Use of web-based ontologies for geodata and service descriptions

Helbert Arenas Paredes
March, 2003
Use of web-based ontologies for geodata and service descriptions

by

Helbert Arenas Paredes

Thesis submitted to the International Institute for Geo-information Science and Earth Observation in partial fulfilment of the requirements for the degree in Master of Science in Geoinformatics.

Degree Assessment Board

Thesis advisor
First supervisor Ir. R. Lemmens
Second supervisor Dr. Ir. R. de By

Thesis examiners
Chairman Dr. Ir. R. de By
External examiner Ms. Drs. M. de Vries

INTERNATIONAL INSTITUTE FOR GEO-INFORMATION SCIENCE AND EARTH OBSERVATION
ENSCHEDE, THE NETHERLANDS
Disclaimer

This document describes work undertaken as part of a programme of study at the International Institute for Geo-information Science and Earth Observation (ITC). All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the institute.
Acknowledgements

I would like to express my gratitude to the Netherlands Government for the award of a fellowship that allow me to study in ITC.

I am very grateful to my supervisors Ir. R. Lemmens and Dr. Ir. R. de By, for their guidance, valuable suggestions and critical comments.

I want to thank to my family in Peru that helped me to be here, and to my classmates that became my family in Netherlands.

Quiero expresar mi sincera gratitud a Estela, mi gran amiga que me enseño a disfrutar de un buen libro.

Akhirnya secara khusus saya sampaikan terimakasih untuk kekasih yang manis, yang telah banyak membantu saya selama belajar di ITC, terimakasih Dessi.
Abstract

Nowadays the increasing amount of both data sets and web services demands for better ways to describe them in order to allow a user to find and access them. This kind of descriptions should allow human and software users to determine if they are compatible. Our approach to solve the problem is to make descriptions of data sets and web services using a meaningful vocabulary. The elements of this vocabulary have relations that allow human and software applications to infer non explicit facts from the descriptions. To make the descriptions we use Description Logics and write them using description languages based on XML. In this research we develop a prototype designed to help a human user to create descriptions of data sets and services. Based on this descriptions we develop a prototype that can make matches between data sets and services based on simple inferences. The results look promising. However further work is required to use this techniques in more complex, real life situations.

Keywords

esemantics, semantic web, web services, taxonomies, metadata, Description Logics, OWL, knowledge representation, knowledge base, vocabulary, concepts
Contents

Acknowledgements i

Abstract iii

List of Tables vii

List of Figures ix

1 What this thesis is all about? 3
  1.1 Objectives ......................................................... 5
  1.2 Research Questions ............................................... 5
  1.3 Methodology ...................................................... 5
  1.4 Thesis Outline ................................................... 6

2 Geo Web Services & Data sets 7
  2.1 Web Services ..................................................... 7
     2.1.1 Web Service Architecture .................................... 8
     2.1.2 Simple Object Access Protocol (SOAP) ......................... 8
     2.1.3 Universal Description Discovery Language (UDDI) ............. 9
     2.1.4 Web Services Description Language (WSDL) .................. 9
  2.2 Geographic Web Services .......................................... 10
  2.3 Data sets ......................................................... 12
  2.4 Concluding Remarks ............................................... 15

3 Knowledge Representation 17
  3.1 Ontologies ........................................................ 17
  3.2 Description Logics ................................................ 18
     3.2.1 Basic Syntax ................................................ 19
     3.2.2 Formal Semantics ............................................ 20
     3.2.3 Reasoning: .................................................. 20
     3.2.4 Algorithms .................................................. 21
  3.3 Working Knowledge Base ........................................... 24
     3.3.1 Individual Descriptions ..................................... 26
     3.3.2 The Processing Service Descriptions .......................... 26
     3.3.3 Definition of the Input and Output Concepts ................. 29
     3.3.4 Matching a Data set with a Service Input ................... 30
  3.4 Description Languages based on XML ............................. 31
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1 RDF &amp; RDFS</td>
<td>32</td>
</tr>
<tr>
<td>3.4.2 OWL</td>
<td>35</td>
</tr>
<tr>
<td>3.4.3 Some Interesting Elements of OWL DL vocabulary</td>
<td>36</td>
</tr>
<tr>
<td>3.5 How to use the ontology information</td>
<td>41</td>
</tr>
<tr>
<td>3.6 Concluding Remarks</td>
<td>42</td>
</tr>
<tr>
<td><strong>4 Implementation</strong></td>
<td>45</td>
</tr>
<tr>
<td>4.1 The Tools</td>
<td>45</td>
</tr>
<tr>
<td>4.2 The Ontology</td>
<td>47</td>
</tr>
<tr>
<td>4.2.1 List</td>
<td>48</td>
</tr>
<tr>
<td>4.2.2 DataSet</td>
<td>48</td>
</tr>
<tr>
<td>4.2.3 Service</td>
<td>50</td>
</tr>
<tr>
<td>4.3 Prototype Design</td>
<td>51</td>
</tr>
<tr>
<td>4.3.1 Users</td>
<td>51</td>
</tr>
<tr>
<td>4.3.2 Sequence Diagrams</td>
<td>51</td>
</tr>
<tr>
<td>4.3.3 Descriptor</td>
<td>55</td>
</tr>
<tr>
<td>4.3.4 Match Tester</td>
<td>60</td>
</tr>
<tr>
<td>4.3.5 Example Case Scenario</td>
<td>62</td>
</tr>
<tr>
<td>4.4 Concluding Remarks</td>
<td>67</td>
</tr>
<tr>
<td><strong>5 Conclusions</strong></td>
<td>69</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>71</td>
</tr>
<tr>
<td><strong>A Glossary</strong></td>
<td>73</td>
</tr>
<tr>
<td><strong>B Examples</strong></td>
<td>77</td>
</tr>
<tr>
<td><strong>C Ontology</strong></td>
<td>81</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Industrial and GIS protocols (source: [AL03]) ..................................... 10
4.1 Information about the products offer by the data set provider .............. 63
4.2 Service characteristics ................................................................. 64
List of Figures

2.1 Service Architecture - source[Sys03] ......................... 9
2.2 Services Taxonomy ............................................ 13
2.3 UML Packages .................................................. 14
2.4 Spatial Representation ........................................ 16
2.5 Data set Restrictions .......................................... 16

3.1 An example of a network graph representing knowledge ...... 18
3.2 The basic architecture of a knowledge representation system based on Description Logics (source: [BMNPS02]) ..................... 19
3.3 A service description involves entities that belong to the ABox and to the TBox, clearly the description of the input of a service, is not an individual, but a concept .......................... 28
3.4 Service Allowed Inputs Definition ............................ 30
3.5 RDF graph of the resource http://www.gfm2.org#myDataset showing two of its relations ............................................. 33
3.6 OWL Union ....................................................... 40
3.7 OWL Intersection ................................................. 41
3.8 OWL Complement ................................................ 41
3.9 An Ontology with Service and Data set descriptions ........ 43

4.1 The Protege ontology editor ................................. 46
4.2 Use Case Diagrams in the prototype design ................. 52
4.3 Sequence Diagram for a Data set Provider ................ 52
4.4 Sequence diagram for a Service Provider .................. 53
4.5 Sequence diagram for a Match operation .................. 54
4.6 UML Class Diagram for the descriptor prototype ...... 56
4.7 Graphical Interface of Estela09 ............................. 56
4.8 Graphical Interface provided by the class OntViewer02 .... 57
4.9 Graphical Interface provided by the class MyInstances .... 58
4.10 Graphical Interface provided by the class RestWindow ... 59
4.11 UML Class diagram of Match Tester ....................... 61
4.12 The Matching Prototype ........................................ 62
4.13 General use diagram of the prototypes .................... 62
4.14 Description of a Scanned Map with Descriptor ............ 64
4.15 Description of a service with Descriptor .................. 65
List of Figures

---

1
Chapter 1

What this thesis is all about?

Nowadays the web is becoming a common medium for the access of resources and the process of them. This also applies to our field, Geo Information. It is more common to request and access data sets through the internet, and it is going to be more common to process these data sets, using specific tools, Services, provided by software developers through the web.

The advantage of this, for the users, is that they will not have to purchase a complete software with a big set of functions, from which they are going to use only few. Using the services provided in this way, the users will be able to create their own specific solutions for their particular tasks.

This way of providing functionalities through the web, is based on the Service Oriented Architecture, a technology that has evolved in the past decade. However, the use of proprietary protocols and formats, has had an effect of slowing down the possible extensive use of applications based on this technology.

With the advent of XML, now we have a common language, that is a standard. web Services, use XML as a format of access, in this way, web Services, can have a communication format, that anybody can use. This also explains the increasing number of services available in the web.

Another important factor to notice is the increasing number of data sets available also on the web mainly because of the high cost of geographic data, and the need to reuse and share it.

This increasing number of available data sets and services creates a new problem: the information overflow [BLHL01]. In our field of work, the Geo Information, we can perceive that when we search the web for some geographic web service or Data set, it could be a difficult task. Nowadays the amount of available data is increasing. This increase threatens to even in some point overload our capability to process the information. There is a need for a way to process this data flow, in an efficient way.

When we try to make a search in today’s web, one factor that difficult our task is the fact that there are lots of terms for similar concepts and different concepts that refer to the same term. This is because we are humans, and human languages are pretty ambiguous.

A way to deal with this information problem, is to structure the data, in a way that could be processable by the computers, in that way computers would be able to help us. That is the aim of a new development in web technology.
called 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."[BLHL01]

The Semantic Web is an effort lead by the World Wide Web Consortium W3C. It is based on the Resource Description Framework (RDF), which uses XML for syntax, and URI's for naming.

RDF is a framework for the representation of information on the web. It is designed to represent information in a flexible way, allowing data, to be processed outside the particular environment where it was created.

Based on RDF, the W3C, developed the OWL (Web Ontology Language), which has as a purpose, the process of information instead of just the presentation of it to human users. OWL goes beyond XML, and RDF. It allows the creation of documents that can be processed by machines, thanks to an additional vocabulary and some formal semantics.

The term Ontology refers to a formal description of the concepts and relationships that can exist in some particular area of interest. There are several definitions of ontology. According to dictionary.com it is:

"An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them"

The Ontologies are used often in the context of Semantic Web to represent some field of knowledge, stating some inference rules, and grouping its elements in classes and structuring these classes in a taxonomy.

The main purpose of an ontology is enable the reuse and sharing of knowledge. This allows different actors to refer to the same ontology, and allow them to speak a common language. It also allows that when we state that one element belongs to a certain class, based on our ontology, we can determine, how this class relates to other classes, finding information that is not explicitly stated.

An Example Scenario:

A given user wants to detect human constructions in a certain area of the rain forest. To perform this operation, this user obtains remote sensing images of the study area of type $Grid_x$. He wants to process this images with an object detection service. In the search for the appropriate service, the user finds several ones. Based on their description the user would have to determine if his/her data set can be processable by this service. In the event that the number of available services is big, and the relations between the different classes of data sets is complex, the matching attempt would take a long time.

Using an ontology we can specify the different relations that exist between the different types of data sets and services. Allowing an application to perform inferences based on these relations. Allowing the user to make easier and more complex queries.
1.1 Objectives

The objective of this research was to develop a method to describe the data and software components in such a way that the descriptive documents refer to ontologies accessed through the web.

This entails

- The definition of the structure and necessary elements of a descriptive document: Describing methodological alternatives to access ontologies and process their information.

- Developing a prototype description creator: a software tool that helps a provider of spatial data / software to create ontology based capability descriptions.

- Developing a prototype description browser: a software tool to visualize descriptions and help the user to identify the usefulness of the component(s).

- Creation of component chain scenarios and test the prototypes with a number of component descriptions.

1.2 Research Questions

- How can we represent knowledge in a way that can be processable by software applications.

- What are the requirements to describe data sets and service in order to allow possible matches between them?

1.3 Methodology

Step 1: Review information about web services and data sets, and the current specifications to make a description of them

Step 2: Review existing description languages, its capabilities and limitations. Define the one that is going to be used in this research.

Step 3: Define the elements required to be described, and the necessary primitives that are going to be used to make the descriptions.

Step 4: Evaluation of available tools

Step 5: Development of prototype

Step 6: Evaluation of prototype
1.4 Thesis Outline

- Chapter 2 Data sets and web Services: web Services in general, the techniques involved in the deployment of the web services architecture. The view of the web services and the data sets by the OGC.

- Chapter 3 Knowledge representation: We focus on Description Logics and its formal background. The definition of classes, and the relations between classes, (inheritance, subclass, superclass). Description of the main algorithms used to search between classes and instances. How are all these concepts are translated into languages based on XML like RDF, DAML, OIL and OWL.

- Chapter 4 Implementation: Available tools to work with OWL, Protege, Jena, java SDK. Description of the prototype and its functionality.

- Chapter 5 Conclusions and Further work:
Chapter 2

Geo Web Services & Data sets

Because of the developments on technology, nowadays we have more users of digital geographic data sets, and geographic information systems. The number of users and producers of geographic data sets increases, and the need for a method to identify those information becomes bigger.

Usually a geographic data set, is used by other people than the ones that produce it. Then, it is necessary for the final users, some documentation to be able to identify the geographic data set that is more suitable for their particular needs, this documentation is commonly known as metadata. This documentation will also allow the final users to better manage, process, store, update and reuse the geographic information.

Nowadays, most of the process of the data sets is done by desktop applications, however the use of new techniques like the web services is growing. The rapid developments in technology and the economic reasons involved in the process of data sets, indicate that the web services are going to be used extensively in the future.

2.1 Web Services

Web services are software applications available on the web. Web services use standard data formats like XML and standard web protocols like Hypertext Transport Protocol (HTTP). Using these standards, web services can be discovered, described and accessed over intranets, extranets and the Internet [DJS03].

Web services work like components of bigger solutions. One big advantage of the web services is that their user does not need to know how the service was implemented, to use it.

Web services could be consider the evolution of the traditional Service Oriented Architecture (SOA). This kind of applications evolved over the last decade to provide high performance, scalability, reliability systems. However in the past, clients had to use specific protocols,(DCOM, CORBA) to access these kind of applications, and this factor used to limit the flexibility of the system. Web Services, take advantage of all the experience earned with SOA, and apply it to the use of the web and web protocols[Sys03].
2.1. Web Services

The advantage of using standard web protocols, is the independence of vendor or platform. Web Services are very flexible, any application can be deployed as a web service.

