FACETED LIGHTWEIGHT ONTOLOGIES: A FORMALIZATION AND SOME EXPERIMENTS

Mohammad Shahjahan Feroz Farazi

April 2010

Technical Report # DISI-10-050

PhD Dissertation

International Doctorate School in Information and Communication Technology

DIT - University of Trento

FACETED LIGHTWEIGHT ONTOLOGIES: A FORMALIZATION AND SOME EXPERIMENTS

Mohammad Shahjahan Feroz Farazi

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

April 2010

Abstract

While classifications are heavily used to categorize web content, the evolution of the web foresees a more formal structure – ontology - which can serve this purpose. Ontologies are core artifacts of the Semantic Web which enable machines to use inference rules to conduct automated reasoning on data. Lightweight ontologies bridge the gap between classifications and ontologies. A lightweight ontology (LO) is an ontology representing a backbone taxonomy where the concept of the child node is more specific than the concept of the parent node. Formal lightweight ontologies can be generated from their informal ones. The key applications of formal lightweight ontologies are document classification, semantic search, and data integration. However, these applications suffer from the following problems: the disambiguation accuracy of the state of the art NLP tools used in generating formal lightweight ontologies from their informal ones; the lack of background knowledge needed for the formal lightweight ontologies; and the limitation of ontology reuse. In this dissertation, we propose a novel solution to these problems in formal lightweight ontologies; namely, faceted lightweight ontology (FLO). FLO is a lightweight ontology in which terms, present in each node label, and their concepts, are available in the background knowledge (BK), which is organized as a set of facets. A facet can be defined as a distinctive property of the groups of concepts that can help in differentiating one group from another. Background knowledge can be defined as a subset of a knowledge base, such as WordNet, and often represents a specific domain.

Keywords

Contextualized Markup Language, GeoWordNet, Background Knowledge, Lightweight Ontology, Faceted Lightweight Ontology

Contributions and publications

This work has been developed in collaboration with prof. Fausto Giunchiglia, Ilya Zaihrayeu, Vincenzo Maltese, Biswanath Dutta, prof. Letizia Tanca, Roberto de Virgilio, Pavel Shvaiko, and Lorenzino Vaccari.

This dissertation makes the following contributions:

- It proposes a novel solution - *faceted lightweight ontology* - to the problems that formal lightweight ontology applications suffer from.
- It develops GeoWordNet, a semantic resource specialized on geo-spatial information. GeoWordNet is a comparatively large knowledge base consisting of hundreds of thousands of concepts and millions of entities, and is designed to be used as background knowledge in the faceted lightweight ontologies.
- It provides the specification of a new language, C-XML (Contextualized Markup Language), for representing faceted lightweight ontologies.
- It presents a formalization of the faceted lightweight ontology. This formalization has a scientific value because it unambiguously represents the constituents of a faceted lightweight ontology.
- It provides an overview of the Semantic Web languages in order to identify a suitable language for representing faceted lightweight ontologies.

- It develops a faceted lightweight ontology in the space domain for the empirical evaluation of this approach.
- It designs and develops algorithms to build the GeoWordNet from external sources such as GeoNames[1].

Part of the materials of this dissertation have been published as follows:

- [15] F. Giunchiglia, F. Farazi, L. Tanca, and R. D. Virgilio. The Semantic Web Languages. In *Semantic Web Information management, a model based perspective.* Roberto de Virgilio, Fausto Giunchiglia, Letizia Tanca (Eds.), Springer Berlin Heidelberg, 2009.
- [17] F. Giunchiglia, V. Maltese, F. Farazi, and B. Dutta. GeoWordNet: a resource for geo-spatial applications. In *ESWC*, Heraklion, Greece, 2010.
- [26] F. Giunchiglia, I. Zaihrayeu, and F. Farazi. Converting classifications into OWL ontologies. In *Proceedings of Artificial Intelligence and Simulation of Behaviour Convention Workshop on Matching and Meaning*, Edinburgh, UK, 2009.
- [45] P. Shvaiko, A. Ivanyukovich, L. Vaccari, V. Maltese, and F. Farazi. A semantic geo-catalogue implementation for a regional SDI. In *INSPIRE Conference*, Krakow, Poland, 2010.

