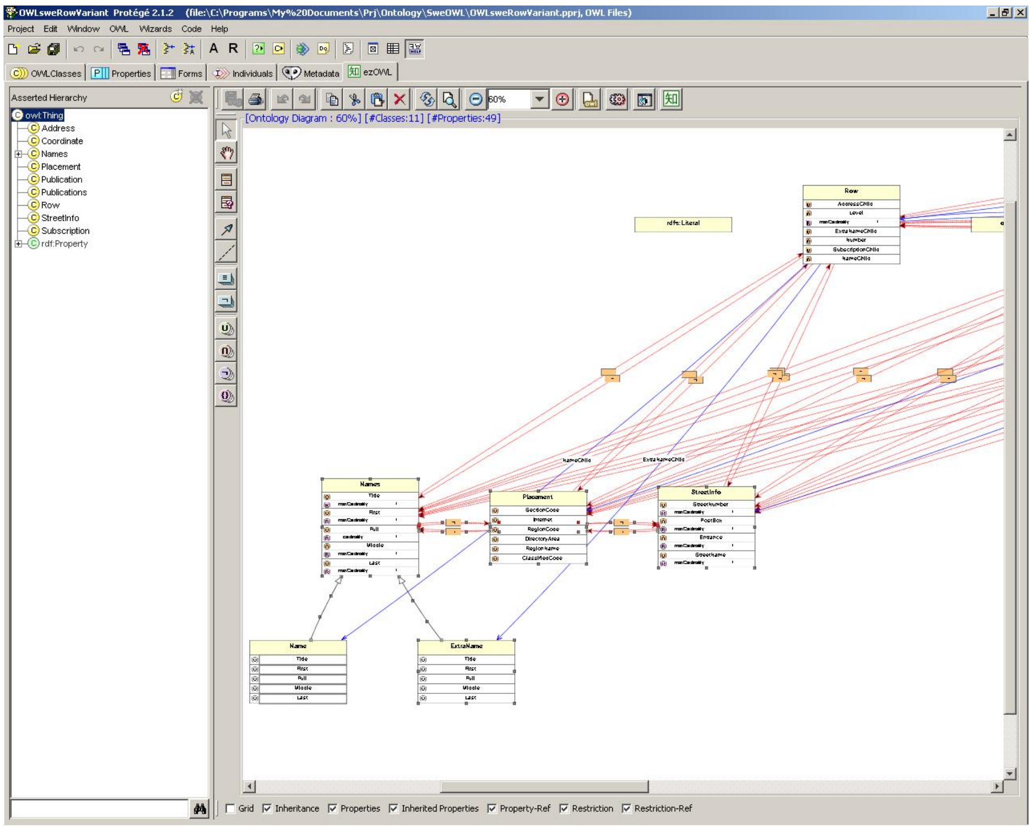


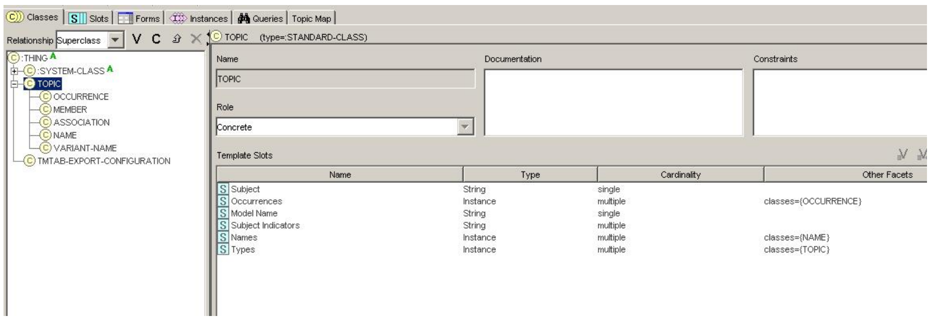
Figure C.7: ezOWL Plug-in for Protégé 2000

In the Figure C.7 we can see how an ontology is developed. On the right section of the screen we can access to all the classes that have been crated and put them into the diagram by dragging them to the design space. We can create descriptions, relations, slots or restrictions in these classes by clicking one of the buttons situated on the left of the design space.

We can adjust the size of the screen, redraw the design or move the classes to the best position but it is disordered. If the ontology has more than ten classes with restrictions and object properties between these classes the design screen will be full of lines and it will be difficult to navigate in the design.