The users of a web service could be:

- A human user accessing the web service though a desktop or a wireless browser.
- An application program.
- Another web service.

Typically the a web service shows these characteristics:

- It is accessible though the web
- It provides an interface that can be accessed by another program.
- It is registered and can be located through a Web Service Registry.
- It can communicate to other web service, by passing messages.

2.1.1 Web Service Architecture

In the Web Service architecture exists three main actors and three main operations (Figure: 2.1)

- the provider of the service, that will provide to a broker, with a description of its service, and how to access it. (operation:publish)
- a broker that will register the service, and will allow a consumer to search for certain service on the registry.
- and the web service consumer, the one search the register for a service (operation:find), and in a later stage, request the use of it to the provider (operation:bind).

These actors use standard protocols based on XML, to perform all the operations, these protocols are:

- Universal Description Discovery Language.
- Web Services Description Language.

2.1.2 Simple Object Access Protocol (SOAP)

SOAP is the messaging protocol, it provides a mechanism that allows one application to send an XML message to another application. A SOAP message consist of four parts:

- SOAP Envelope, Identifies the contents of the message, and explain how the message should be processed.
• SOAP Transport Binding Framework
• SOAP Serialization Framework
• SOAP RPC Representation

The SOAP with attachments binding can be used to transport non-XML data, like multimedia files, as attachments to a SOAP message.

2.1.3 Universal Description Discovery Language (UDDI)

UDDI is the mechanism to register and categorize web services. A UDDI registry, contains the following information:

• Business Entity, the information necessary to identify a business, name, contact information.

• Business Service, business description of the service, list of categories that describe the service, and a list of binding templates that could provide technical information about the service.

• Binding Templates, it could contain, the access point of the service implementation and a pointer to the WSDL document that describes the service.

• Service Types, defined by a construct called tModel, it defines an abstract service.

2.1.4 Web Services Description Language (WSDL)

WSDL is an XML vocabulary used to describe web services. A WSDL document, contains information about the functionality of the web service, how to access
2.2. Geographic Web Services

A WSDL document describes a service as a collection of ports, each port is defined by a port type, which supports a collection of operations. Each operation processes a particular set of messages. There are five major elements in a WSDL document:

- `<type>`: It defines the types used by the `<message>` elements they refer to the WC XML schema Part: Datatypes Recommendation.

- `<message>`: It defines the format of the messages. Messages are used as input and output for operations.

- `<portType>`: Defines an operation. Each `<operation>` has some input and output messages associated with it.

- `<binding>`: It associates the operations and the messages defined in a `<portType>` to a specific protocol and data format specification.

- `<service>`: Defines a set of related ports, a `<port>` associates a binding to a location of an instance of the web service.

WSDL service definitions provide documentation for distributed systems. However, WSDL does not support semantic descriptions like constraints. It has its concept of input and output defined by XSD [LH03].

Lots of GIS functionalities are being provided as web services, using in some cases the W3C recommendations. Although the Open GIS Consortium adopts most of the W3C standards, OGC encourages the use of its own standards to support the GIS web services.

<table>
<thead>
<tr>
<th>operations in web services</th>
<th>Industrial standards (W3C/UDDI org.)</th>
<th>GIS community (OGC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binding</td>
<td>HTTP GET/POST SOAP</td>
<td>HTTP GET/POST SOAP-RPC</td>
</tr>
<tr>
<td>describing</td>
<td>WSDL</td>
<td>GetCapabilities doc</td>
</tr>
<tr>
<td>brokering</td>
<td>UDDI</td>
<td>Registry Services (based on ebXML)</td>
</tr>
</tbody>
</table>

2.2 Geographic Web Services

The focus of this research is the description of geographic web services and the data sets where these services operate on. According to [Web81], a description is the act of mentioning the essential characteristics associated with a thing. A
common procedure associated with the description of a thing, is its classification into certain class. The fact of belonging to certain class, implies that this thing, possesses the characteristics associated with that specific class. In this section we review the classification schemas for services provided by Open Gis Consortium.

The Open Gis Consortium provides a set of geographic information standards through the ISO19100 series. In this series, we can find the Topic 12: OpenGIS Service Architecture, (ISO19119)[Con01b], which provides us with a set of different ways to consider the web services architecture, based on the different viewpoints.

• The enterprise viewpoint
• The computational viewpoint
• The information viewpoint
• The engineering viewpoint
• The technology viewpoint

For this research I will restrain to the information viewpoint, which deals with the semantics of information and information processing. The information viewpoint, considers six categories for the web services:

• Human interaction services
• Model/Information management services
• Workflow/Task services
• Processing services
• Communication services
• System management services

The services that deal with geographic data sets are the processing services. These kind of services, partially modify the attributes of the data sets. The Processing services can be classified in four categories:

**Geographic processing services spatial**
Example: Coordinate conversion, Coordinate transformation, Coverage/vector conversion, Route determination.

**Geographic processing services thematic**
Example: Geoparameter calculation, Feature generalization, etc.

**Geographic processing services temporal**
Example: Subsetting service, Sampling service, etc.

**Geographic processing services metadata**
Example: Statistical calculation service, Geographic annotation services, etc.
Currently OGC prescribes geographic web services to describe themselves through the "GetCapabilities" operation. The result of the operation is an XML-encoded document, called "capabilities document".

The "capabilities document" contains a high level description of the service instance and its provider. The first part of the document describes the service interface. In the second part it describes the data content of the service, (the data where the document operates on)[Con02].

According to the OGC the elements necessary to make the descriptions of the data sets and the web services, can be found in the ISO19115[Con01a] and the ISO19119. In these documents OGC defines metadata elements, groups this elements in packages and provides an schema to describe the geographic components(data sets and services).

This research provides guidelines to describe geo web services and data sets referring these descriptions to public, web based structures known as ontologies. To show this, data sets and web services are described using a subset of all the elements enumerated in the documents ISO19115, and ISO19119.

The main elements used are the ones related to the identification of the web service and the ones that link this to the data where it operates on, the so called operation signature.

Using this information we can build one taxonomy of the services (Figure 2.2), and declare for them some characteristics. We can also state in which kind of data sets they operate on.

To facilitate the discovery of services, and allow the use of them by customers that do not have the knowledge to navigate a taxonomy, It is possible to group some service types used for some specific task. OGC calls this structure a Service Organizer Folder (SOF) this is a subset of all the available services types, applicable to some specific situation. Then the user can look for a situation similar to his, and browse only in a subset of service types.

### 2.3 Data sets

We define a data set as a identifiable collection of data. By data we understand numbers, characters,images,or any other method of recording that can be accessed by humans, or can be used as input for computers. Data by itself does not have meaning, only when processed it transforms in information.[LPg]

The ISO19115, define a set of different elements that refer to the characteristics of data sets. They are grouped in sections (UML Packages), each of this packages can contain one or more entities (UML Classes), which can be specified (subclasses). Each of the entities contain elements (attributes). In Figure 2.3 we see the main packages described in the ISO19115 [Con01a].

From all the attributes enumerated in this document , in most of the cases only a subset is used. One subset of these attributes is the core metadata. The purpose of these elements is to identify a data set usually to catalogue it. Based on the information we must be able to state the, topic, location, date, and point of contact of the data set. The core metadata is composed by the following items:

- Data set title (mandatory) The title of the dataset declared by the data set
provider.

- Data set reference date (mandatory)

- Data set responsible party (optional) The person or institution that is responsible for the data set.

- Geographic location of the data set by four coordinates or by geographic identifier (conditional), the minimum bounding rectangle within which the data is available.

- Data set language (mandatory)

- Data set character set (conditional)

- Data set topic category (mandatory), the main theme of the data set, it refers to a topic category code list.

- Spatial resolution of the data set (optional)

- Abstract describing the data set (mandatory)

- Distribution Format (optional)
2.3. Data sets

- Additional extent information for the data set (vertical and temporal) (optional)
- Spatial representation type (optional)
- Reference System (optional)
- Lineage statement (optional)
- On line resource (optional)
- Metadata file identifier (optional)
- Metadata standard name (optional)
- Metadata standard version (optional)
- Metadata language (conditional)
Chapter 2. Geo Web Services & Data sets

- Metadata character set (conditional)
- Metadata point of contact (mandatory)
- Metadata date stamp (mandatory)

2.4 Concluding Remarks

WSDL is a commonly used XML format for describing web services. However, it does not provide support for semantic descriptions of services [LH03]. Using WSDL and the GetCapabilities document, we presume that the service is an exact implementation of some service type [Con02]. However, in the case of geographic web services, the information stored in the documents is not enough to describe their functionality, and their possible functionality restrictions. In Geographic Information Services, it is not unusual that even if a Service operates on a certain type of data set, it requires that this data set have some specific characteristics in order that the operation have sense, even more, it is possible that the specific input had some non-explicit relation with the available data set. Those kinds of relations and constraints, have to be declared to allow possible matches.

A way to do this is to organize the different geo-components, data sets and services, in classes and then specify the relations between the different classes.

In this research I provide guidelines to describe data sets in order that these descriptions could be used to determine whether a given data set could be used with a web service.

For testing and demonstrative purposes we can create a classification of data sets based on its spatial representation. This classification is based on a Uniform Modified Language (UML) package known as MD_SpatialRepresentation, Figure 2.4. This package is contained in the document ISO19115. This choice is made to show and test techniques to describe data sets based on a taxonomic classification.

To organize the web services, we use the classification described in the document ISO19119. To describe the input of a given service we use the same terminology used to describe data sets. In this way, we are able to determine the relation between a data set $X$, and the input of certain service, because both descriptions will be based on the same structure. The descriptions of the data sets are compared with the declared input of the services and the possible match will be evaluated.

Back, in the first chapter I introduce an example scenario, we have an available data set of type $Grid_x$ and we want to know if we can use it as input of a given service that in its description declares that accepts data sets of type $Grid_i$ under certain restrictions $R1$ and $R2$.

Based on the descriptions of the type $Grid_x$ and the descriptions of type $Grid_i$, and the relation between those classes in our ontology, we can declare if all members of the class $Grid_x$ can be considered as members of the class $Grid_i$.

Then we can analyze if the properties declared for our data set fulfill the requirements of the service, based on that analysis we can determine if our
2.4. Concluding Remarks

Figure 2.4: Spatial Representation

data set fulfills the restrictions \((d_1)\), if it fulfills the restrictions partially \((d_2)\), or if it does not fulfill the restrictions \((d_3)\), in the last two cases the data set could not be used as input of the selected service. However, in case \(d_2\) could result in a further analysis of matchmaking requirements, induced by the user. (Figure 2.5)
Chapter 3

Knowledge Representation

3.1 Ontologies

The best known definition of an ontology is “a specification of a conceptualization” [Gru93].

An ontology is a formal description of the entities, concepts and relationships that can exist in some field of knowledge. We say it is formal because it has mathematical semantics. An ontology however will always be a simplified view of the world.

The use of an ontology in the field of computer science allows different users to share information. With an ontology we want to represent knowledge, which thanks to the formal representation, could be processed in an “intelligent” way by software applications. By intelligent we mean that the application should be able to find implicit consequences from what is explicitly stated in the ontology.

To define an ontology we have to [NM]:

- Identify the concepts in our domain of discourse, also known as classes
- Associate properties with each concept, describing the particular characteristics of that concept.
- Create restrictions in some of the properties to describe the concepts better.
- Define the relations between different concepts.
- Associate individuals with the concept that represents them better.

To design an ontology, first we have to define the field of knowledge our ontology will describe. In this study, our domain are the concepts that define geographic data sets and geographic web services.

The next step is to define the purpose of use of our ontology. In this research the goal is to create descriptions of data sets and web services. These descriptions should allow a user to determine the usefulness of data sets for geographic web services. The question that a software application based on our ontology must be able to answer is: Can I use data set X with the service Y?
3.2 Description Logics

A common approach to represent knowledge is to express it in the way of a network graph, as in Figure: 3.1. Commonly the nodes represent the concepts, and the links represent the relations between concepts. The concepts usually have attributes attached to them, representing particular characteristics of the concepts.

![Figure 3.1: An example of a network graph representing knowledge](image)

Description Logics (DL) is a formalism for Knowledge Representation (KR). It is a formal representation of the knowledge stored in the graphs. It represents the knowledge of an application domain, defining the concepts of that domain, the terminology, and later specifying properties for those concepts and for the individuals associated with them. Description Logics uses formal Tarski-style semantics, which allows a software application that uses them, to extract implicit facts from what it is explicitly declared.

The basic elements used by Description Logics are [BMNPS02]:

- Atomic Concepts (unary predicates, they represent sets of individuals)
- Atomic Roles (binary predicates, they represent relations between individuals), and
- Individuals.

A software application, designed to handle a knowledge base (KB), is called a knowledge representation system. A knowledge base is composed of two main parts (Figure: 3.2):

- The TBox (Terminology Box) contains the concepts identified in the field of knowledge that we are studying. These concepts form a vocabulary (terminology) of the application domain.
- The ABox (Assertions Box) contains statements about individuals using the terms declared in the vocabulary. In an ABox, we declare individuals, by giving them names, and making assertions on them.
3.2.1 Basic Syntax

In abstract notation, we use the letters $A$ and $B$ for Atomic Concepts, the letter $R$ for Atomic Roles, and the letters $C$ and $D$ for concept descriptions.

The basic description language elements ($AL$):

$$C, D \rightarrow A \mid (\text{Atomic Concept})$$

$$\top \mid (\text{Universal Concept})$$

$$\bot \mid (\text{Bottom Concept})$$

$$\neg A \mid (\text{Atomic Negation})$$

$$C \sqcap D \mid (\text{Intersection})$$

$$\forall R.C \mid (\text{Value Restriction})$$

$$\exists R.\top \mid (\text{Limited existential quantification})$$

**Example 1**

We are going to declare the following entities:

- *DataSet* and *CI_ResponsibleParty* as Concepts.
- *provided by* as a Role.
- *NL_ROADS* and *GURU* as Individuals.\(^1\)

\[
\forall \text{provided by, C1_ResponsibleParty} \quad \text{There is a set of individuals that have the Role provided by involving only individuals from the concept C1_ResponsibleParty.}
\]

\[
\text{C1_ResponsibleParty(GURU)} \quad \text{GURU is a member of the set of objects known as C1_ResponsibleParty}
\]

\[
\text{DataSet(NL_ROADS)} \quad \text{NL_ROADS is a member of the set of objects known as DataSet}
\]

\[
\text{provided by(NL_ROADS, GURU)} \quad \text{NL_ROADS has a provider identified as GURU}
\]

\(^1\)We use the notation introduced in [BMNPS02]. In the case of concepts the first character is upper case, in the case of individuals, the whole name is upper case and in the case of roles the whole name is lower case.
3.2.2 Formal Semantics

To define the semantics of the language, we consider interpretations $I$ which consists of a non empty set $\Delta^I$ (the domain of the interpretation).