[1]See http://www.geonames.org/

Acknowledgments

Above all I would like to thank my scientific advisor prof. Fausto Giunchiglia for his teachings over the course of these PhD years, especially in relation to how to conduct research and how to turn ideas into research papers. I am grateful for the innumerable hours he has spent patiently teaching me how to organize, structure, and improve my writing. His thoughtful insights introduced me to the emerging and compelling research field of the Semantic Web. Without his support, inspiration, and encouragement it would have been impossible to finish this thesis.

I would also like to thank Ilya Zaihrayeu for the lessons and suggestions he has provided me during these years. I am grateful for his continuous support and feedback in the specification of C-XML.

I am thankful to Vincenzo Maltese for his contribution, collaboration, and continuous feedback in building GeoWordNet. Meaningful discussions with him enabled me finish such a large volume of work on time.

I am also thankful to Biswanath Dutta for all the discussions we had about conceptual analysis in building GeoWordNet. In addition, I thank Viktor Pravdin and Marco Marasca for their technical support in the experimental works of this thesis.

Finally, I would like to express my deep gratitude to the family members who encouraged me to go for doctoral studies. I am especially grateful to my wife, Farzana, who left her job in Bangladesh to accompany me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Context

While classifications are heavily used to categorize web content, the evolution of the web foresees a more formal structure which can serve this purpose, namely *ontology*. Ontology is defined in Computer Science as *a specification of a conceptualization* [28]. Ontologies are core artifacts of the Semantic Web, a proposed extension of the current Web, in which information is given formal semantics so that computers can use inference rules to conduct automated reasoning on pieces of this information [7]. The key factor which makes this reasoning task possible is that ontologies can be represented in the formal language languages (e.g., RDF and OWL).

Lightweight ontologies, proposed in [25], bridge the gap between informal classifications and formal ontologies. A lightweight ontology can be defined as an ontology representing a backbone taxonomy where only an `is-a` subsumption relation holds between a child node and a parent node. As a result, the concept of the parent node is more general than the concept of the child node. Depending upon their usage, lightweight ontologies, can be classified into two kinds: a *descriptive lightweight ontology* and a *classification lightweight ontology*.

Descriptive lightweight ontologies are used to specify the semantics of terms, and the nature and structure of the domain the terms belong to [25]. Thesauri and controlled vocabularies are examples of this kind. They usually have a noun word or a simple noun phrase in each of their node labels, representing (usually) an atomic concept. On the other hand, classification lightweight ontologies are used to describe, categorize, and access collections of documents. Web directories and user classifications are examples of this kind. They usually have a compound noun phrase in each of their labels, representing (usually) a complex concept.

Both the classification and descriptive lightweight ontologies can further be classified into: an *informal lightweight ontology* and a *formal lightweight ontology* [25]. A *formal descriptive lightweight ontology* can be generated from an informal one by converting its organizational structure into a rooted tree, and by converting its natural language node labels into concepts which are represented in a formal language, which belongs to the family of Description Logic languages [3]. Similarly, a *formal classification lightweight ontology* can be generated from an informal one. However, this requires an extra step in which each node is converted into a *concept of node*. A concept of node is the intersection of the concepts of all node labels in the progression from the given node to the root node.