C.3.3 TMTab Plug-in

TMTab is a plug-in for Protégé that allows you to build an ontology which may be exported as a topic map in XTM syntax. This plug-in is accessible on the web page http://www.techquila.com/tmtab/index.html.



TOPIC (type=:STANDARD-CLASS)

Name: TOPIC

Documentation

Constraints

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
Subject	String	single	
Occurrences	Instance	multiple	classes={OCCURRENCE}
Model Name	String	single	
Subject Indicators	String	multiple	
Names	Instance	multiple	classes={NAME}
Types	Instance	multiple	classes={TOPIC}

Figure C.8: TMTab Plug-in for Protégé 2000

As it is shown in the Figure C.8 there are only a few new characteristics in Protégé but we can design our Topic Maps ontology easily. The web page of TM Tab contains a brief description of the plug-in with a short tutorial for starting to create your own ontology.

In the plug-in you can create all the elements of a Topic Map by creating subclasses from Topic. To export the project to a XTM file format it is necessary to access the TMTab and choose the export filename and the configuration of the project. Figure C.9 shows the tab.

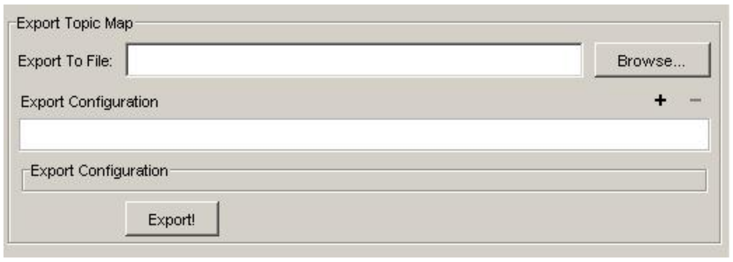


Figure C.9: TMTab Plug-in for Protégé 2000

C.4 Concluding Remarks

In this chapter, we have studied the tool for creating ontologies and knowledge-based editor. This tool offer many functionalities, from the basics like class creation or slots creation to most advanced features like defining the constructs in order to create our own language in order to model it in Protégé.

We can download from the Protégé Web site several plug-in created in diverse fields of knowledge management like the Protégé Axiom Language (PAL) Tab Widget, Semantic Web like the OWL extension[1] or Software Engineering like

[1]http://protege.stanford.edu/plugins/owl/

the UML Storage Backend[2] and manuals, FAQs and more information about the topic of Protégé 2000.

[2]http://protege.stanford.edu/plugins/uml/

Appendix D

Comparison Tables

These tables represent the analysis done to the transformation models. We compare our models represented in the three Web ontology languages with the model designed in UML.

D.1 Description of the Tables

The tables are divided in three sections. The first section contains the classes, the second section contains the attributes and the third section contains the relationships. The names are represented by C* for classes, A* for attributes and R* for relationships where * can be a number (for example RDF(S)C1).

D.1.1 Legend of the tables

TelephoneNumber: LanguageC1, Person: LanguageC2, Company: LanguageC3, Administration: LanguageC4, Human: LanguageC5, Information: LanguageC6, Place: LanguageC7, Organization: LanguageC8, Coordinate: LanguageC9, Subscription: LanguageC10, PostalCode: LanguageC11, Title: LanguageC12, VisitCard: LanguageC13, StreetNumber: LanguageC14, City: LanguageC15, Country: LanguageC16, Region: LanguageC17, Street: LanguageC18.

Attributes:

CityName: LanguageA1, ClassifiedCode: LanguageA2, CountryName: LanguageA3, DirectoryArea: LanguageA4, Entrance: LanguageA5, ExtraName: LanguageA6, First: LanguageA7, Full: LanguageA8, Id: LanguageA9, Internet: LanguageA10, Last: LanguageA11, Level: LanguageA12, Middle: LanguageA13, Name: LanguageA14, Number: LanguageA15, PostalCodeAttr: LanguageA16, PostBox: LanguageA17, RegionCode: LanguageA18, RegionName: LanguageA19, SectionCode: LanguageA20, StreetName: LanguageA21, StreetNumberAttr: LanguageA22, StreetNumbers: LanguageA23, Subdivision: LanguageA24, TelAddress: LanguageA25, TextAfter: LanguageA26, TitleName: LanguageA27, Type: LanguageA28, xCor: LanguageA29, yCor: LanguageA30.