<table>
<thead>
<tr>
<th>constructor</th>
<th>syntax</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Concept</td>
<td>$A$</td>
<td>$A^I \subseteq \Delta^I$</td>
</tr>
<tr>
<td>Atomic Role</td>
<td>$R$</td>
<td>$R^I \subseteq \Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>Universal Concept</td>
<td>$\top$</td>
<td>$\top^I = \Delta^I$</td>
</tr>
<tr>
<td>Bottom Concept</td>
<td>$\bot$</td>
<td>$\bot^I = \emptyset$</td>
</tr>
<tr>
<td>Atomic Negation</td>
<td>$\neg A$</td>
<td>$(\neg A)^I = \Delta^I \setminus A^I$</td>
</tr>
<tr>
<td>Intersection</td>
<td>$C \cap D$</td>
<td>$(C \cap D)^I = C^I \cap D^I$</td>
</tr>
<tr>
<td>Value Restriction</td>
<td>$\forall R.C$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Limited existential quantification</td>
<td>$\exists R. \top$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Union</td>
<td>$C \cup D$</td>
<td>$(C \cup D)^I = C^I \cup D^I$</td>
</tr>
<tr>
<td>Full $\exists$</td>
<td>$\exists R.C$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>At least cardinality restriction</td>
<td>$(\geq nR)$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>At most cardinality restriction</td>
<td>$(\leq nR)$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Exact cardinality restriction</td>
<td>$(= nR)$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Negation</td>
<td>$\neg C$</td>
<td>$\Delta^I \setminus C^I$</td>
</tr>
<tr>
<td>Concrete domain exactly restriction</td>
<td>$(= n R)$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Concrete domain min restriction</td>
<td>$(\geq n R)$</td>
<td>${ a \in \Delta^I</td>
</tr>
<tr>
<td>Concrete domain max restriction</td>
<td>$(\leq n R)$</td>
<td>${ a \in \Delta^I</td>
</tr>
</tbody>
</table>

3.2.3 Reasoning:

A knowledge base contains information in an explicit form. Using this information, it should be possible to obtain implicit, new information. There are two kinds of reasoning in a knowledge representation based on Description Logics.

- Reasoning with the Terminology (TBox): The terminology is the set that contains all the concepts, also known as classes, in our knowledge base.

  We identify our TBox as $T$.

  If $I$ satisfy all definitions in $T$, then $I$ is model of $T$.

  There are four kinds of reasoning using the concepts:
Satisfiability: \( C \) is satisfiable with respect to \( T \) if there exists a model \( I \) of \( T \) such that \( C^I \) is nonempty.

Subsumption: \( C \sqsubseteq D \) \( C \) is subsumed by \( D \) with respect to \( T \) if \( C^I \subseteq D^I \). The concept \( D \) is more general than the concept \( C \). In this case we write \( C \sqsubseteq_T D \) or \( T \models C \subseteq D \).

Equivalence: \( C \equiv D \) \( C \) and \( D \) are equivalent with respect to \( T \) if \( C^I = D^I \).

Disjointness: \( C \text{ disjoint } D \) \( C \) and \( D \) are disjoint if \( C \cap D = \emptyset \).

- Reasoning with the Assertions (ABox): The Assertions is the set of all the statements about individuals, the instances of the concepts.

We identify our ABox as \( A \)

We can define \( A \), if \( I \) satisfies all assertions in \( A \) while at the same time is a model for the TBox \( T \).

There are two kinds of assertion in \( A \).

- Concept Assertions: \( C(a) \) if \( a^I \in C^I \) If the individual \( a \) belongs to the concept \( C \)

- Role Assertions: \( R(a, b) \) if \( (a^I, b^I) \in R^I \) If the role \( R(a, b) \) exists in \( I \)

### 3.2.4 Algorithms

There are two kinds of algorithms to work with a knowledge base, the first kind operates on the concepts inferring relations between them. The second kind operates on the relations between concepts and individuals.

- **Structural Subsumption Algorithms**: The classification of concepts determines the concept/subconcept relationships, also known as subsumption. This relationship is commonly used to infer facts.

**Proposition 1**

*Let \( C \) and \( D \) be concept descriptions: \([BMNPS02]\)*

\[
C = A_1 \cap \ldots \cap A_m \cap \forall R_1. C_1 \cap \ldots \cap \forall R_n. C_n
\]

\[
D = B_1 \cap \ldots \cap B_k \cap \forall S_1. D_1 \cap \ldots \cap \forall S_l. D_l
\]

We can state that \( C \sqsubseteq D \) if the following two conditions hold:

- for all \( i, 1 \leq i \leq k \), there \( j, 1 \leq j \leq m \) such that \( B_i = A_j \) and

- for all \( i, 1 \leq i \leq l \), there exists \( j, 1 \leq j \leq n \) such that \( S_i = R_j \) and \( C_j \sqsubseteq D_i \).

- **Concept Membership Algorithm**: To infer if a given individual is an instance of a concept, we have to analyze the assertions made about this individual, and compare them to the structure of the concept.
Typically, a concept is defined using the operands Intersection (\(\cap\)) and Union (\(\cup\)) that chain syntactically smaller components that together define the concept.

In the case of a concept defined by an intersection (\(\cap\)), an individual that belongs to that concept must also belong to all the members involved in the intersection:

**Proposition 2**

Let \(C\) be a Concept description:

\[
C \equiv A_1 \cap \cdots \cap A_m \cap \\
\forall R_1.C_1 \cap \cdots \forall R_n.C_n \cap \\
(=_{v R_1}) \cap \cdots \cap (=_{v R_o})
\]

Let \(a\) be an individual with the following assertions:

\[
B_1(a), \cdots, B_j(a) \\
P_1(a, b_1) \cdots P_k(a, b_k)
\]

To infer that \(a\) is an instance of \(C\), we must have assertions that indicate that \(a\) is a member of all of \(C\)'s components.

for all \(x\), \(1 \leq x \leq m\), there exists \(y\), \(1 \leq y \leq j\) such that \(B_y \subseteq A_x\) and

for all \(p\), \(1 \leq p \leq n\), there exists \(q\), \(1 \leq q \leq k\) such that

\(R_p = P_q\) and \(\forall P_q(a, b_q) \ C_p(b_q)\) and

for all \(p\), \(1 \leq p \leq n\), there exists \(q\), \(1 \leq q \leq k\) such that

\(R_o = P_q\) and in \(P_q(a, b_q)\) \(b_q = v\)

**Example 2**

Suppose that in our domain we have the following atomic concepts:

\(\text{DataSet, CI_ResponsibleParty}\)

And the following atomic roles:

\(\text{has\_row\_count, has\_col\_count, has\_object\_count, provided\_by, has\_reference\_system}\)

Using the previous concepts, we can declare complex concepts, using the atomic roles:
Chapter 3. Knowledge Representation

\[(01A)\quad \text{DataSet} \subseteq T\]
\[(01B)\quad \text{CI\_ResponsibleParty} \subseteq T\]

\[(02)\quad \text{Vector} \subseteq \text{DataSet}\]
\[(02A)\quad \text{Vector} \equiv \text{DataSet} \cap \forall\text{has\_object\_count}.\text{Integer}\]

\[(03)\quad \text{Grid} \subseteq \text{DataSet}\]
\[(03A)\quad \text{Grid} \equiv \text{DataSet} \cap \forall\text{has\_row\_count}.\text{Integer} \cap \forall\text{has\_col\_count}.\text{Integer}\]

\[(04)\quad \text{TestConcept} \equiv \text{DataSet} \cap \forall\text{has\_object\_count}.\text{Integer} \cap \forall\text{has\_reference\_system}.\text{String} \cap \forall\text{provided\_by}.\text{CI\_ResponsibleParty}\]

In (02) we stated that every Vector is a Dataset. A more accurate definition of Vector is the set of elements resulting of the intersection of Dataset and the characteristics that differentiate Vector from the other Dataset (02A).

In (03) and (03A), in a similar way as in (02) and (02A), we define the concept Grid, however with different roles.

In (04) we define a new concept, TestConcept, as the set of elements that result from the intersection of Dataset and the elements that have a role has\_object\_count.\text{Integer} with value \text{Integer}, and a role has\_reference\_system with a value \text{String}.

To determine the relations of TestConcept with the other members of the TBox we would have to analyze its structure using [Proposition 1]:

\[\begin{array}{ccc}
\text{TestConcept} & \subseteq & \text{Vector} \\
\text{DataSet} \cap & \text{DataSet} \cap & ???
\end{array}\]

\[\begin{array}{ccc}
\forall\text{has\_object\_count}.\text{Integer} \cap & \forall\text{has\_object\_count}.\text{Integer} \\
\forall\text{has\_reference\_system}.\text{String} \\
\forall\text{provided\_by}.\text{CI\_ResponsibleParty}
\end{array}\]

Using [Proposition 1] we can state that: TestConcept \subseteq Vector holds. Vector is a more abstract concept than TestConcept.

\[\begin{array}{ccc}
\text{TestConcept} & \equiv & \text{Vector} \cap \\
\forall\text{has\_reference\_system}.\text{String} \cap \\
\forall\text{provided\_by}.\text{CI\_ResponsibleParty}
\end{array}\]

We can also determine whether TestConcept \subseteq Grid holds:

\[\begin{array}{ccc}
\text{TestConcept} & \subseteq & \text{Grid} \\
\text{DataSet} \cap & \text{DataSet} \cap & ???
\end{array}\]

\[\begin{array}{ccc}
\forall\text{has\_object\_count}.\text{Integer} \cap & \forall\text{has\_row\_count}.\text{Integer} \cap \\
\forall\text{has\_reference\_system}.\text{String} \\
\forall\text{provided\_by}.\text{CI\_ResponsibleParty}
\end{array}\]

Using [Proposition 1] we can state that: TestConcept \subseteq Grid does not hold. Grid is not a more abstract concept of TestConcept.
3.3 Working Knowledge Base

In this research, we are focus on the concepts that define the descriptions of geographic data sets, and geographic web services.

As a starting point we use the standards ISO19115, for the definition of the general metadata, and ISO19119, for more specific information about the geographic service architecture.

This research has as objective to identify guidelines for the use of web-based ontologies for the description of geographic web services and data sets, for that reason the ontology will be simplify, and some elements described in the ISO standards will not be used. However, we try to include all the elements considered core metadata, the minimum elements required for the identification of data sets and geographic services.

We identify three basic concepts, that are the base of our terminology (TBox). These are:

*Service*  It is the abstract representation of any geographic web service.

*DataSet*  It is the representation of any set of geographic information. The classification of sub concepts of *DataSet* is based on the MD_SpatialRepresentation package described in the document ISO19115. However, we add extra sub concepts in order to have better test scenarios for the algorithms described in Section 3.2.4.

*List*  Represents a finite enumeration of individuals, that are used as values in the description of any *Service* or any *DataSet*.

Based on these three basic concepts we construct a set of terms.
Chapter 3. Knowledge Representation

Service ⊑ ⊤
DataSet ⊑ ⊤
List ⊑ ⊤

MD_ReferenceSystem ≡ List □
∀ projection\_String □
∀ datum\_String □
∀ ellipsoid\_String

CI_ResponsibleParty ≡ List □
∀ address\_String □
∀ phone\_String □
∀ organization\_name\_String □

MD_TopicCategoryCode ⊑ List
MD_TopologyLevelCode ⊑ List

DataSet ≡ T □
∀ title\_String □
∀ provided\_by\_CI_ResponsibleParty □
∀ has\_topic\_category\_MD_TopicCategoryCode □
∀ has\_reference\_system\_MD_ReferenceSystem

MD_GridSpatialRepresentation ≡ DataSet □
∀ column\_count\_Integer □
∀ row\_count\_Integer □
∀ number\_of\_dimensions\_Integer

Remote\_Sensing\_Image ≡ MD_GridSpatialRepresentation □
∀ number\_of\_bands\_Integer
∀ quantization\_Integer □
∀ ground\_pixel\_size\_Integer

Passive\_Sensor\_Image ⊑ Remote\_Sensing\_Image
Thermal\_Scanner\_Image ⊑ Passive\_Sensor\_Image
Aerial\_Photo ⊑ Passive\_Sensor\_Image

Active\_Sensor\_Image ⊑ Remote\_Sensing\_Image
Laser\_Scanner\_Image ⊑ Active\_Sensor\_Image
Radar\_Image ⊑ Active\_Sensor\_Image

MD_VectorSpatialRepresentation ≡ DataSet □
∀ has\_topology\_level\_MD_TopologyLevelCode □
∀ has\_geometry\_object\_count\_Integer

Complexes ⊑ MD_VectorSpatialRepresentation
Composites ⊑ MD_VectorSpatialRepresentation
Curves ⊑ MD_VectorSpatialRepresentation
Point ⊑ MD_VectorSpatialRepresentation
Solid ⊑ MD_VectorSpatialRepresentation
Surface ⊑ MD_VectorSpatialRepresentation

Service ≡ T □
∀ service\_name\_String □
∀ service\_provided\_by\_CI_ResponsibleParty

Communication\_Service ⊑ Service
Human\_Interaction\_Service ⊑ Service
System\_Management\_Service ⊑ Service
Workflow\_Task\_Management\_Service ⊑ Service
Model\_Information\_Management\_Service ⊑ Service

Processing\_Service ≡ Service □
∀ has\_operation\_String

Metadata\_Processing\_Service ⊑ Processing\_Service
Temporal\_Processing\_Service ⊑ Processing\_Service
Thematic\_Processing\_Service ⊑ Processing\_Service
Spatial\_Processing\_Service ⊑ Processing\_Service
3.3. Working Knowledge Base

Each of the sub concepts of \textit{Processing Service} has its own sub concepts. For example the concept \textit{Spatial Processing Service} has as sub-concepts, among others:

- \textit{Coordinate Conversion Service}
- \textit{Proximity Analysis Service}
- \textit{Positioning Service}
- \textit{Route Determination Service}

The complete list of concepts in a similar hierarchy position is big, and it is not included here for reasons of clarity.