The key applications of formal lightweight ontologies are document classification, semantic search (semantically relevant term or document retrieval), and data integration [25]. We define these applications in the context of lightweight ontologies in the following way. Document classification can be defined as a means of classifying documents to a term/category in a taxonomy, controlled vocabulary, business catalogue, user classification, web directory, or faceted classification. Semantic search can be defined as a means of retrieving the relevant categories, documents, or both, from the lightweight ontologies they are classified into. Data integration can be defined as a means of identifying, and then utilizing, semantic relations (i.e., more general, more specific, equivalent, or disjoint) between terms/categories of the lightweight ontologies, for their integration, interoperation, or merging. All the applications described above depend upon the reasoning on formal lightweight ontologies. In this case, the outcome of the reasoning tasks is conditioned by the accuracy of the axioms encoded through assertion as well as inference, where asserted axioms are retrieved from a knowledge base, e.g., WordNet [38].

1.2 The Problem

Formal lightweight ontologies suffer from the following problems, however.

Disambiguation accuracy of natural language processing: As described in Section 1.1, formal lightweight ontologies are (usually) generated from informal ones. The generation procedure involves the processing of natural language labels for the identification of their concepts. This identification often requires the disambiguation of the label terms (words). However, the disambiguation accuracy of the state of the art NLP tools is limited to a certain extent. This accuracy is influenced by the lack of coverage of the lexical knowledge base that (e.g., WordNet) these NLP tools have.

Insufficient background knowledge for the ontologies: As described in Section 1.1, the applications of formal lightweight ontologies depend upon the reasoning on the axioms extracted from a knowledge base. Due to the insufficiency of the background knowledge needed for the lightweight ontologies, all the necessary axioms can not be extracted. This is the main reason for the relatively low recall of these applications [22].

Difficulties in building and reusing ontologies: The adoption and use of formal lightweight ontologies is limited as a result of the difficulties and costs involved in building them manually. They are often domain specific and cover a particular area of knowledge. An ontology built on a specific domain to fulfil a particular purpose can hardly be reused to fulfil another purpose on a different domain.

Furthermore, because informal lightweight ontologies can be available on the Web, the open-ended and rapidly-changing nature of the Web can influence their character. This requires that they be able to cope with the following factors:

Scalability: A formal lightweight ontology can be generated from an informal one, and can consist of millions of nodes.

Dynamics: A new node might be created, an existing node might be renamed, and/or an existing node might even be removed.

1.3 The Solution

In this thesis, we propose a novel solution to the problem of formal lightweight ontologies, which we call *faceted lightweight ontologies*. To the best of our knowledge, the term was coined by Giunchiglia et al. in [14]. A faceted lightweight ontology can be defined as a lightweight ontology in which both the terms in each node label and their concepts are available in the *background knowledge*, which is organized as a set of facets. A facet can be defined as a distinctive property of the groups of concepts that can help in differentiating one group from another. Background knowledge can be defined as a subset of the (generic) knowledge base, such as WordNet, and usually represents a specific domain.

Background knowledge is organized into two distinct sections: a language-independent part and a language-dependent part. In the language independent-part, knowledge is organized as a set of domains, each domain is grouped into a set of facets, and each facet is constituted by a hierarchy of a set of homogeneous concepts. Instances of a concept are called entities, which are grouped into a set of entity types. A concept can belong to a set (possibly empty) of domains. An entity type can correspond to a concept and a set of entities can be connected to a concept as instances. In the language dependent part, knowledge is organized as a list of words in a given language and grouped into synsets. There are two kinds of synsets: concept synsets and entity synsets. Each concept synset is connected to a concept, but each concept may not have a synset representation in a human language (language gaps). Similarly, each entity synset is connected to an entity, but an entity may not have a synset representation in a human language (language gaps).

We develop GeoWordNet, a comparatively large knowledge base of around 7.1 million synsets, with the full integration of WordNet, Italian MultiWordNet, and GeoNames, specialized in geo-spatial information. It is designed to provide background knowledge to the faceted lightweight ontologies. While using GeoWordNet as background knowledge, we carried-out a number of experiments to verify the feasibility of our solution. In all the experiments we found reasonable performance increases in the lightweight ontology applications. In addition, we specify C-XML for representing faceted as well as informal lightweight ontologies. This solves the problem of dynamics. We also address the issue of scalability while developing software for our experiments.