Relationships:

BelongsPerson: LanguageR1, hasCities: LanguageR2, hasPerson: LanguageR3, hasPostalCOde: LanguageR4, hasRegions: LanguageR5, hasStreet: LanguageR6, hasStreetNumber: LanguageR7 ,hasSubscriptors: LanguageR8, hasTelephoneNumber: LanguageR9, hasTitle: LanguageR10, hasVisitCard: LanguageR11, hasWorkers: LanguageR12, isInaCOuntry: LanguageR13, isInaRegion: LanguageR14, isInCIty: LanguageR15, isInStreet: LanguageR16, isSubscribed: LanguageR17, worksFor: LanguageR18.

The rows represent if a concept is represented or not, specified or not, what cardinality does the concept have and if this concept has the same restrictions in both languages or not. The columns represent the concepts in the ontology in each Web ontology language. These concepts are classes, attributes and relationships. In each cell we can find 1 if the concept fulfils the property indicated in the column, 0 if not, same if it has the same restrictions,more if the analysed language has added more restrictions to the representation, - if it is no applicable and the type of cardinality.

D.2 RDF(S) and UML Comparison Table

	Represented	Specified	Cardinality	Restrictions
RDF(S)C1	1	0	*	same
RDF(S)C2	1	0	*	same
RDF(S)C3	1	0	*	same
RDF(S)C4	1	0	*	same
RDF(S)C5	0	0	*	-
RDF(S)C6	0	0	*	-
RDF(S)C7	0	0	*	-
RDF(S)C8	1	0	*	same
RDF(S)C9	1	0	1	same
RDF(S)C10	1	0	1	same
RDF(S)C11	0	1	1	-
RDF(S)C12	1	0	*	same
RDF(S)C13	0	0	*	-
RDF(S)C14	0	1	1	-
RDF(S)C15	1	0	1	same
RDF(S)C16	1	0	*	same
RDF(S)C17	1	0	*	same
RDF(S)C18	1	0	*	same
RDF(S)A1	1	0	1	same
RDF(S)A2	1	0	1	none
RDF(S)A3	1	0	1	same
RDF(S)A4	1	0	1	none
RDF(S)A5	1	0	1	none
RDF(S)A6	1	0	1	none
RDF(S)A7	1	0	1	none
RDF(S)A8	1	0	1	same
RDF(S)A9	1	0	1	same

RDF(S)A10	1	0	1	none
RDF(S)A11	1	0	1	none
RDF(S)A12	1	0	1	none
RDF(S)A13	1	0	1	none
RDF(S)A14	1	0	1	same
RDF(S)A15	1	0	1	same
RDF(S)A16	0	1	1	none
RDF(S)A17	1	0	1	none
RDF(S)A18	1	0	1	none
RDF(S)A19	1	0	1	same
RDF(S)A20	1	0	1	none
RDF(S)A21	0	1	1	same
RDF(S)A22	1	0	1	none
RDF(S)A23	1	0	1	none
RDF(S)A24	1	0	1	none
RDF(S)A25	1	0	1	same
RDF(S)A26	1	0	1	none
RDF(S)A27	1	0	1	same
RDF(S)A28	1	0	1	none
RDF(S)A29	1	0	1	none
RDF(S)A30	1	0	1	none
RDF(S)R1	0	1	1	none
RDF(S)R2	0	1	1	none
RDF(S)R3	0	1	1	none
RDF(S)R4	0	0	1	none
RDF(S)R5	0	1	1	none
RDF(S)R6	0	1	1	none
RDF(S)R7	0	0	1	none
RDF(S)R8	0	1	1	none
RDF(S)R9	0	1	1	none
RDF(S)R10	0	1	1	none
RDF(S)R11	0	0	1	none
RDF(S)R12	0	1	1	none
RDF(S)R13	0	1	1	none
RDF(S)R14	0	1	1	none
RDF(S)R15	0	1	1	none
RDF(S)R16	0	1	1	none
RDF(S)R17	0	1	1	none
RDF(S)R18	0	1	1	none

Table D.1: RDF(S)-UML comparison model

The results in the Table D.1 are the following. Six classes are not represented within UML but in RDF(S). That means 33% of classes can not be represented in UML but in RDF(S). Two attributes can not be represented in UML. That means 6,66% of attributes can not be represented in UML but in RDF(S). None of the relationships are represented in UML but in RDF(S).