3.3.1 Individual Descriptions

Using the concepts defined in our TBox, we can describe specific objects, stating that they belong to certain class, and then assigning values to the roles associated to that concept. The set that contains all the assertions made using the vocabulary elements of the TBox, is known as ABox (A for Assertions). The knowledge base is composed of the TBox and ABox.

These are sample assertions from a ABox, using the previous, defined vocabulary.

\textbf{Example 3}

\begin{verbatim}
MD_TopicCategoryCode(FARMING)
MD_TopicCategoryCode(TRANSPORTATION)

MD_TopologyLevelCode(FULL_PLANAR_GRAPH)
MD_TopologyLevelCode(GEOMETRY_ONLY)

MD_ReferenceSystem(ETRS89)
datum(ETRS89, “ETRS 89”)
ellipsoid(ETRS89, “WGS 84”)
projection(ETRS89, “conic”)

CI_ResponsibleParty(GURU)
address(GURU, “Guru House”)
phone(GURU, “+31(0)53 487 44 44”)
organization_name(GURU, “GuruSoft”)

MD_VectorSpatialRepresentation(NLROADS)
title(NLROADS, “Roads in Netherlands”)
has_topic_category(NLROADS, TRANSPORTATION)
has_reference_system(NLROADS, ETRS89)
has_topology_level(NLROADS, GEOMETRY_ONLY)
has_geometry_object_count(NLROADS, 5)
\end{verbatim}

3.3.2 The Processing Service Descriptions

The definition of the concept \textit{Processing Service}, brings a special problem. The members of this kind of services characterize themselves by the fact that they require input and output to perform certain \textit{operations}. Intuitively we know that the description of the \textit{allowed inputs} of a service, and its \textit{expected outputs},
must be considered as concepts rather than single individuals [LH03]. (Figure: 3.3). These concepts are sub-concepts of DataSet. We can define the *allowed inputs*, as the set of data sets that can be processed in an operation of a service. In the case of the *expected outputs*, these are the set of expected results of that operation.

The description of a service must reflect this relation between the *allowed inputs* and the *expected outputs*. We refer to this relation as functionality.

\[ \text{Allowed Input} \rightarrow \text{Expected Output} \]

We suggest that the best way to declare this relation is with a *Role*. We can state that there is a link between members of two given concepts. To do this, we define the *domain* and the *range* of this *Role*. The domain is the *input* and the range is the *output*. In this way we can model the functionality of a certain service.

\[
\begin{align*}
\text{Processing Service}(s) & \quad s \text{ is an instance of the concept Processing Service} \\
\text{Functionality} & \quad \text{Functionality is a Role} \\
\text{Allowed Input} \subseteq \text{DataSet} & \quad \text{Allowed Input is a sub concept of DataSet} \\
\text{Expected Output} \subseteq \text{DataSet} & \quad \text{Expected Output is a sub concept of DataSet} \\
\forall \text{Functionality}(a, b) \quad \text{Allowed Input}(a) \text{ and Expected Output}(b) & \quad \text{The domain of } R \text{ is Allowed Input} \\
& \quad \text{The range of } R \text{ is Expected Output}
\end{align*}
\]

In a next step we take the identifier of the role and transform it to a *string*. This *string* is assigned as a value for the role *has_operation*. To do this, we create a function that performs this operation:

\[
\begin{align*}
\text{Functionality} & \rightarrow \text{“Functionality”} \\
\text{has_operation}(s, \text{“Functionality”}) & \quad \text{We state that the individual } s \text{ has in its role } \text{has_operation} \text{ the string value } \text{string value “Functionality”}
\end{align*}
\]

We think that by doing this, we can associate the functionality of a service to its description. Any inference will still be made in the actual role. The string value will be used only to identify the role that represents the functionality of the service. It will not be used in any case to infer anything.

The advantage of this procedure is that we can handle the services as regular instances of a concept, and at the same time being able to identify the properties that represent their functionality.

**Example 4**

Suppose that there is one specific service, which we call *MY_SERVICE*, it is an individual of concept *Processing Service*

\[
\text{Processing Service}(\text{MY_SERVICE})
\]

To indicate that this specific service operates on data sets of type *Point*, to
3.3. Working Knowledge Base

Figure 3.3: A service description involves entities that belong to the ABox and to the TBox, clearly the description of the input of a service, is not an individual, but a concept that produce as result a data set of type MD_GridSpatialRepresentation we can define a role that defines that functionality.

\[ \text{Point} \rightarrow \text{MD\_GridSpatialRepresentation} \]

Operation01 is a role.

\[ \forall \text{Operation01}(a, b) \]

For all the relations defined by Operation01

Point(b)

a is an individual member of concept Point

MD_GridSpatialRepresentation(b) and

b is an individual member of concept MD_GridSpatialRepresentation

Later we can take the identifier of Operation01 and transform it to the string “Operation01” and assign this value to the role has_operation of the individual MY\_SERVICE.

\[ \text{has\_operation(MY\_SERVICE, “Operation01”)} \]

The individual identified as MY\_SERVICE has a value “Operation01” in its role has_operation.
3.3.3 Definition of the Input and Output Concepts

In the previous section we identified one concept, called the *Allowed Input* of certain operation, as the *Domain* of a role that identifies that operation. This concept comprises all the data sets, that could be processed by the service in that operation. However it would be unusual that the already existing concepts in our terminology, could be enough for an accurate definition of this concept.

To better determine the allowed input of a processing service, we suggest that the service provider should define this concept by himself, and *on the fly*. The *Allowed Input* should be a sub concept of *DataSet*. In fact *Allowed Input* should be always subsumed by *DataSet*, but it can be specified according to the service provider criteria.

We suggest that a good starting point for the definition of this concept is an existing concept; this in most of the cases, will be the first condition in the definition. Later we can add role restrictions, until the result satisfies the constraints of the service. The most suitable constraints for this purpose, are the concrete domain restrictions. Using this restrictions we could clearly specify the requirements a data set must fulfill to be considered suitable to be used with certain service.

The concrete domain restriction are three:

- Concrete domain exactly restriction ($=_{n} R$)
- Concrete domain min restriction ($\geq_{n} R$)
- Concrete domain max restriction ($\leq_{n} R$)

A formal semantics for these restrictions can be found in section 3.2.2. The purpose of this, is to define a special new concept. This concept is defined by intersection of all the requirements needed to be consider an allowed input of a service.

**Example 5**

*Allowed Input Concept Definition:* We assume that our service only operates on data sets that fulfill the following conditions, the data set:

**Condition 01** must be *MD_VectorSpatialRepresentation*, and

**Condition 02** must have a *GEOMETRY_ONLY* topology, and

**Condition 03** must have as reference system GRS80, and

**Condition 04** must have at least 5 elements.

**Definition of the conditions for the Allowed Input Concept:**

\[ \text{Allowed Input} \equiv \text{MD\_VectorSpatialRepresentation} \cap
\quad =_{\text{GEOMETRY\_ONLY}} \text{has\_topology\_level} \cap
\quad =_{\text{GRS80}} \text{has\_reference\_system} \cap
\quad \geq_{5} \text{has\_geometry\_object\_count} \]
3.3.4 Matching a Data set with a Service Input

To infer whether a certain specific data set can be used for a specific web service, we have to verify if our data set can be considered a member of the class defined as the allowed input data sets for that service.

The process consists in gathering all the assertions about the individual (the specific data set), and comparing these assertions against the conditions that define the set of allowed inputs.

To state if an certain individual belongs to certain class, we are going to use proposition 2.

Example 6

We consider two data set NL_ROADS and UK_ROADS
Chapter 3. Knowledge Representation

MD_VectorSpatialRepresentation(NL_ROADS)

Individual: NL_Roads

title(NL_ROADS, “Roads in Netherlands”) has_topic_category(NL_ROADS, TRANSPORTATION) has_reference_system(NL_ROADS, ETRS89) has_topology_level(NL_ROADS, GEOMETRY_ONLY) has_geometry_object_count(NL_ROADS, 5)

Curves(UK_ROADS)

Individual: UK_Roads

title(UK_ROADS, “Roads in UK”) has_topic_category(UK_ROADS, TRANSPORTATION) has_reference_system(UK_ROADS, GRS80) has_topology_level(UK_ROADS, GEOMETRY_ONLY) has_geometry_object_count(UK_ROADS, 10)

Test01: Allowed_Input(NL_ROADS)

Result: Allowed_Input(NL_ROADS) does not hold

Test02: Allowed_Input(UK_ROADS)

Result: Allowed_Input(UK_ROADS) does hold

In Test01, we see that the data set NL_ROADS fulfills all the conditions of the web service except for the third one. This data set has a property identified as has_reference_system. This property has a value ETRS89. All the elements part of the concept Allowed_Input must have a value in that property equal to GRS80. Because of this reason this data set is not a member of the class Allowed_Input.

In Test02, we can see that there are assertions about the data set UK_ROADS, that make it satisfy all the conditions to be a member of the class Allowed_Input. This data set is member of the class Curves, but this class is a subclass of MD_VectorSpatialRepresentation, then UK_ROADS is also a member of this class.

3.4 Description Languages based on XML

XML is a metalanguage, used to define markup languages. Using XML, any user is allowed to create his own markup language, which provides great flexibility. A document with markups provides identification to the information, by using these markups as identifiers.

Documents using XML can contain any kind of structured information. Basically a XML document is a text format file, with a structure. Because of the format, this information could be shared between different computing systems,
that in other case could not communicate with each other. XML is a non proprietary format, the advantage of using it is that anybody can process it, and the user does not need any special application to deal with it [Pal01].

The advent of XML and the growth of the Web have caused the amount of users and information to become huge. As a consequence we are in danger of been flooded by irrelevant information. The amount of information and the number of sources of this information is so big, that we can hardly handle it [DJS03].

W3C tries to face this problem through the Semantic Web. It is an attempt to provide the information with meaning, in the sense that the format of information could be processable by computers, in a way that may look intelligent. In this way with the help of computers, the human users would be able to control the information overload. With the Semantic Web, we want to structure the information on the web, in a format that could represent knowledge and that could be processable by computer applications.

Current languages based on XML, designed to represent knowledge, are based on Description Logics. The main advantage of using Description Logics, is the sound algorithms for subsumption and membership that already exists [GCDT01]. These languages use Uniform Resource Identifiers (URI) to identify resources, a resource is anything that can be identified. An example of URI is:

http://www.gfm2.org#myDataset

Then we can describe this resource, identifying its relations, and assigning other URIs to this relations to identify them, and associating values, or other URIs to those relations. In this way, we can create statements composed by three elements. A language based on this principle is Resource Description Framework (RDF).

### 3.4.1 RDF & RDFS

RDF was developed by W3C as part of its Semantic Web effort. It started as an extension of PICS, a W3C recommendation, to rate services and systems.

To describe things using RDF, first we have to identify the thing we want to describe using an URI, and then we make statements assigning values to the properties that define it. Each one of these statements is called a triple, because it has three elements:

```xml
<subject><object><predicate>
```

This kind of structure is equivalent to a role statement in Description Logics ($R(a, b)$, where $R$ is a role, $a$, and $b$ are description concepts).

### Example 7

For example: We have a resource, in this case a data set, and we want to state that it has reference system Amersfoort.

The first step would be to identify it, using a URI. Then we can assign one URI for each relation it has, and other URIs or constants to the values associated to those properties (Figure: 3.5):
Chapter 3. Knowledge Representation

http://www.gfm2.org#myDataset
(A Resource we want to describe)

http://www.gfm2.org#has_reference_system
(A relation that this resource has)

http://www.gfm2.org#Amersfoort
(A value for the relation)

Figure 3.5: RDF graph of the resource http://www.gfm2.org#myDataset showing two of its relations.

RDF uses URI’s to make its statements. Using RDF, a user can make simple statements about resources. The most common format of these statements is in a well formatted XML document. To do this, we use a RDF/XML serialization.

When several related URI’s share the same prefix, we can define a namespace. In an XML document we can declare a namespace by:

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:gfm2="http://www.gfm2.org">
```

Using a name space, we can state that:

http://www.gfm2.org#myDataset

is equivalent to:

gfm2:myDataset

**Example 8**
The statements contained in the graph represented in Figure: 3.5 would look like:

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:gfm2="http://www.gfm2.org">
```
3.4. Description Languages based on XML

Using RDF, a user can make simple statements about resources. However, it is not enough to associate resources in classes and state relations between these classes, or to declare properties and specify the domain and the range of these properties. This is because concepts like class, Domain and Range have not been defined in RDF.

To solve the limitations of RDF, W3C released RDF Schema (RDFS). It is a collection of RDF resources that can be used to describe other RDF resources [W3C02]. It conforms a vocabulary, that can be used to improve the descriptions done with RDF. It was designed to provide the user of RDF, with the necessary elements to describe more complex scenarios, that involve the use of classes and properties.

**rdfs:Class**

Using RDF Schema, we can state that there is a set of individuals with similar characteristics, this is the definition of a Class. Its definition is similar to the term Concept, used in Description Logics.

**Example 9**

This is the declaration of some classes of individuals using RDF/XML:

```xml
<rdf:Description rdf:ID="Dataset">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="Vector_Spatial_Representation">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="MD_ReferenceSystem">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
```

**rdfs:subClassOf**

Different set of individuals can have relations between them, one of this relations is inclusion. If we have two sets, it is possible that one of them, includes all the members of the second.

This kind of relation is declared in RDFS, as a subclass relation between two classes. In meaning is similar to the $C \subseteq D$ construction in Description Logics.