1.4 Structure of the Thesis

The thesis is structured as follows. In Chapter 2 we discuss the state-of-the-art Semantic Web languages. In Section 2.1, we introduce Web and Semantic Web languages. In Section 2.2 we describe the Semantic Web and its features. In Section 2.3 we provide a hierarchy of the languages which can be used to represent information on the Semantic Web. Sections 2.4, 2.5, and 2.6 outline the Semantic Web languages RDF, OWL, and C-OWL, respectively. Having discussed these three ontology representation languages, we conclude the chapter in Section 2.7.

In Chapter 3 we discuss the different kinds of ontologies that we dealt with in our thesis work. In Section 3.1 we introduce the classification scheme, lightweight ontology, and ontology. In Section 3.2 we provide a description of the classification schemes and ontologies, which is followed by a comparison between them. Section 3.3 discusses lightweight ontolgies, their applications, and the problems involved in their applications. Section 3.4 discusses background knowledge (BK) for the ontologies. In Section 3.5 we present the faceted lightweight ontology as a solution to the problems of the lightweight ontology applications, and in Section 3.6 we conclude the chapter.

In Chapter 4 we describe our approach to converting lightweight ontologies into OWL ontologies. In Section 4.1 we describe the conversion of lightweight ontology nodes, labels, and documents into axioms in OWL. Section 4.2 reports the evaluation results of our approach. Section 4.3 exemplifies the practical applications of the generated OWL ontologies. In Section 4.4 we present related works. In Section 4.5 we add concluding remarks about the lightweight ontology conversion into OWL ontologies.

Chapter 5 provides the specification of C-XML that can be used for representing faceted lightweight ontologies. In Section 5.1 we introduce C-XML. In Section 5.2 we present the hierarchical organization of the objects that can be represented in C-XML, and we specify its abstract syntax. Section 5.3 demonstrates a mapping between C-XML and XML, and in Section 5.4 we conclude the chapter.

In Chapter 6 we describe our approach to building background knowledge from WordNet for faceted lightweight ontologies. In Section 6.1 we analyze the lexical, semantic, and both lexical and semantic relations between synsets in WordNet. We also exemplify the relations and describe the procedure to import the synsets and relations. Section 6.2 presents our observations during this import. In Section 6.3 we report the results of our evaluation of the background knowledge building approach from WordNet. We conclude the chapter in Section 6.4.

Chapter 7 presents our construction of GeoWordNet, with the full integration of the background knowledge developed in Chapter 6, and the knowledge available in GeoNames. In Section 7.1 we describe the criteria for selecting knowledge sources. In Section 7.2 we identify and present facets that can be built out of the GeoNames information. Section 7.3 presents classes that can be organized into the facets described in the previous section. Section 7.4 presents the main research issues we dealt with in importing knowledge from GeoNames. In Section 7.5 we describe the GeoNames knowledge import procedure. In Section 7.6 we report the statistical results of the import, and in Section 7.7 we present the open issues we identified during this import. We conclude the chapter in Section 7.8.

In Chapter 8 we conclude the thesis and provide the direction for future research work.

Chapter 2

State of the art

Web Languages

In this chapter we describe the Web and the Semantic Web, and the means of representing information on each of them. We also discuss the reasons for building the Semantic Web, and its features. We then outline the state-of-the-art languages used to represent information on the Semantic Web.

The chapter is structured as follows. Section 2.1 provides a brief account of the Web, the Semantic Web, and Semantic Web languages. In Section 2.2 we provide a more detailed description of the Semantic Web. In Section 2.3 we outline a hierarchy of the languages that can be used to represent information on the Semantic Web. Section 2.4 presents the data model used in RDF and provides an example of how simple statements can be represented in RDF. In Section 2.5 we describe OWL and its sublanguages, and provide an example of the same RDF statements represented in OWL. In Section 2.6 we outline C-OWL (Context OWL). Section 2.7 concludes the chapter.