In the specification section 11% of classes are not specified in RDF(S) but in UML. 6,66% of attributes are specified in UML and 83,33% of relationships in the RDF(S) representation are specified in UML. The cardinality is the same in both representations, RDF(S) and UML when the attribute, class or relationship is present in both representations. The restrictions are the same when the class, attribute or relationship is present, required when in both representations the restriction is required and none when the restriction can not be applied in UML.

D.3 OWL and UML Comparison Table

	Represented	Specified	Cardinality	Restrictions
OWLC1	1	0	*	more
OWLC2	1	0	*	more
OWLC3	1	0	*	more
OWLC4	1	0	*	more
OWLC5	0	0	*	-
OWLC6	0	0	*	-
OWLC7	0	0	*	-
OWLC8	1	0	*	more
OWLC9	1	0	1	same
OWLC10	1	0	1	same
OWLC11	0	1	1	same
OWLC12	1	0	*	same
OWLC13	0	0	*	-
OWLC14	0	1	1	more
OWLC15	1	0	1	same
OWLC16	1	0	*	same
OWLC17	1	0	*	more
OWLC18	1	0	*	more
OWLA1	1	0	1	same
OWLA2	1	0	1	same
OWLA3	1	0	1	same
OWLA4	1	0	1	same
OWLA5	1	0	1	same
OWLA6	1	0	1	same
OWLA7	1	0	1	same
OWLA8	1	0	1	same
OWLA9	1	0	1	same
OWLA10	1	0	1	same

OWLA11	1	0	1	same
OWLA12	1	0	1	same
OWLA13	1	0	1	same
OWLA14	1	0	1	same
OWLA15	1	0	1	same
OWLA16	0	1	1	same
OWLA17	1	0	1	same
OWLA18	1	0	1	same
OWLA19	1	0	1	same
OWLA20	1	0	1	same
OWLA21	0	1	1	same
OWLA22	1	0	1	same
OWLA23	1	0	1	same
OWLA24	1	0	1	same
OWLA25	1	0	1	same
OWLA26	1	0	1	same
OWLA27	1	0	1	same
OWLA28	1	0	1	same
OWLA29	1	0	1	same
OWLA30	1	0	1	same
OWLR1	0	1	1	same
OWLR2	0	1	1	same
OWLR3	0	1	1	same
OWLR4	0	0	1	-
OWLR5	0	1	1	same
OWLR6	0	1	1	same
OWLR7	0	0	1	same
OWLR8	0	1	1	same
OWLR9	0	1	1	same
OWLR10	0	1	1	same
OWLR11	0	0	1	-
OWLR12	0	1	1	same
OWLR13	0	1	1	same
OWLR14	0	1	1	same
OWLR15	0	1	1	same
OWLR16	0	1	1	same
OWLR17	0	1	1	same
OWLR18	0	1	1	same

Table D.2: OWL-UML comparison model

The results in the Table D.2 are the following. Six classes are not represented within UML but in OWL. That means 33% of classes can not be represented in UML but in RDF(S). Two attributes can not be represented in UML. That means 6.66% of attributes can not be represented in UML but in OWL. None of the relationships are represented in UML but in OWL.

In the specification section 11.11% of classes are specified in OWL and the rest of the classes are specified in UML. 6.66% of attributes are specified in UML and 93.33% are not specified in OWL. 83.33% of relationships in the OWL representation are specified in UML and only 16.66% are specified in OWL. The cardinality is the same in both representations, OWL and UML when the attribute, class or relationship is present in both representations. The restrictions are the same when the class, attribute or relationship is present, required when in both representations the restriction is required and none when the restriction can not be applied in UML. Here are more "none" than others. This is because we specify in OWL the "disjoint" restriction and it can only be specified in OWL.