**Example 10**

This is the declaration of subclass between two classes of individuals:

```xml
<rdfs:Class rdf:ID="Vector_Spatial_Representation">
  <rdfs:subClassOf rdf:resource="DataSet"/>
</rdfs:Class>
```
rdfs:domain & rdfs:range

Using RDF, we can declare a property. Using RDFS we can extend this capability, by defining the domain and range of that property. In Description Logics this is equivalent to:

\[ \forall R(a, b) \]

\[ \text{rdfs:domain } C(a) \]

\[ \text{rdfs:range } D(b) \]

Where \( R \) is a role, and \( a \) and \( b \) are individuals

Where \( C \) represents the set of individuals for which the relation holds

Where \( D \) represent all the allowed values for this relation

Example 11

This is the specification of the domain and the range for some properties.

```xml
<rdf:Property rdf:ID="has_geometry_object_count">
    <rdfs:domain rdf:resource="#Vector_Spatial_Representation"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="has_reference_system">
    <rdfs:domain rdf:resource="#DataSet"/>
    <rdfs:range rdf:resource="#MD_ReferenceSystem"/>
</rdf:Property>
```

Besides RDFS there exist other vocabularies like DAML, OIL and OWL. DAML (DARPA Agent Markup Language) was created as part of a research program that started in August 2000. This program was sponsored by DARPA, a US military government research organization. It is very similar to OIL (Ontology Inference Layer), this is a vocabulary result of a research project sponsored by the European Union programme for the Information Society Technologies. These vocabularies were so similar that finally merge becoming DAML+OIL. The main objective of them was to solve the findability problem, give support to e-commerce and enable knowledge management [Gar01].

Later W3C introduced OWL (Web Ontology Language), which is based on DAML+OIL, and takes in account the experiences earned with it.

3.4.2 OWL

On December 15, 2003, W3C presented OWL (Web Ontology Language) as a Proposed Recommendation. According to W3C, OWL is designed to be used by applications that need to process the content of information rather than just presenting the information to humans. OWL can represent the main elements of some field, describing its characteristics, and the relations that exist between them. It has more facilities for expressing semantics than XML, RDF, and RDFS.

With OWL, it is possible to define in a formal way a field of knowledge. By formal we mean that our definitions and statements, conforms to accepted conventions. In the context of knowledge sharing that set of definitions and statements is known as an ontology. ontology (section 3.1).
OWL has three sublanguages, with increasing degrees of expressiveness [W3C04]:

**OWL Lite** It was designed for those users requiring classification hierarchy and simple constraints. It does not allow the use of some primitives available on other sublanguages of OWL.

**OWL DL** It was designed to allow more expressive statements. DL stands for Description Logic. It provides the primitives to construct statements processable by reasoning systems using Description Logics.

**OWL Full** It was designed for those users that require maximum expressiveness. It allows the syntactic freedom of RDF, however without computational guarantees. In OWL Full we can treat a class at the same time as an individual and as a set of individuals. There is no reasoning system that is able to support all the features of OWL Full.

Each of the sublanguages is an extension of the previous one, any legal statement of OWL Lite is a legal statement of OWL DL, and any legal statement using OWL DL is a legal statement of OWL Full, but that is not true in the other direction.

In our research we are going to use OWL DL. This language provides us with primitives that allow us to make expressive descriptions that can be computable.

### 3.4.3 Some Interesting Elements of OWL DL vocabulary

**OWL properties:**

In the example of *myDataset* we notice that the range of the properties could be a constant value or the instance of some class. OWL provides us with the resources to differentiate those two kinds of properties. In fact, OWL defines two kinds of properties:

**Object Property** Defines a property that associates instances of one *domain class* and instances of a given *range class*.

In Description Logics:

\[
\forall R(a, b) \quad \text{Where } R \text{ is a role, and } a \text{ and } b \text{ are individuals}
\]

\[
rdfs:range \quad D(b) \quad \text{Where } D \text{ is a class (range)}
\]

**Data type property** Defines a property that associates instances of one *domain class* and values of an XML Schema data type.

In Description Logics:

\[
\forall R(a, b) \quad \text{Where } R \text{ is a role, and } a \text{ and } b \text{ are individuals}
\]

\[
rdfs:range \quad D(b) \quad \text{Where } D \text{ is XML Schema data type}
\]

The complete list of the recommended data types for use with OWL can be found in [W3C03].
Example 12
In the following example we declare an object property identified as:
has_reference_system
This property exists between instances of the class DataSet and instances of the class GeoCoordSystem.

<owl:ObjectProperty rdf:ID="has_reference_system">
  <rdfs:domain rdf:resource="#DataSet">
  <rdfs:range rdf:resource="#MD_ReferenceSystem">
</owl:ObjectProperty>

In this example we declare a property has_geometry_object_count that associates instances of MD_VectorSpatialRepresentation and a number, in this case a positive integer.

<owl:DatatypeProperty rdf:ID="has_geometry_object_count">
  <rdfs:domain rdf:resource="#Vector_Spatial_Representation">
  <rdfs:range rdf:resource="&xsd;positiveInteger">
</owl:DatatypeProperty>

Restrictions on Properties:

We can make statements that affect the range of a property. The result of such statements is a class of individuals that satisfy that restriction. There are two kinds of restrictions: value constraints and cardinality constraints.

Value Constraints

owl:allValuesFrom Defines the set of individuals, for which the all the values of the restricted property, are instances of a certain class.
In Description Logics is represented as (∀R.C), see section 3.2.2).

owl:someValuesFrom Defines the set of individuals that have at least one relation with an instance of a certain class.
In Description Logics is represented as (∃R.C)

owl:hasValue Defines the set of individuals for which the value of the restricted property is equal to certain instance.
In Description Logics is represented as (=n R) see section 3.2.2).

Cardinality Constraints owl:maxCardinality Defines the set of individuals that have as a most N distinct values of the property concerned.
In Description Logics is represented as (≤nR) see section 3.2.2).

owl:minCardinality Defines the set of individuals that have as a least N distinct values of the property concerned.
In Description Logics is represented as (≥nR)

owl:cardinality Defines the set of individuals that have as an exact N distinct values of the property concerned.
In Description Logics is represented as (=nR).
Example 13

In this example we create the set of individuals that have a reference system ETRS89.

<owl:Restriction>
  <owl:onProperty rdf:resource="#has_reference_system">
    <owl:hasValue rdf:resource="#ETRS89"/>
  </owl:Restriction>

In the following example we use hasValue restriction, to create the set individuals that have as a value for the property address the string Enschede.

<owl:Restriction>
  <owl:onProperty rdf:resource="#address">
    <owl:hasValue rdf:resource="#"/>
  </owl:Restriction>

Using the cardinality constraints, we can define the set of individuals that have at least 5 telephones.

<owl:Restriction>
  <owl:onProperty rdf:resource="#phone">
    <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
      5
    </owl:minCardinality>
  </owl:Restriction>

Using the cardinality constraints, we can define the set of individuals that have at most 2 telephones.

<owl:Restriction>
  <owl:onProperty rdf:resource="#phone">
    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
      2
    </owl:maxCardinality>
  </owl:Restriction>

Or that have exactly 3 telephones.

<owl:Restriction>
  <owl:onProperty rdf:resource="#phone">
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
      3
    </owl:cardinality>
  </owl:Restriction>

Class Axioms

An axiom is a proposition, that does not require formal demonstration to prove its truth, and it is received as mentioned.

In OWL we have three classed of axioms:

subClassOf Using this axiom, we can define one class as an extension of another.

38
**equivalentClass** This axiom allows us to declare that a class description has the same extension as another class description.

**disjointWith** Using the disjoint axiom, we can declare that two classes have no members in common.

**Example 14**

*In this example we declare that the class Point is a subclass of Vector.*

```xml
<owl:DatatypeProperty rdf:ID="#Point">
  <rdfs:subclassOf rdf:resource="#Vector"/>
</owl:DatatypeProperty>
```

*In the following example we declare that the MD_VectorSpatialRepresentation class has no common members with the class MD_VectorSpatialRepresentation.*

```xml
<owl:Class rdf:about="#Vector_Spatial_Representation">
  <owl:disjointWith rdf:resource="#Grid_Spatial_Representation"/>
</owl:Class>
```

*In this example we declare that the class Point is disjoint with the class Curve. There is no individual that could be a member of both classes.*

```xml
<owl:Class rdf:about="#Point">
  <owl:disjointWith rdf:resource="#Curve"/>
</owl:Class>
```

*In this example we declare that the class Point is equivalent to the class identified as http:\abc#Puntos.*

```xml
<owl:Class rdf:about="#Point">
  <owl:equivalentClass rdf:resource="http:\abc#Puntos"/>
</owl:Class>
```

**Equivalences**

**Equivalent Class** Two classes $C$ and $D$ are equivalent, if all the statements that define $C$ also define $D$. Then all the members of the class $C$ are also members of the class $D$. These two equivalent classes, can exist in the same ontology, or in two different ontologies. In the last case, it can become a useful tool to link two different vocabularies.

```xml
<owl:DatatypeProperty rdf:ID="#C">
  <owl:equivalentProperty rdf:resource="http://www.ABC.org#D"/>
</owl:DatatypeProperty>
```

**Equivalent Property** Two properties $P$ and $R$, are equivalent if they have the same range and the same domain.

```xml
<owl:DatatypeProperty rdf:ID="#P">
  <owl:equivalentProperty rdf:resource="http://www.ABC.org#R"/>
</owl:DatatypeProperty>
```
3.4. Description Languages based on XML

Complex Classes

It is possible to create complex classes using the primitives supported by OWL, these primitives are: *intersection*, *union* and *complement*, equivalent to AND, OR and NOT operators.

**Intersection** We can define a class as the set of individuals that fulfill all the specifications declared as components of the intersection.

Using Description logics:

\[ C \equiv A_1 \cap \ldots \cap A_m \cap \forall R_1.C_1 \cap \ldots \forall R_n.C.n \]

**Union** Using this primitive we can define a set of individuals that fulfill any of the specifications declared as components of the union.

Using Description logics:

\[ C \equiv A_1 \cup \ldots \cup A_m \cup \forall R_1.C_1 \cup \ldots \forall R_n.C.n \]

**Complement** It includes all the individuals that are Not members of the class.

The complement of \( C \) is \( \neg C \).

**Example 15**

*In the following example, we define the set of individuals that are members of the class MD_GridSpatialRepresentation, OR members of the class Point, OR members of the class Curve (See Figure 3.6).*

```xml
<owl:Class rdf:ID="My_Selected_Datasets1">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Grid_Spatial_Representation">
    </owl:Class>
    <owl:Class rdf:about="#Point">
    </owl:Class>
    <owl:Class rdf:about="#Curve">
    </owl:Class>
  </owl:union>
</owl:Class>
```

![Figure 3.6: OWL Union](image)

*In the following example, we define the set of individuals, that are members of the class MD_GridSpatialRepresentation, and have a property called row_count, for which they have a value equal to 50. (See Figure 3.7).*

40
In the following example we define the set of individuals that are members of the class MD_VectorSpatialRepresentation and are NOT members of the class Curve (See Figure 3.8).

3.5 How to use the ontology information

Using the vocabulary stored in our ontology, we can make descriptions of web services and data sets. In the previous section we have seen that it is possible
3.6 Concluding Remarks

to make these descriptions using a XML based language, like RDF and its extensions like OWL. Using a language based on XML, allow us to publish this information in the web allowing others to use it. The fact that the information is available on the web, allow users that have information that use our vocabulary, to check the terminology and infer not explicitly declared facts on their information. This is because we are using a language with formal semantics.

In this research the main focus is the description of Data sets and Web Services. Specifically the web services that we are focusing on, have properties that associates them to data sets. In the form of some allowed inputs and expected outputs of a given process. This allowed input and expected output are classes that have as their members, all the data sets that fulfill certain conditions.

In section 3.3.2, we discuss the characteristics of a Processing Service Description, the kind of service that operates on Data sets.

To declare the allowed inputs of a Service, we are going to work with OWL and RDF. This languages have almost all the constructs that can be found in Description Logics.

We suggest that the input of a service could be defined by declaring it as the intersection of all the necessary requirements for that service. The necessary requirements could be declared using the has value restriction of OWL like in Figure 3.7. It is possible to add more operands to this intersection making it more specific.

To state whether a data set is an allowed value for a given web service. We have to read the description document of both elements, then read the ontology they refer to (that is the common vocabulary, available on the Web) and infer if the data set could be a member of the class defined as the allowed input of the service.

In their description all data sets require a statement that indicates that they belong to some class. In that description will also appear information about characteristics of the data set. A software application has to based on these assertions, indicate if the data set is an instance of the class declared as input of the service.

This kind of restrictions should be created on the fly, and thus the user will not be restricted to predefined classes when he/she creates service descriptions. The definition of the new class, created by the user can be stored in the document that describes the service, then there is no need to modify the existing public terminology. Only when the user needs to process the description of the service he/she will access the ontology through the web, and get the complete vocabulary (Figure 3.9).

3.6 Concluding Remarks

Description languages based on XML, like RDF, and the vocabularies developed to work with it, provided us with powerful tools to make descriptions, it is possible to associate services with some data sets as inputs, without being restricted only to de description of some predefined classes of data sets.
Figure 3.9: An Ontology with Service and Data set descriptions
3.6. Concluding Remarks
Chapter 4

Implementation

The purpose of this thesis work is to develop guidelines to create descriptions of services and data sets with the use of terms of web-based ontologies. To reach our goal, first we have to select an ontology that contains the vocabulary we require, referring to services and data sets. In this research we decided to create a small ontology, based on the standards ISO 19119 (services) and ISO 19115 (general metadata).

The properties included in the ontology come from the core metadata elements [Con01a]. We developed a prototype, which is able to create descriptions using the terms of this vocabulary. This prototype implements the ideas described in the previous chapter.

4.1 The Tools

This is still a new technology, there are many tools, unfortunately most of them are in experimental stage, and there are new versions almost every day. In this research, we selected two of the leading tools to work with:

**Protege-2000 Beta 2.0** This is an ontology editor. It is an open source Java tool, that could be customizable and extended using Java. As we can see in Figure 4.1, it has a graphical user interface. *Protege* works as a platform on top of which we can deploy other Java applications, plug-ins, to meet some specific purpose [SMI].

Using *Protege*, with a plug-in that allows us to export OWL RDF/XML format, we can create our ontology, describe our classes, attributes, ranges, and relations between our classes.

*Protege* was developed by the Medical Informatics department of Stanford University, (US). The current version of *Protege* is 2.0.1. It was released on February 12, 2004.

The Plug-in, that allows the user to work with OWL, was also developed at Stanford University. It was developed by Holger Knublauch, who is part of the staff of the Stanford Medical Informatics. The latest version of the plug-in dates from February 13, 2004 (build 72). This plug-in is developed in Java, using Jena to load and save OWL models.
4.1. The Tools

**Jena 2.0** This is an Application Program Interface (API) that provides a base level interface to RDF, RDFS and OWL files. Jena is an open source API developed by HP Labs (Hewlett Packard Research Laboratories). The history of Jena is almost as old as the one of RDF, in fact, the co-chair of the RDF Working Group is Brian McBride, one of the Jena developers [Pow03].