2.1 Introduction

The Web is an enormous collection of documents whose number is growing exponentially. Not only can the content of a document be changed, an entire document can be removed from the Web. The effective management of Web documents is essential to improving their use. However, this management task is made difficult by the sheer volume of information and the ever-changing, rapidly growing, and inconsistent nature of the Web. While machines can be used to solve many of today's problems, they are largely unhelpful in solving Web-related problems for a simple reason; the Web was not initially built to be processed by machines.

The Semantic Web [6, 7] is proposed to be built on top of the current Web by its inventor Tim Berners-Lee, and will enable information to be machine-processable by transforming this information from document form to data form. Thus, while the current Web can be considered to be a Web of documents, the Semantic Web can be considered to be a Web of data. Whereas unambiguous meanings cannot be provided for the information on the current Web, unambiguous meanings will be able to be provided for the information on the Semantic Web. For this reason, the Semantic Web is also called the Web of meanings. Recently, researchers have begun to represent Web information in Semantic Web-suitable forms. In order to represent information on the Semantic Web, the knowledge representation languages RDF (Resource Description Framework) and OWL (Web Ontology Language) are used.

2.2 The Semantic Web

The Semantic Web is the final part of its inventor, Tim Berners-Lee's, two-part dream for the Web. In this part he envisions machines having the ability to understand, analyze, infer, and reason about all kinds of data, and the links between data, as represented on the Semantic Web.

Machines can understand information published in machine understandable form. However, only a limited amount of information on the current Web is published in this form. By contrast, all the information on the Semantic Web is published in a form that enables machines to understand and to perform automated analysis on a group of interrelated data. Such automated analysis could involve, for example, the business transactions of a company; a machine might formulate a concluding remark that can assist the person who is dealing with this data, e.g., the managing director of the company.

In this way, the analysis of the data moves to the level of logical analysis, and can thereby provide optimally correct answers to the user's queries in relation to the information space. To further improve logical analysis, information can be represented as simple logic formulas that enable the analyzer to understand the disambiguated meaning of the data. Semantic Web information is proposed to be published in RDF, which provides the infrastructure for representing data as simple logic formulas in order to keep track of meaning.

The descriptive power of RDF is kept to a minimum so as to ensure that the Semantic Web remains a flexible, unconstrained medium of representing knowledge. Besides, a more powerful form of RDF could cause Semantic Web applications to behave in unexpected ways. Even with its current descriptive power, RDF use can result in convoluted, unanswerable questions due to the large and complex amounts of data produced from RDF applications.

To further limit the expressive power of RDF in order to make its behavior predictable on the Semantic Web, schema languages (e.g., XML Schema and RDF Schema), are proposed. Schema languages provide a predefined set of terms to describe the elements of Web pages.

Even though schema languages on the Semantic Web can help improve interoperability, they cannot help build common understanding because they are unable to provide the meaning of terms, and therefore terms cannot be linked. Inference languages such as RDF can help link terms. Moreover, inference languages provide the necessary infrastructure to express the fact that while two terms might have different lexicons, their meanings are the same.

Inference layer on the Semantic Web supports the representation of different definitions for the same thing by different websites, and enables machines to identify these divergent definitions and link them. With the translation support of inference layer, the Semantic Web allows the use of globally-used standard terms in combination with locally-used terms. Inference layer also supports the representation of inference rules which enable machines to reason about the data represented on the Semantic Web.

Among its other features, the Semantic Web has the capacity to evolve. It can learn regardless of the natural language used to represent data, as it can relate the use of one term with another term from different sources. Because it enables machines to learn as well as to understand, analyze, infer, and reason, Semantic Web agents can be built that are able to: (i) maintain the daily schedule of an individual; (ii) suggest improvements to his/her schedule; (iii) provide an individual with reminders about his/her next appointment; and so on.