D.4 Topic Maps and UML Comparison Table

	Represented	Specified	Cardinality	Restrictions
TopicMapsC1	1	0	*	same
TopicMapsC2	1	0	*	same
TopicMapsC3	1	0	*	same
TopicMapsC4	1	0	*	same
TopicMapsC5	0	0	*	-
TopicMapsC6	0	0	*	-
TopicMapsC7	0	0	*	-
TopicMapsC8	1	0	*	same
TopicMapsC9	1	0	1	same
TopicMapsC10	1	0	1	same
TopicMapsC11	0	1	1	-
TopicMapsC12	1	0	*	same
TopicMapsC13	0	0	*	-
TopicMapsC14	0	1	1	-
TopicMapsC15	1	0	1	same
TopicMapsC16	1	0	*	same
TopicMapsC17	1	0	*	same
TopicMapsC18	1	0	*	same
TopicMapsA1	1	0	1	same
TopicMapsA2	1	0	1	same
TopicMapsA3	1	0	1	same
TopicMapsA4	1	0	1	same
TopicMapsA5	1	0	1	same
TopicMapsA6	1	0	1	same
TopicMapsA7	1	0	1	same

TopicMapsA8	1	0	1	same
TopicMapsA9	1	0	1	same
TopicMapsA10	1	0	1	same
TopicMapsA11	1	0	1	same
TopicMapsA12	1	0	1	same
TopicMapsA13	1	0	1	same
TopicMapsA14	1	0	1	same
TopicMapsA15	1	0	1	same
TopicMapsA16	0	1	1	same
TopicMapsA17	1	0	1	same
TopicMapsA18	1	0	1	same
TopicMapsA19	1	0	1	same
TopicMapsA20	1	0	1	same
TopicMapsA21	0	1	1	same
TopicMapsA22	1	0	1	same
TopicMapsA23	1	0	1	same
TopicMapsA24	1	0	1	same
TopicMapsA25	1	0	1	same
TopicMapsA26	1	0	1	same
TopicMapsA27	1	0	1	same
TopicMapsA28	1	0	1	same
TopicMapsA29	1	0	1	same
TopicMapsA30	1	0	1	same
TopicMapsR1	0	1	1	same
TopicMapsR2	0	1	1	same
TopicMapsR3	0	1	1	same
TopicMapsR4	0	0	1	same
TopicMapsR5	0	1	1	same
TopicMapsR6	0	1	1	same
TopicMapsR7	0	0	1	same
TopicMapsR8	0	1	1	same
TopicMapsR9	0	1	1	same
TopicMapsR10	0	1	1	same
TopicMapsR11	0	0	1	same
TopicMapsR12	0	1	1	same
TopicMapsR13	0	1	1	same
TopicMapsR14	0	1	1	same
TopicMapsR15	0	1	1	same
TopicMapsR16	0	1	1	same
TopicMapsR17	0	1	1	same
TopicMapsR18	0	1	1	same

Table D.3: Topic Maps-UML comparison model

The results in the Table D.3 are the following. Six classes are not represented within UML but in Topic Maps. That means 33% of classes can not be represented in UML but in RDF(S). Two attributes can not be represented in UML. That means 6.66% of attributes can not be represented in UML but in Topic Maps. None of the relationships are represented in UML but in Topic Maps.

In the specification section 11.11% of classes are specified in Topic Maps and the rest of the classes are specified in Topic Maps. 6.66% of attributes are specified in UML and 93.33% are not specified in Topic Maps. 83.33% of relationships in the Topic Maps representation are specified in UML and only 16.66% are specified in Topic Maps. The cardinality is the same in both representations, Topic Maps and UML when the attribute, class or relationship is present in both representations. The restrictions are the same when the class, attribute or relationship is present, required when in both representations the restriction is required and none when the restriction can not be applied in UML.

Appendix E

Interview

The process for creating this interview has been based in [32]. To write the structure and the questions of this interview we decided to search for a good theoretical basis before start. We used [32] and as it is indicated in the book there are mainly three types of interview: fully structured interview (predetermined questions, with fixed words in determinate moments of the interview), semi structured interviews (predetermined questions but with a more flexible disposition of words) and unstructured interviews (without any kind of structure). We selected the second type of interview because the other two do not fit in our goals. We need some flexibility to allow the person interviewed to tell to us all his knowledge but we need some kind of structure in the interview.

The question focus will be the databases. All the questions will be related to our databases. We will mainly make direct questions to know the information we need but if it is necessary we will be flexible when asking to our interviewed. The core of the interview is shown in the following lines. The questions are asked by us (We in the interview) and the answers come from the Invenio domain expert Anders Kofod-Petersen (Invenio Expert IE).