Jena does not provide any graphical user interface, but instead offers a set of parsers and query engines, in the form of Java classes. It was designed to be a framework for the development of Semantic Web applications. Jena is well documented and has an active discussion group at

http://groups.yahoo.com/group/jena-dev/

The web page of Jena in Source Forge is:

http://jena.sourceforge.net/

With Jena, we can read an XML file containing information in several formats, like RDF, DAML or OWL.

When we read an ontology, we store it in an instance of a special Java class called *OntModel*. This special class has several methods:

- **listClasses** The result of this method is the complete list of all the classes defined in the model.
- **addSubModel** Using this method we add a given model to the main one.
- **createClass** The user assigns a URI as the identifier of the new class,
Chapter 4. Implementation

this new class is contained in a new Jena Java class known as OntClass.

cREATEINTERSECTIONCLASS To use this method, an RDFList has to be created. This RDFList that contains an enumeration of classes. These are the operands of the intersection. Any instance of this class, is also a member of all the classes that form the intersection.

cREATELIST This method creates a new empty list.

cREATEINDIVIDUAL This method creates an instance of a given class.

cREATEDATATYPEPROPERTY Using this method we create a OWL data type property.

cREATEOBJECTPROPERTY Creates a property that associates instances of two classes.

cREATERESTRICTION With this method, a user can create a restriction on a given property declared in the model, later the user can specify the class of restriction he/she refers to.

cREATEHASVALUERESTRICTION This method creates a new restriction of type hasValue, on the model.

The class OntClass contains a given class of the model.

LISTSUBCLASSES The answer to this command is the list of all the classes with a subclass relation with this class. It is possible to specify if we want all the subclasses or only the immediate ones.

LISTSUPERCLASSES Similar to listSubClasses.

LISTDECLAREDPROPERTIES The answer of this method is a list of properties of this class.

LISTINSTANCES This method creates a list containing the individuals that have this class as its type.

This is not an exhaustive list of the classes and methods contained in Jena. The complete list of methods is included in the JavaDoc files of Jena. This documentation can be downloaded from the web.

In the development of the prototype we use several of these classes. They provide an excellent environment to develop an application that handles ontologies contained in an OWL document.

4.2 The Ontology

The ontology that we use for testing purposes was created by the author, based on ISO19115 and ISO19119. To create this ontology we used Protege. The format of our ontology is OWL DL.

To create more complex scenarios for testing purposes we add new subclasses of data sets to the classification suggested in the standard ISO19115. These new sub classes form a taxonomy of remote sensing. This classification is based on their type of sensor that produced them [BJV+01].
4.2. The Ontology

The basic structure of the ontology was described in Section 3.3. There are three basic classes:

- List
- DataSet
- Service

The rest of the classes are subclasses of these main ones.

4.2.1 List

The class List contains information that is needed to describe either a data set or a service. It is the superclass of:

**CI_ResponsibleParty** This class contains provider information, for both the data sets and the services. It is based on a UML package described on ISO19115 under the same name. In our ontology, we declare that it has three properties with the respective data type ranges:

- address: String
- phone: String
- organization_name: String

**MD_ReferenceSystem** This class contains information about the geographic reference system that the data set uses. It has been modelled as the class of same name in the ISO19115. It has three properties:

- projection: String
- datum: String
- ellipsoid: String

**MD_TopologyLevelCode** This is the list of the topology levels. It is a restricted vocabulary. In our research, we model this list as a class, and the elements of the list are instances of this class.

**MD_TopicCategoryCode** This class represents a list of restricted vocabulary. This class is modelled after the MD_TopicCategoryCode class defined in the ISO 19115. Its instances are the name of the topics.

4.2.2 DataSet

This class represents all the objects that can be considered as data sets. By data set we understand a representation of the real world for use in computer analysis and graphical display of information [Con01a]. We associate with this class the following properties:

- title: String
- provided_by: CI_ResponsibleParty
- topic_category: MD_TopicCategoryCode
- has_reference_system: MD_ReferenceSystem
We also create a hierarchy of geographic data sets based on its spatial representation. However, we add some extra classes in order to have better testing scenarios for the algorithms described in Section 3.2.4. The class `DataSet` has two subclasses:

**MD_GridSpatialRepresentation** This class contains all the data sets commonly known as raster. We associate with this class the following properties:

- `column_count`: Integer
- `row_count`: Integer
- `number_of_dimensions`: Integer

We define the class `Remote_Sensing_Image` as subclass of `MD_GridSpatialRepresentation`. We associate the following properties with this class:

- `number_of_bands`: Integer
- `quantization`: Integer
- `ground_pixel_size`: Integer

`Remote_Sensing_Image` has itself two subclasses:

- **Passive_Sensor_Image** this class has the following subclasses:
  - `Thermal_Scanner_Image`
  - `Aerial_Photo`
- **Active_Sensor_Image** the subclasses of this class are:
  - `Laser_Scanner_Image`
  - `Radar_Image`

**MD_VectorSpatialRepresentation** This class contains data sets in a vector format. We define the following properties for this class:

- `has_topology_level`: `MD_TopologyLevelCode`
- `has_geometry_object_count`: Integer

We define six subclasses for this class:

- Complexes
- Composites
- Curves
- Solid
- Surface
- Point

We did however not declare any new property for these subclasses, we only state that they are subclasses of `MD_VectorSpatialRepresentation`. 

---

49
4.2.3 Service

This class contains the Geographic Web Services, (we defined a service in Section 2.2). We declare as common characteristics for all services:

- service.name: String
- service.provided_by: CI_ResponsibleParty

We organize the different subclasses of Service using as a model, the taxonomy suggested in ISO19119:

- Geographic human interaction services
- Geographic model/information management services
- Geographic workflow/task management services
- Geographic processing services
  - Geographic processing services-spatial
  - Geographic processing services-thematic
  - Geographic processing services-temporal
  - Geographic processing services-metadata
- Geographic communication services
- Geographic system management services

The Geographic processing services, operate on geographic data sets, performing operations. This close relation with the geographic data sets make them of much interest in this research. We define one property for this class of services:

- has.operation: String

With this property we link the service with a specific functionality. The string value is the URI of a property that has as domain the allowed inputs of the service, and as range the expected outputs. To allow the prototype to distinguish this property from the rest, we added a label that indicates the prototype the special nature of this property. The string value of this property is not used to make any inference. It is only used to indicate the existence of a property that represent a functionality. To make any inference, a special software will be needed. This software will have to read the string value, transform it into a URI, and get a property that corresponds to it. This is not supported by OWL, or Description Logics, it is a mechanism develop by the author to associate a functionality to a service.

Example 16

In the next example we describe a service that has as functionality the process of a data set of type Point into a data set of type MD_GridSpatialRepresentation:

Point → MD_GridSpatialRepresentation
From line 1 until line 4, we represent the functionality of this service as a property identified as Service_Example_Functionality. With the following characteristics:

- **domain**: Point
- **range**: MD_GridSpatialRepresentation

From line 5 until line 14 we assign values for the service characteristics. From line 11 until line 13, we associate the functionality previously declared, with this service. We transform the URI of the property that represents the functionality into a string, line 12, and assign this value to the property has_operation.

In the Example 16, we declare the domain and the range of the property that represents the functionality as simple classes. However, in our application we construct complex classes to represent both input and output. We think that a complex class represents better these concepts.

### 4.3 Prototype Design

#### 4.3.1 Users

In the process of the prototype design, we identify three types of user which are illustrated in Figure 4.2:

- The Data set Provider: This user makes descriptions of Data sets.
- The Service Provider: This user creates descriptions of Geographic Web Services.
- Ordinary Service/Data set consumer: Any user that would like to infer if a certain data set is an appropriate input for a given service.

#### 4.3.2 Sequence Diagrams

In each scenario involving any of the users, certain operations are performed in a given sequence. We can represent this chain of operations in a diagram, in the Unified Modeling Language (UML) this is known as a sequence diagram [Qua01]. We elaborate such diagrams for each of the user types.
Data set provider

In the case of the data set provider, the operations performed are represented in Figure 4.3. These operations are:

1. The user selects the ontology that he will use to describe the data set.
2. The user selects the class that corresponds to this data set.
3. The prototype searches in the ontology for the properties that have as domain, this class or any superclass of it, and shows them to the user.
4. The user assigns values to the properties associated with the selected class, and submits the new description.
5. The prototype checks if the assigned values have the right data type for the corresponding properties, if they are valid members of the range of the
properties. If the type of the values is correct, it creates a new description, if not it informs the user of the mistake.

6. The prototype creates a new description and adds it to the knowledge base. Only values of the correct type are included in this description. In this way we ensure the consistency of our knowledge base.

**Service provider**

![Sequence diagram for a Service Provider](image)

In Figure 4.4 we represent the operations performed in the scenario involving a service provider. In this case the first four operations are similar to the data set provider scenario, however this user performs other operations:

1. The user selects the ontology that he will use to describe the service.
2. The user selects the class of service that better describe his own service.
3. The prototype selects the properties associated with that class or to its superclasses.
4. The user assigns values to these properties.
5. The prototype checks that the values assigned to this the service have the correct type.
6. The user selects one subclass of data set to further specify. Then the user creates restrictions on the properties of this class.
7. The prototype checks if the constraints are valid.
8. The prototype creates a new description and adds this to the knowledge base. The prototype also creates one property that represents the functionality, assigning the input as the domain and the output as the range of this property.
4.3. Prototype Design

Ordinary Service/Data set consumer

In Figure 4.5, is provided the sequence diagram for the third type user. In this case the following operations are performed:

1. The user selects the ontology to use as common vocabulary.
2. The prototype reads the ontology, and creates a model, representing it in memory.
3. The user selects the service description document.
4. The statements declared in the service description are added to the model.
5. The user selects the data set description document.
6. The statements declared in the data set description are added to the model.
7. The prototype compares the statements of the data set with the requirements to be an allowed input. The allowed inputs of the service are represented as a class. We try to infer whether this data set is an instance of this class. (We use proposition 2, further explanation in Section 3.2.4). The class that represents the allowed inputs
8. The prototype creates a report indicating the matching result.

As we can see the first five operations are similar in the case of the data set provider and the service provider scenarios. Because of this, we decided to create a single prototype application for both them. We call this application prototype “Descriptor” This application is accessed by these types of users to create a new description.
Chapter 4. Implementation

We assume that the data set or service provider will access the ontology through the web. This type of user will not be allowed to modify the ontology, then his/her descriptions should be stored in separately independent documents. Each one of these documents will use the concepts defined in the ontology and indicate where in the web this ontology is located. The format of these new documents will be also XML/RDF. Using this common language we can transfer our descriptions through the web.

The vocabulary used in these documents is declared in the ontology, if any user wants to infer something about a description, he would have to read the ontology in the Web, and based on the relations declared there, infer knowledge not explicitly declared in the description document. That is the goal of the third type of user.

Because of its different operations we decided to create another application, with close links to the first one. We called this application prototype “Match Tester”.

Both prototypes were developed using the Java language, and Jena API.

4.3.3 Descriptor

The users of this prototype are data set providers and service providers. This application allows these users to access a web based ontology with a graphical interface. Later the users can create instances of the classes declared in the ontologies, making statements on the properties associated with instances of these classes.

In Figure 4.6, we can see a UML class diagram, representing the main classes used in this application.

Description of the used Java classes

Estela09 This Java class is the starting point of the application. It creates one instance of the Java class Config which reads local variables to configure the application. This class provides the user with a list of URL's of the recently used ontologies. From these URI's the user selects the ontology that will be used to create new descriptions. Then it creates a new instance of OntViewer02 providing the URL of the ontology as a parameter. The Figure 4.7 shows the graphical interface created by this class.

Config This Java class reads one text file from the local directory of the application. The name of this file is config.ini. Here is stored the location where the application will store the descriptions. Here is also indicated the label used by the application to distinguish the property has_operation from other properties.

OntViewer02 It requires the use of a parameter, indicating the selected URI (as string) of the ontology to be used.

This Java class accesses the URI, and store the ontology located there into an instance of OntModel class. Once the ontology is stored in the model, the prototype requires the model to list all the classes, and its
4.3. Prototype Design

This is stored as an array with the form \textit{Parent - Child}. This array is required for the class \texttt{TreeMaker} to construct a graphical interface. The graphical interface created by this class is shown in Figure 4.8.

From the graphical interface the application allows the user to select one class. The entity the user wants to describe is an instance of this class. To make a description the user selects the button \textit{Add Instance}. This
actions creates a new instance of the Java class Description. This class requires as parameter the ontology class selected by the user in the graphical interface.

The text fields and combo boxes created by Description allow the user to enter values for the properties of the instance (Figure 4.8).

![Graphical Interface provided by the class OntViewer02](image)

Figure 4.8: Graphical Interface provided by the class OntViewer02

In the case the selected ontology class requires the description of inputs and outputs, the user selects one ontology class, (any subclass of DataSet) and provides it as a parameter to the Java class Class2Restrict. This is the starting point to create the restrictions. In this point the prototype creates a new instance of the Java class Class2Restrict.

When the user selects the button **Write Instance** the prototype evaluates if the values have the correct data type. All the statements with the correct data type are added to the model $m$ and to the model $m_1$. Then a new XML/RDF file is created with the information stored in $m_1$.

The user can write the content of the model $m$ in a XML file, in a local folder by selecting the button **Write Model**.

Another capability of this class is to allow the user to view the instances declared for a selected class. The button **View Instances** creates a new instance of the Java class MyInstances.

Because of the structure of the application is possible to create several instances of OntViewer02, and read multiple OWL documents simultaneously.

**TreeMaker** Requires an array of the pairs with the form Parent - Child. This class uses this array to construct a tree, that is used in OntViewer02 to visualize the concepts and the relations between them. Using this class is possible to visualize classes that have multiple immediate super classes.
4.3. Prototype Design

**OntModel** This is a Jena Java class that serves as temporal repository of an ontology.

- m This model stores the main ontology used by the application.
- m1 This model is a temporal repository of the description of any instance.

**Description** This Java class receives as parameter one ontology class. It queries the model for all properties associated to that ontology class. These properties are both, explicitly declared and inferred (by inheritance from the ontology superclasses). For each property depending if it is a data type or an object property, it creates one text field, or one combo box respectively. This is the place where the users enters the values for the properties. In the case of the object property, the combo box, contains the identifiers of the instance of the range of this property.