2.3 The Hierarchy of Languages

At present, there exists a series of knowledge representation languages for the Semantic Web. These are: XML [9], XML Schema [13], RDF [5] , RDF

Schema [10], OWL [37], and C-OWL [8].

A. Documents: XML XML is an acronym for Extensible Markup Language. It is designed to support various applications, mostly executed on the Internet, by providing a means of encoding information in the form of documents, which are not only understandable by the machines, but also readable by humans. Information encoding is accomplished using customized tags. Customized tag support is used to exchange a wide variety of information on the Web and elsewhere. The statements "GeoNames has coverage of all countries" and "It was modified on April 25, 2009" can be represented in XML using the tags 'GeoNames', 'coverage', and 'modified', and a preceding statement to represent that the following information is in XML, along with the XML version used to represent this information. The preceding statement is called a prolog and the remaining statements are called elements. Each document must contain a root element containing all the other elements called non-root elements. In the following XML encoding the first line is the prolog, the GeoNames tag is the root element, and coverage and modified tags are the non-root elements.

```
<?xml version="1.0" ?>
  <GeoNames>
    <coverage>Countries</coverage>
    <modified>April 25, 2009</modified>
  </GeoNames>
```

The purpose of XML Schema is to define a set of rules to which an XML document conforms. An XML Schema is similar to a class in an object-oriented programming language, and an XML document is similar to an instance of that class. XML Schema is used for exchanging information between interested parties who have agreed to a predefined set of rules. However, the absence of the meaning of the vocabulary terms used in XML

Schema makes it difficult for machines to accomplish intercommunication when new XML vocabulary terms are introduced. On one hand machines cannot differentiate between polysemous terms, and on the other hand they cannot combine synonymous terms.

B. Objects and Relations: RDF(S) RDF is an acronym for Resource Description Framework, and RDFS is an acronym for RDF Schema. We use RDF(S) to represent both RDF and RDFS. The goal of RDF(S) is to provide meaning to the data represented in RDF, in order to overcome the drawback (absence of meaning) of XML. RDF is used to: (i) describe information about Web resources and the systems that use these resources; (ii) make information machine-processable; (iii) provide internetworking among applications; and (iv) provide automated processing of Web information by intelligent agents. RDF is designed to provide flexibility in representing information. Its specification is given in [5, 36, 34, 30, 10, 27].

RDF Schema is an extension of RDF. It provides a vocabulary for RDF to represent classes of the resources, subclasses of the classes, properties of the classes, and relations between properties. The capability of representing classes and subclasses allows users to publish ontologies on the Web. However, these ontologies have limited use as RDFS cannot represent information containing disjointness and specific cardinality values.

C. Ontologies: OWL

Lightweight Ontologies: Propositional OWL Lightweight ontologies [25] can be defined as concepts, and relations between them form subsumption hierarchy. In lightweight ontology, instances assigned to a parent node concept are a superset of the instances assigned to a child node concept. In comparison to full-fledged ontology, all possible relations in a lightweight ontology are a subset of all possible relations in a full-fledged ontology.

Full Ontologies: All OWLs OWL is an acronym for Web Ontology Language. It begins to approach cardinality and disjoint class issues by providing richer syntax than RDF Schema. OWL provides the syntax to specify that two or more classes are disjoint, and to specify cardinality (e.g., "exactly one") constraints. Moreover, OWL provides greater machine interpretability of Web content than RDF and RDF Schema. However, there are no built-in primitives for part-whole relations in OWL [43]. Furthermore, ontologies represented in OWL cannot deal with context dependent data [8]. OWL specification is given in [37, 46, 4, 40, 11, 31].