We: Our first question is, do you have the design model for the databases? I mean a graphical schema or something like that?

IE: No, we bough the databases, just the data from another company. We did not receive a model. We only got the DTD for the Swedish database.

We: We have been looking at the data you gave to us and we have some questions about it. First of all, this design is a bit messy. Where did you get this database?

IE: The company bough the databases to a company dedicated to print directory services. Probably the databases are designed for this purpose.

We: We have printed several examples of the data stored in these databases; can you have a look at them?

IE: Yes, it seems what I have told you. This format seems to be for printing a directory service. Like a "yellow pages" or something like that. At least if we look at these examples of both databases.

We: Can we start to look at the structure of the databases? We have several questions about the structures and attributes of the databases. First the Swedish one. What is the Publications structure for?

IE: It seems the main structure of the database. It contains all the Publication.

We: And Publication? What is it for?

IE: Is the structure for storing the elements of each region. Publication and its attributes are for representing the people who lives in this region.

We: Then, Placement and Row are structures that contain all the information related to a region.

IE: Yes, Placement seems to be the elements that identify a region, this is because it can only have one in each Publication and we can have several Row.

We: If we have understood correctly, Row represents one person.

IE: Not exactly. Row represents one set of addresses which belong to the same person. Each Row can have assigned several telephone numbers, but to the same person. Or in one Row has several telephone numbers, with several addresses and several names, but all related.

We: Where the telephone number is represented?

IE: It is represented in the TelAddress attribute.

We: What is the Subscription structure for?

IE: It is for indicate the kind of subscription to the company. You can find there the telephone number, maybe an email.

We: What are Name and ExtraName for?

IE: They represent a Person. ExtraName contains some extra names, aliases or something like that for the person represented in Name.

We: Does Address represent the data of the address of the person represented in the Name class?

IE: Yes, it does. It contains all the information of the address of one person.

We: What is the structure Coordinate for?

IE: It is for representing the coordinates of the addresses we have.

We: But, there is no data in this structure in the database.

IE: This is because it is a proposed extension to the database that it has not been developed yet.

We: We have finished with our questions about the Swedish database. Let's start with the Norwegian database. We do not have any kind of schema, model or design here. We will start asking for the attributes. What is each attribute for?

IE: If we look at this example of data from the database we can guess the meaning of each attribute.

We: Ok, we can suppose that the first attribute is the identifier of the each company, because we have not seen any person name here. And the second one is the name of the company.

IE: Exactly, first the Id and second the name.

We: What is the third attribute?

IE: It seems a subdivision of the company. For example, university and faculty inside this university.

We: What is the fourth attribute?

IE: It is a Post Office box.

We: Can you explain us the rest of the attributes of the database?

IE: Yes. The fifth attribute is the zip code for the post office box. The next one is the city where this name is. The seventh is the street where the company is and the next one is the street zip code. The ninth attribute is the region and the next two attributes are telephone numbers assigned to this company. The twelfth and the thirteenth attributes are codes for identifying the region and the last one is the classification of the company.

We: What kind of classification?

IE: It is divided in two. A code for identifying the kind of activity of the company and a short description of this activity.

We: We have finished with our questions about the databases. Do you have any recommendation for the next steps we have to do?

IE: The most important question is the design of the ontologies. This should center the approach to the case study. The databases are only data and you can only use it to extract data. The data will become in information and knowledge in the ontology. Therefore the ontology is the most important. But of course we need the data to obtain the knowledge we want.