**Class2Restrict** This Java class, creates text fields and combo boxes in a similar way to Description. This class creates a new instance of the Java class RestWindow.

**RestWindow** This class provides a graphical interface that allows the user to create has value constraints on the properties of a given class. The consistency of these constraints is tested by this Java class, comparing the range of the properties with the assigned values. The graphical interface created by this class is shown in Figure 4.10.

**MyInstances** This class is only for visualization purposes. It creates a window (Figure 4.9) where it displays the instances of a selected class.

![Figure 4.9: Graphical Interface provided by the class MyInstances](image)

**Output of Descriptor Prototype**

Once the descriptor creator reads the ontology described in Section 3.3, it is able to make the two kinds of descriptions:

**Data set description** The description of a data set makes use of the following kinds of assertions:
Chapter 4. Implementation

Figure 4.10: Graphical Interface provided by the class RestWindow

\[ C(a) \]
We declare this data set \( a \) is member of the class \( C \) (\( C \) could be any subclass of \( \text{DataSet} \)).

\[ P_1(a, b_1) \ldots P_n(a, b_n) \]
\( P_1 \ldots P_n \) are the properties that have the class \( C \) (or any of its superclasses) as domain. These properties have values \( b_1 \ldots b_n \) associated with them.

**Service description** In the case of a Service, the description has two parts:

- In the first part, the description declares fixed properties and their values, for example, the name of the service or its provider. The assertions used in this point are similar to the ones used for the description of a data set.

\[ D(a) \]
We declare the service \( a \) as member of the class \( D \) (a given subclass of \( \text{Service} \)).

\[ P_1(a, b) \ldots P_n(a, b) \]
\( P_1 \ldots P_n \) are the properties that have the class \( D \) (or any of its superclasses) as domain. These properties have values \( b_1 \ldots b_n \) associated with them.

- In the second part, the description refers to the functionality of the service, in the form of a relation between two classes, the domain of this relation is the input of a service, and the range is the output. The functionality of the service is described with the following kinds of assertions:
4.3. Prototype Design

The declaration of a new class Input, which is the result of an intersection between a selected class by the user \( C \) and has value constraints created on the fly \( (\equiv x \ R_1)\ldots(\equiv x \ R_n) \). These constraints affect the properties that have class \( C \) as their domain.

The declaration of a new class Output which is the result of the intersection of a user selected class \( C' \) and has value constraints created on the fly \( (\equiv y \ Q_1)\ldots(\equiv y \ Q_m) \). These constraints affect properties that have \( C' \) as their domain.

\[ \forall Functionality(a, b) \]
\[ \text{Input}(a) \quad \text{and} \quad \text{Output}(b) \]
\[ \text{has\_operation}(a, \text{“Functionality”}) \]

4.3.4 Match Tester

The user of this prototype is an ordinary service/data set consumer. This user wants to test the suitability of a certain data set for a service. In Figure 4.11 is represented the UML class diagram of this prototype.

For this application we developed one single Java class. We call it IO.Checker. The operations performed in this class are:

- The prototype, creates an instance of Config, with this operation, the prototype obtains the location of the catalogs with the list of descriptions created by Descriptor.

- The user selects one ontology that is going to be the common vocabulary for the data set and for the service descriptions. This ontology is stored into an instance of OntModel in the application we identified it as m.

- The user selects a document that describes a data set, stores it into a temporal OntModel instance, and later adds it to the to m.

- The user perform the an operation similar to the previous one, but this time he/she selects a document that describes a service.
• The prototype compares the statements that describe the data set and the input of the service, using the algorithm described in Section 3.2.4 in Proposition 2.

• The prototype makes a report of the possible match. The report has three possible results:
  
  – Positive: The data set is an allowed input of the service.
  – Negative: In the description of the data set, there are statements that make it violate the constraints declared for the input of the service.
  – Not enough information: In this case, there are not statements that violate the constraints, but the available statements does not fulfill all the constraints declared for the service input.

The graphical interface provided by this prototype is shown in Figure 4.12. The prototypes previously described are used by the users to create descriptions and to infer possible matches between data sets and service. Figure 4.13 shows the way the three users interact with the prototypes.
4.3. Prototype Design

4.3.5 Example Case Scenario

This example case scenario involves three data set provider:

Data set Provider  Suppose a data set provider identified as IGM offers three
data sets for a certain area. The data sets offered by this data set provider are: one aerial photo, one laser scanner image and one scanned image of a hard copy map. The information the data set provider offers about the products is presented in Table 4.1.

<table>
<thead>
<tr>
<th>Title</th>
<th>Aerial Photo</th>
<th>Laser Scanner Image</th>
<th>Scanned map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference system</td>
<td>Aerial Photo Example</td>
<td>Laser Scanner Image Example</td>
<td>Scanned Map ITRF94</td>
</tr>
<tr>
<td>Row count</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Col count</td>
<td>2</td>
<td>2</td>
<td>1500</td>
</tr>
<tr>
<td>Number of dimensions</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of bands</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Quantization</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Ground pixel size</td>
<td>5 meters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using the “Descriptor” prototype (Figure 4.14), the data set provider selects the ontology he/she will use to describe the data sets. In this example the ontology selected is located in:

file:/D:/Prototype/gfm2.owl

Then data set provider selects one class for the data set, and creates one instance of this class. To identify the instance the user provides a name, that is later transformed into a URI by the application. In the case of the scanned map the URI is:

file:/D:/Prototype/gfm2.owl#Scanned_Map_Example"

Later the data set provider declares values for the properties associated to this class. The document produced in this way is represented in Example 17.

Example 17

```xml
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="file:/D:/Prototype/gfm2.owl#" >
    <rdf:Description rdf:about="file:/D:/Prototype/gfm2.owl#Scanned_Map_Example">
      <j.0:row_count rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1000</j.0:row_count>
      <j.0:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Scanned Map Example</j.0:title>
    </rdf:Description>
</rdf:RDF>
```
4.3. Prototype Design

In Example 17, from line 1 to line 4 is declared the name space of the vocabulary used in this document. The description of the data set starts in line 5 until line 22. In line 6 is indicated the class of this data set. From line 7 until line 21 are indicated the properties and their respective values for this data set.

The RDF/XML description documents produced by the prototype are located in Appendix B (Examples 18-20).

Service Provider Suppose there is a service provider identified as IGN who offers an object detection service. This service is used to detect real world objects on images. Table 4.2 contains the characteristics of this service.

<table>
<thead>
<tr>
<th>Title</th>
<th>Service Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>100 US Dollars</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Functionality:</td>
<td>be a remote sensing image</td>
</tr>
<tr>
<td>Input must:</td>
<td>have a ground resolution of 5 meters</td>
</tr>
<tr>
<td></td>
<td>have 1000 columns</td>
</tr>
<tr>
<td></td>
<td>have 1000 rows</td>
</tr>
<tr>
<td>Output is:</td>
<td>a vector</td>
</tr>
</tbody>
</table>

Using the prototype Descriptor, the service provider can select the ontology to use to describe the service, then indicate that this service belongs to a certain class, and assign values to its properties.
The service provider also defines a class *Input*, that represents the allowed inputs for this service. This class is defined as the intersection of a existing ontology class, and classes defined by constraints created by the user on the fly (owl:hasValue restrictions). In the case of this service this class is defined as:

\[
Input \equiv \ Remote\_Sensing\_Image \cap ( = 1000 \ column\_count ) \cap ( = 1000 \ row\_count ) \cap ( = 5 \ meters \ ground\_pixel\_resolution )
\]

*Input* is composed by the set of individuals that are members of the following classes at the same time:

The class *Remote\_Sensing\_Image*

The set of elements that have the value 1000 in the property *column\_count*

The set of elements that have the value 1000 in the property *row\_count*

The set of elements that have the value “5 meters” in the property *ground\_pixel\_resolution*

Figure 4.15 shows the prototype with the information of the service.

With the information provided by the user, the prototype creates an OWL document which describes the service using elements from the selected ontology. The document that describes the service is located in Appendix B Example 21.

**Ordinary data set/service consumer** This user wishes to identify from the data sets offered by the data set provider which one to use with the service previously described. To use the Match Tester prototype, the user selects the ontology to use and creates a temporal copy of it in memory. Then
he/she reads the descriptions of both the data set and the service and adds the statements of those descriptions to the temporal copy of the ontology.

From the description of the service, the application obtains the string value of the property has_operation. Then transforms this string into a URI and gets the property identified by this URI. This property describes the functionality of the service. Then the application gets the domain of this property, which is a class that represents the allowed inputs for this service. We call this class Input.

Each of declare requirements of the input of the service (Table 4.2) is consider a class. The Input is the intersection of all these classes.

From the description of the data set the application obtains a set of statements that are going to be used to infer if this data set is a member of the class that represents the allowed inputs of the service. To perform this operation the application implements Proposition 2, described previously in Section 3.2.4.

Then the prototype evaluates if in the copy of the ontology stored in memory there are statements that indicate whether a certain data set is member of the class Input.

<table>
<thead>
<tr>
<th>Service Input Conditions</th>
<th>Aerial Photo</th>
<th>Laser Scanner Image</th>
<th>Scanned Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote_Sensing Image</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>(=1000 column_count)</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>(=1000 row_count)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(=5meters ground_pixel_resolution)</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

The inference is made in the this way:

Aerial_Photo(Aerial_Photo, Example)  The individual Aerial_Photo, Example is of class Aerial_Photo.
Aerial_Photo ⊑ Passive_Sensor_Image Aerial_Photo is subclass of Passive_Sensor_Image.

then:
Aerial_Photo ⊑ Remote_Sensing_Image Aerial_Photo is subclass of Remote_Sensing_Image.
and
Remote_Sensing_Image(Aerial_Photo, Example) Aerial_Photo, Example is of class Remote_Sensing_Image.

In the case of the has value restrictions:

column_count(Aerial_Photo, Example, 1000) There is a statement that indicates that the individual Aerial_Photo, Example has a property column_count with a value 1000.

(=1000 column_count)(Aerial_Photo, Example) Then Aerial_Photo, Example is member of the set of elements that have the value 1000 in the property column_count.

In the case of the scanned map:
Chapter 4. Implementation

The individual \textit{Scanned\_Map\_Example} is of class \textit{MD\_GridSpatialRepresentation}. Based on its type we can not say that this individual is of type \textit{Remote\_Sensing\_Image}. Column count of \textit{Scanned\_Map\_Example} is 1500. \textit{Scanned\_Map\_Example} is NOT member of the set of elements that have the value 1000 in the property column count. Then this data set is NOT an allowed input for this service.

In the case of the \textit{Laser\_Scanner\_Image} we have not statements about its value on the property ground pixel resolution then we can not declare that this is an allowed input, however we can not say it is not, because we do not have enough information.

4.4 Concluding Remarks

The result of the descriptor prototype is a new XML document that contains all the statements referring to the subject of description. Because these descriptions use a format with mathematical semantics, it is possible to infer new information from the one the documents explicitly contain.

It is necessary to explore methods to store all the assertions in a database. This would allow faster operations, and easier maintenance. In a real case scenario, this catalogue should be available on the web.
4.4. Concluding Remarks
Chapter 5

Conclusions

OWL DL is a very expressive language that allow users to create semantically rich descriptions. OWL DL because is based on Description Logics, allows the use of sound algorithms. These algorithms allow a user that works with OWL to perform inferences based on the descriptions.

In our research we develop mechanisms to create descriptions that use terms from web based ontologies. Using an inference prototype we proved that these descriptions could be used to determine whether a given data set could be used for a certain service.

In our research we were able to create new concepts from the ones defined in our ontology. New complex terms could be created based on more simple ones. By identifying the origin of the basic terms that define complex ones, we can infer not explicit knowledge.

OWL DL provides the user with great expressive capabilities. However there are factors that constraint the use of services that are hard to describe using OWL, constructions like concrete domain min restriction or concrete domain max restriction are not available in OWL. These kind of restrictions would be very useful, for example to compare the bounding box of a data set with the bounding box where a given service operates. For a proper match the bounding box of the data set must be within the bounding box of the web service. It is important to find a way to express such constraints, in order to create richer descriptions of both data sets and web services.

In a real scenario, with multiple services, the matching problem between a data set and a service is not the only problem. One important issue is the ability to determine if two services can be chained. Based on their descriptions, we can infer the relations between them. If the output of the first one is a superclass of the input of the second, it should be possible to chain them, but besides that, there are also other considerations to study that have not been treated in this research.

We use a small ontology constructed by us to show the mechanisms to describe data sets and services. Based on these descriptions we can infer matches between a data set and the input of a service. This ontology is small, and incomplete. We think that the methods used to create the descriptions and evaluate the matching possibility can be used with larger vocabularies. Further work should be carried on larger ontologies in real case scenarios.
OWL provides richer expressions than WSDL to describe services and data sets. However, WSDL is an industrial standard and there is an already developed infrastructure around it. It would be interesting to explore the procedures to translate the descriptions from RDF/XML format into industrial standard formats, like WSDL. This will allow us to work with the infrastructure already built with WSDL.

In this research we only used models stored in memory and descriptions created in XML format files. We think that the possibility to store knowledge in the form of OWL files in a XML data base is interesting. This would allow faster operations, and easier maintenance.
Bibliography

[AL03] Trias Aditya and Rob Lemmens. Chaining distributed GIS services, 2003. vii, 10


[Con02] Open Gis Consortium. Appendix D: Interoperability program service model (ISPM), April 2002. 12, 15


[GCDT01] Javier Gonzalez-Castillo and Claudio Bartolini David Trastour. Description logics for matchmaking of services. page 12, 2001. 32


[LH03] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. page 9, May 2003. 10, 15, 27

Bibliography


[SMI] Stanford University School of Medicine Stanford Medical Informatics. Protege user’s guide. 45

[Sys03] Systinet. Introduction to web services. 2003. ix, 7, 9

[W3C02] W3C. RDF Vocabulary Description Language 1.0: RDF Schema, November 2002. 34


Appendix A

Glossary

**ABox** It contains assertions about named individuals in terms of this vocabulary [BMNPS02].

**BNF** "Backus Naur Form", John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language [oG].

**DARPA** Defense Advanced Research Projects Agency, is the central research and development organization for the Department of Defense (DoD). It manages and directs selected basic and applied research and development projects for DoD, and pursues research and technology where risk and payoff are both very high and where success may provide dramatic advances for traditional military roles and missions.