Appendix F

The Thesis CD

In this chapter, we indicate the contents of the CD that is attached to the documentation. In this CD it is possible to find all the necessary file for running and creating the ontologies.

```
This CD contains:

readme.txt: this file

Directory Documentation:

    Contains the directory:
        \documentation\References: Inside this directory are stored
        most of the references used.

    Contains the files:
        \documentation\thesis.pdf: Main document of this master's
        thesis.
        \documentation\planning.mpp: Planning of the master's thesis.

Directory Ontologies:

Contains the subdirectories:

    \Ontologies\MergedDB:

    Contains the subdirectories:

    \Ontologies\MergedDB\OWL
    \Ontologies\MergedDB\RDFS
    \Ontologies\MergedDB\TopicMaps

    In each subdirectory there are the corresponding files for the
    Protege projects in each language.
    The TopicMaps directory also contains the TMMerged.xtm that
    contains the ontology in XTM format.
    The OWL directory also contains the OWLMergedDBinferred.owl
    which contains the inferred version of the OWL ontology.

    \Ontologies\SweOWL
    Contains the initial OWL ontology for the Swedish database.

    \Ontologies\SweRDF
    Contains the initial RDFS ontology for the Swedish database.

    \Ontologies\SweTopicMaps
    Contains the initial TopicMaps ontology for the Swedish database.


Directory Parser:

Contains the files:

    Conversion parsers for the Norwegian database:
        \Parser\OWLParser1Nor.java
        \Parser\RDFNorMerged.java
        \Parser\TMNorMerged.java

    Conversion parsers for the Swedish database:
        \Parser\MergedParser.java
        \Parser\RDFMerged.java
        \Parser\TMMerged.java

    An example of the Norwegian database in the ontology format
    after the parsing process:
        \Parser\NorOWLontoparsed.txt
        \Parser\NorRDFontoparsed.txt
        \Parser\NorTopicMapsontoparsed.txt

    An example of the Swedish database in the ontology format
    after the parsing process:
        \Parser\RDFontoparsed.txt
        \Parser\OWLontoparsed.txt
        \Parser\TopicMapsontoparsed.txt

    Usage of the parser:

    For compiling them: javac parsername.java
    For running them: java parsername and follow the instructions.
    The output is in the directory c:\temp

Directory tools:
```

```
Contains the files:

    \Tools\install_protege.exe (installation file for Protege)
    \Tools\tmtab.zip (Topic Maps plugging for Protege)


Directory databases:

Contains the files:
    \databases\norwegian.txt (sample of the data stored in the
    Norwegian database)
    \databases\swedish.txt (sample of the data stored in the
    Swedish database)


Note: once the output files are generated from the conversion
parsers it is necessary to copy and paste the data parsed into the
ontology files. Be careful not to delete the definition of the
ontology in each format.
```

Appendix G

The Project Planning

In this chapter, we are going to explain the planning of our project. The planning contains a Gantt diagram that shows the way of working during the semester. It also shows some unexpected events and how we managed them. We started the 24th of January and the initial idea was to deliver on 16th of June. Finally we delivered the 30th of June because I had to return to Spain. My grandfather died the 30th of April.

G.1 The Planning

We started our project by planning all the research process. We needed two working days to do this task. After this task we had planned the project. We left several free days because we thought the things may don't go like we want. The next task planned was to make the interview to the domain expert of the company and develop the initial design of the ontologies. We needed 10 working days to do this. The interview needed most of the time because we had to choose correctly the questions, how to do it and select a proper date for the interview. The initial design for the models needed less time because we had an initial view of the problem from the last semester project [8].

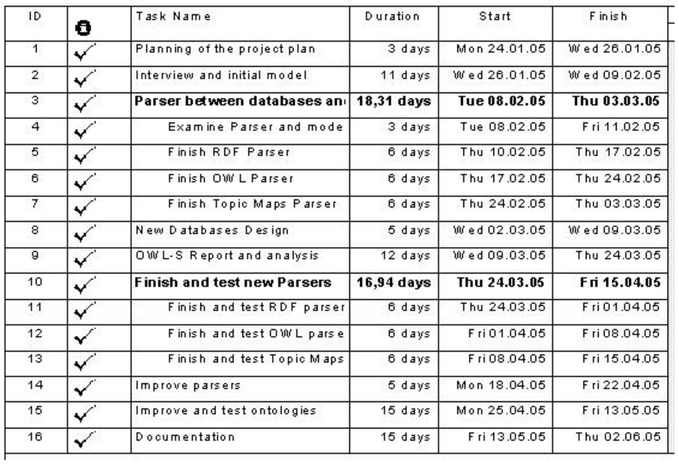
Once we had the initial model we started to develop the conversion parsers for translating the data represented in the original databases into the ontology format of each Web ontology language. To do this we needed 17 working days. We needed two working days to plan the way for developing the conversion parsers and five days for each parser. This was our initial planning, but we modify the time for each parser. Topic Maps was more difficult than the others, but we managed to get everything on time. We used less time for the RDF(S) and OWL conversion parsers.

Once we had the conversion parsers we saw that the data was quite similar and we can merge both databases. We started with the new design and we finished it in five days, as we planned.

After the new design, we analysed OWL-S, the Web services extension for OWL. We planned to need about 12 working days. We achieve this objective in this time because we already know how to analyse the languages and the time we will need.

We saw the parsers didn't work properly, mainly the Topic Maps parser and we had to add ten more working days for improving them. Once we had the parsers completed we started to test the ontologies. The 25th of April I received sad news. My grandfather was dying. I had to return to Spain. I returned from Spain the 5th of May. Because of this reason I extended the deadline for delivering the master's thesis. In the Gantt diagram this is not shown.

We planned 15 days for improving and testing the ontologies. We used a bit more due to we had to modify and test again the conversion parser. We needed 15 days to finish the documentation. Reports of all the tasks have been done during all the tasks. Five days were used to try to access the databases via RACER and PAL tab was unsuccessful. The eastern holidays are not represented in the diagram. With these holidays, the days for the RACER test and the travel to Spain we arrive to the 25th of June. We have to count the weekends and we get the full planning. The planning is shown in Figure G.2 and Figure G.1.



ID		Task Name	Duration	Start	Finish
1	✓	Planning of the project plan	3 days	Mon 24.01.05	Wed 26.01.05
2	✓	Interview and initial model	11 days	Wed 26.01.05	Wed 09.02.05
3	✓	**Parser between databases an**	**18,31 days**	**Tue 08.02.05**	**Thu 03.03.05**
4	✓	Examine Parser and mode	3 days	Tue 08.02.05	Fri 11.02.05
5	✓	Finish RDF Parser	6 days	Thu 10.02.05	Thu 17.02.05
6	✓	Finish OWL Parser	6 days	Thu 17.02.05	Thu 24.02.05
7	✓	Finish Topic Maps Parser	6 days	Thu 24.02.05	Thu 03.03.05
8	✓	New Databases Design	5 days	Wed 02.03.05	Wed 09.03.05
9	✓	OWL-S Report and analysis	12 days	Wed 09.03.05	Thu 24.03.05
10	✓	**Finish and test new Parsers**	**16,94 days**	**Thu 24.03.05**	**Fri 15.04.05**
11	✓	Finish and test RDF parser	6 days	Thu 24.03.05	Fri 01.04.05
12	✓	Finish and test OWL parse	6 days	Fri 01.04.05	Fri 08.04.05
13	✓	Finish and test Topic Maps	6 days	Fri 08.04.05	Fri 15.04.05
14	✓	Improve parsers	5 days	Mon 18.04.05	Fri 22.04.05
15	✓	Improve and test ontologies	15 days	Mon 25.04.05	Fri 13.05.05
16	✓	Documentation	15 days	Fri 13.05.05	Thu 02.06.05

Figure G.1: Planning tasks

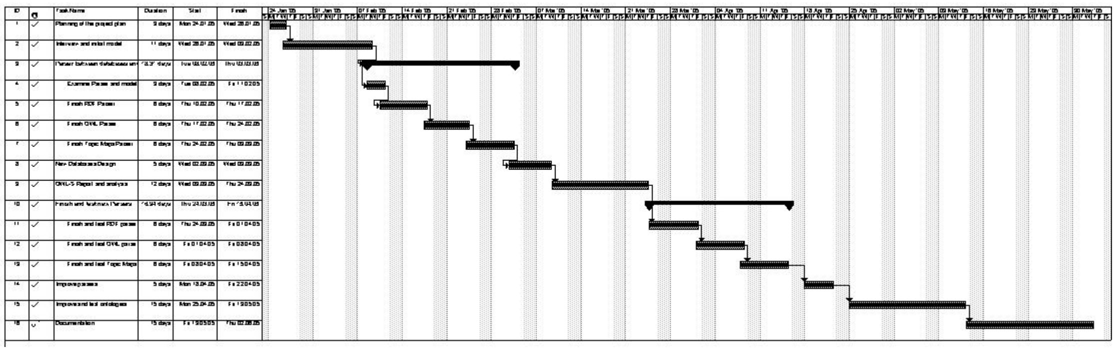


Figure G.2: Gantt Diagram

G.2 Concluding Remarks

We have seen in this chapter that to plan a project is a useful idea. It helps to work when you need to and the planning tool helps to fulfil the objectives on time. Of course always happen mistakes and unexpected situations and it is important to know how to manage them.