**DAML** DARPA Agent Markup Language

**DL** Description Logics

**HTTP** Hypertext Transfer Protocol

**Metadata** Is defined as background information that describes the content, quality, condition and other appropriate characteristics of the data. Metadata is a simple mechanism to inform others of the existence of data sets, their purpose and scope. In essence, metadata answer *who, what, when, where, why, and how*, questions about all facets of the data made available.

Metadata can be used internally by the data provider to monitor the status of the data sets and externally to advertise to potential users through a national clearinghouse. Metadata are important in the production of a digital spatial data clearing house, where potential users can search for the data they need.

Metadata play a variety of informative roles:

**availability** information needed to determine the data sets that exist for a geographic location.
fitness for use information needed to determine whether a data set meets a specific need.

access information needed to acquire an identified data set.

transfer information needed to process an use a data set.

administration information needed to document the status of existing data (data model, quality, completeness, temporal validity, etcetera) to define internal policy for update operations from different data sources.

The metadata should be flexible enough to describe a wide range of data types. Details of the metadata vary with the purpose of their use, so certain levels of abstraction are required.

OIL Ontology Inference Layer

Ontology Is a definition of the words and concepts (the meaning) used to describe and represent an area of knowledge. Is a specific vocabulary used to describe a part of the reality, plus a set of explicit assumptions regarding the intended meaning of that vocabulary. These concepts and the relationships between them are usually implemented as classes, relations, properties, attributes, and values (of the properties/attributes). The properties or attributes have either explicit values or more often, have value ranges. By range we mean that the only possible values for any instance of a property must come from one class. An ontology can include:

- Classes (general things) in the many domains of interest
- Instances (particular things)
- The relationships among those things
- the functions of and processes involving those things
- constraints on and rules involving those things

An ontology models the vocabulary and meaning of domains of interest: the objects (things) in domains; the relationships among those things; the properties, functions and processes involving those things; and constraints on and rules about those things.

OWL Web Ontology Language (sometimes called Ontology Web Language) a language developed by W3C's Web Ontology Working Group and intended to be the successor of DAML+OIL. OWL is the most expressive knowledge representation for the Semantic Web so far.


Protege Is an ontology management tool developed and maintained by the Medical Informatics Laboratory at Stanford University based on the OKBC
knowledge model and is recognized as an exemplary tool for managing ontologies.

**RDF** Resource Description Framework expresses instance-level semantic relations phrased in terms of a triple:

\[ \text{<subject><verb><object>} \]

**RDFS** RDF Schema It expresses class-level relations describing acceptable instance-level relations.

**SOA** Service Oriented Architecture.

**SOAP** Simple Access Protocol, is the XML-based message protocol (or API) for communicating with web services. It is the most accepted protocol.

**Taxonomy** A hierarchic classification (typically in a tree structure) of real world objects. In information technology, a taxonomy is used to classify the information correlates of those objects.

**TBox** Introduces the terminology, i.e., the vocabulary of an application domain. The vocabulary consists of concepts, which denote sets of individuals, and roles, which denote binary relationships between individuals. [BMNPS02].

**UDDI** Universal Description Discovery and Integration.

**UML** Unified Modeling Language.

**URL** Universal Resource Locator.

**URI** Universal Resource Identifier.

**Webservices** software applications that can be discovered, described and accessed based on XML and standard Web protocols over intranets, extranets and the Internet.

Web services are software applications available on the Web that perform specific functions.

Web Services are built on XML, a standard that is supported and accepted by thousands of vendors worldwide, web services first focus on interoperability. XML is the syntax of messages, and HTTP is the underlying protocol, is how applications send XML messages to Web Services in order to communicate.

**W3C** World Wide Web Consortium.

**WSDL** Web Service Description Language.

**XML** Extensible Markup Language.
Appendix B

Examples

Example 18
In this example we show an RDF/XML document produced by the prototype. It describes a data set Aerial Photo of type Aerial_Photo introduced in Section 4.3.5.

```xml
<?xml version="1.0" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="file:/D:/Prototype/gfm2.owl#">
  <rdf:Description rdf:about="file:/D:/Prototype/gfm2.owl#Aerial_Photo_Example">
    <j.0:row_count rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1000</j.0:row_count>
    <j.0:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Aerial Photo Example</j.0:title>
    <j.0:number_of_dimensions rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</j.0:number_of_dimensions>
    <j.0:number_of_bands rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3</j.0:number_of_bands>
    <j.0:column_count rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1000</j.0:column_count>
    <j.0:ground_pixel_size rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5 meters</j.0:ground_pixel_size>
    <j.0:has_reference_system rdf:resource="file:/D:/Prototype/gfm2.owl#ITRF_94"/>
    <j.0:topic_category rdf:resource="file:/D:/Prototype/gfm2.owl#GEOSCIENTIFIC_INFORMATION"/>
    <j.0:provided_by rdf:resource="file:/D:/Prototype/gfm2.owl#IGM"/>
  </rdf:Description>
</rdf:RDF>
```
In this example the type of the data set is indicated in line 6. The values for
the properties are assigned from line 7 until line 27.

Example 19
This example shows an RDF/XML document produced by the the prototype. It
describes a data set of type Laser.Scanner.Image introduced in Section 4.3.5.

```xml
<?xml version="1.0" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="file:/D:/Prototype/gfm2.owl#" >
  <rdf:Description rdf:about="file:/D:/Prototype/gfm2.owl#Laser_Scanner_Image_Example">
    <rdf:type rdf:resource="file:/D:/Prototype/gfm2.owl#Laser_Scanner_Image"/>
    <j.0:row_count rdf:datatype="http://www.w3.org/2001/XMLSchema#int">100</j.0:row_count>
    <j.0:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Laser Image Example</j.0:title>
    <j.0:number_of_dimensions rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</j.0:number_of_dimensions>
    <j.0:number_of_bands rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</j.0:number_of_bands>
    <j.0:column_count rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1000</j.0:column_count>
    <j.0:quantization rdf:datatype="http://www.w3.org/2001/XMLSchema#int">255</j.0:quantization>
    <j.0:has_reference_system rdf:resource="file:/D:/Prototype/gfm2.owl#ITRF_94"/>
    <j.0:topic_category rdf:resource="file:/D:/Prototype/gfm2.owl#GEOSCIENTIFIC_INFORMATION"/>
    <j.0:provided_by rdf:resource="file:/D:/Prototype/gfm2.owl#IGM"/>
  </rdf:Description>
</rdf:RDF>
```

Example 20
This is an RDF/XML document produced by the the prototype. It describes a
data set of type MD_GridSpatialRepresentation introduced in Section 4.3.5.

```xml
<?xml version="1.0" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="file:/D:/Prototype/gfm2.owl#" >
  <rdf:Description rdf:about="file:/D:/Prototype/gfm2.owl#Scanned_Map_Example">
  </rdf:Description>
</rdf:RDF>
```
Appendix B. Examples

In this example the type of the data set is indicated in line 6. The values for the properties are assigned from line 7 until line 21.

Example 21
This is an OWL document produced by the prototype. It describes a service of class Object_Detection_Service introduced in Section 4.3.5.
In this example the description of the characteristics of the service is declared between lines 21 and 28. Between lines 56 to 60 it is described the property that represents the functionality of this service. The prototype creates a restriction class using anonymous nodes identified as:
A1, A2, A3, A4, A5 and A6
Appendix C

Ontology

This is the Ontology that contains our basic vocabulary. It is an OWL file serialized in RDF/XML syntax.

```xml
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdfs="http://purl.org/rdf-schema/
  xmlns:rdf="http://www.w3.org/2000/01/rdf-schema/
  xmlns:owl="http://www.w3.org/2002/07/owl/
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0/
  xmlns:daml="http://www.daml.org/2001/03/daml+owl/
  xmlns:dc="http://purl.org/dc/elements/1.1/
  xmlns:file:/D:/Prototype/gfm2.owl#">
  <owl:Ontology rdf:about="file:/D:/Prototype/gfm2.owl#"/>

  <owl:Class rdf:id="CI_ResponsibleParty"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#List"/>

  <owl:Class rdf:id="MD_TopologyLevelCode"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#List"/>

  <owl:Class rdf:id="MD_GridSpatialRepresentation"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#DataSet"/>

  <owl:Class rdf:id="Communication_Service"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>

  <owl:Class rdf:id="Coordinate_Conversion_Service"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Spatial_Processing_Service"/>

  <owl:Class rdf:id="Model_Information_Management_Service"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>

  <owl:Class rdf:id="Image_Processing_Service"
    rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Thematic_Processing_Service"/>

  <owl:Class rdf:id="Feature_Matching_Service"/>
</rdf:RDF>
```
Appendix C. Ontology

113  </owl:Class>
114  <owl:Class rdf:ID="System_Management_Service">
115   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>
116  </owl:Class>
117  <owl:Class rdf:ID="Point">
118   <rdfs:subClassOf>
119    <owl:Class rdf:about="file:/D:/Prototype/gfm2.owl#MD_VectorSpatialRepresentation"/>
120   </rdfs:subClassOf>
121  </owl:Class>
122  <owl:Class rdf:ID="Metadata_Processing_Service">
123   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Processing_Service"/>
124  </owl:Class>
125  <owl:Class rdf:ID="Geoparsing_Service">
126   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Thematic_Processing_Service"/>
127  </owl:Class>
128  <owl:Class rdf:ID="Complexes">
129   <rdfs:subClassOf rdf:about="file:/D:/Prototype/gfm2.owl#MD_VectorSpatialRepresentation"/>
130  </owl:Class>
131  <owl:Class rdf:ID="Coordinate_Transformation_Service">
132   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Spatial_Processing_Service"/>
133  </owl:Class>
134  <owl:Class rdf:ID="Human_Interaction_Service">
135   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>
136  </owl:Class>
137  <owl:Class rdf:ID="MD_VectorSpatialRepresentation">
138   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#DataSet"/>
139  </owl:Class>
140  <owl:Class rdf:ID="Sampling_Service">
141   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Temporal_Processing_Service"/>
142  </owl:Class>
143  <owl:Class rdf:ID="Aerial_Photo">
144   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Passive_Sensor_Image"/>
145  </owl:Class>
146  <owl:Class rdf:ID="Temporal_Proximity_Service">
147   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Temporal_Processing_Service"/>
148  </owl:Class>
149  <owl:Class rdf:ID="Thermal_Scanner_Image">
150   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Passive_Sensor_Image"/>
151  </owl:Class>
152  <owl:Class rdf:ID="Radar_Image">
153   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Active_Sensor_Image"/>
154  </owl:Class>
155  <owl:Class rdf:ID="Laser_Scanner_Image">
156   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Active_Sensor_Image"/>
157  </owl:Class>
158  <owl:Class rdf:ID="Image_Manipulation_Service">
159   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#Thematic_Processing_Service"/>
160  </owl:Class>
161  <owl:Class rdf:ID="Composites">
162   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#MD_VectorSpatialRepresentation"/>
163  </owl:Class>
164  <owl:Class rdf:ID="Curve">
165   <rdfs:subClassOf rdf:resource="file:/D:/Prototype/gfm2.owl#MD_VectorSpatialRepresentation"/>
166  </owl:Class>
Appendix C. Ontology

<owl:DatatypeProperty rdf:ID="has_geometric_object_count">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#MD_VectorSpatialRepresentation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="has_operation">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#Processing_Service"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:label xml:lang="en">service functionality</rdfs:label>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="organization_name">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#CI_ResponsibleParty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="number_of_dimensions">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#MD_GridSpatialRepresentation"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="has_price">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#Service"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ellipsoid">
  <rdfs:domain rdf:resource="file:/D:/Prototype/gfm2.owl#MD_ReferenceSystem"/>
</owl:DatatypeProperty>

<MD_TopicCategoryCode rdf:ID="ELEVATION"/>
<MD_TopologyLevelCode rdf:ID="GEOMETRY_ONLY"/>
<MD_TopicCategoryCode rdf:ID="BIOTA"/>
<MD_ReferenceSystem rdf:ID="POSTDAM">
  <ellipsoid>BESSEL 1841</ellipsoid>
  <datum>POSTDAM</datum>
</MD_ReferenceSystem>
<CI_ResponsibleParty rdf:ID="IGM">
  <organization_name>Instituto Geografico Militar Chile</organization_name>
  <address>Av. Angamos</address>
</CI_ResponsibleParty>
<MD_ReferenceSystem rdf:ID="GRS80_EUROF">
  <ellipsoid>GRS80</ellipsoid>
  <datum>GRS80 EUROF</datum>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="OCEANS"/>
<MD_ReferenceSystem rdf:ID="ECONOMY"/>
<MD_ReferenceSystem rdf:ID="TRANSPORTATION"/>
<MD_ReferenceSystem rdf:ID="CLIMATOLOGY_Meteorology_AtmOSPHERE"/>
<MD_ReferenceSystem rdf:ID="RT90">
  <ellipsoid>BESSEL RT90</ellipsoid>
  <datum>RT90</datum>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="GDA94">
  <ellipsoid>GRS80</ellipsoid>
</MD_ReferenceSystem>
Appendix C. Ontology

<datum>ANS84</datum>
<ellipsoid>AUSTRALIAN NATIONAL</ellipsoid>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="DHDN_1">
<ellipsoid>BESSEL DHDN</ellipsoid>
<datum>DHDN-1</datum>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="DHDN">
<ellipsoid>BESSEL</ellipsoid>
<datum>DHDN</datum>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="ED87">
<datum>ED87</datum>
<ellipsoid>INTERNATIONAL</ellipsoid>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="LUZON">
<datum>LUZON</datum>
<ellipsoid>CLARKE 1866</ellipsoid>
</MD_ReferenceSystem>
<MD_ReferenceSystem rdf:ID="NAVAL_WEAPONS_LABORATORY">
<ellipsoid>NWL 9D</ellipsoid>
<datum>NAVAL WEAPONS LABORATORY</datum>
</MD_ReferenceSystem>
<MD_TopicCategoryCode rdf:ID="BOUNDARIES"/>
<MD_ReferenceSystem rdf:ID="CH_1903">
<ellipsoid>BESSEL 1841</ellipsoid>
<datum>CH_1903</datum>
</MD_ReferenceSystem>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.0, Build 72)
http://protege.stanford.edu -->