D. Semantically Heterogeneous Ontologies: C-OWL The Context OWL (C-OWL) [8] is an extension of OWL. It provides even richer syntax and semantics than OWL. It represents OWL ontologies and the mappings between these ontologies, where each ontology represents a localized view of the domain. An ontology representing a localized view of a domain is called a contextual ontology. C-OWL permits restriction of the visibility of the ontologies to the outside. By the same token, it allows limited and controlled access by providing explicit mappings.

2.4 RDF(S)

RDF is a language for representing data in the Semantic Web. RDF is designed to provide: (i) a simple data model so that users can make statements about Web resources; and (ii) the capability to perform inference on the statements represented by users.

The data model in RDF is a graph data model. The graph used in RDF is a directed graph. A graph consists of nodes and edges. Statements about resources can be represented using graph nodes and edges. Edges in RDF graphs are labeled. An edge with two connecting nodes form a triple. Between two nodes, one node represents the subject, the other node represents the object, and the edge represents the predicate of the statements. As the graph is a directed graph, the edge is a directed edge and the direction of the edge is from subject to object. The predicate is also called the property of the subject, or a relationship between subject and object.

RDF uses URI references to identify subjects, objects, and predicates. The statement "GeoNames has coverage of all countries" can be represented in RDF, where 'GeoNames' is a subject, 'countries' is an object and 'coverage' is a predicate. The URIs of the subject 'GeoNames', object 'countries', and predicate 'coverage' are "http://www.geonames.org", "http://www.geonames.org/countries", and "http://purl.org/dc/terms/coverage", respectively.

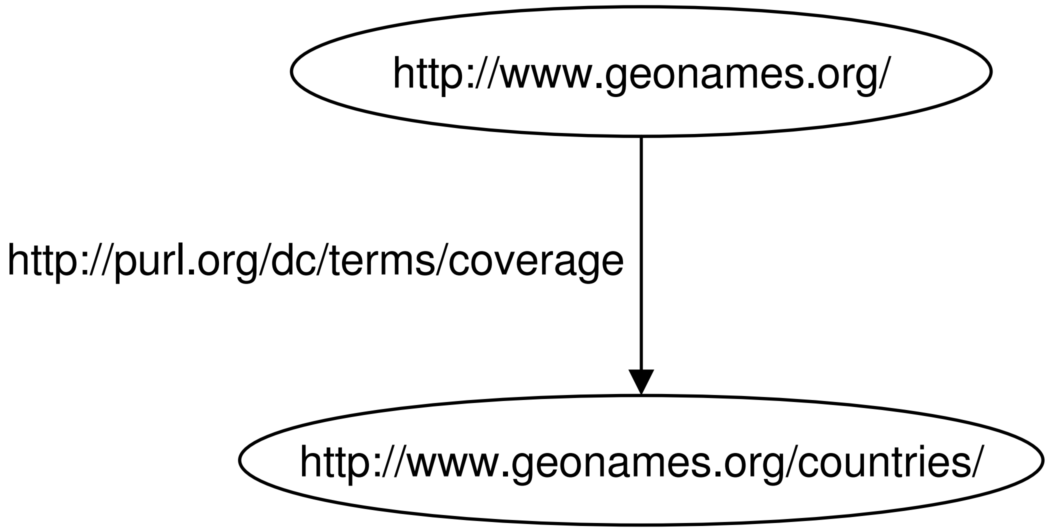
http://www.geonames.org/
http://purl.org/dc/terms/coverage
http://www.geonames.org/countries/

Figure 2.1: Graph data model of a statement represents subject, object, and predicate as URIs.

Objects in RDF statements can be literals. In the statement "GeoNames was modified on April 25, 2009”, ‘GeoNames’ is a subject, ‘modified’ is an object, and ‘April 25, 2009’ is a predicate which is a literal. The URIs of the subject ‘GeoNames’ and predicate ‘modified’, are “http://www.geonames.org” and “http://purl.org/dc/terms/modified”, respectively. The object ‘April 25, 2009’ can be represented as is, without a URI.

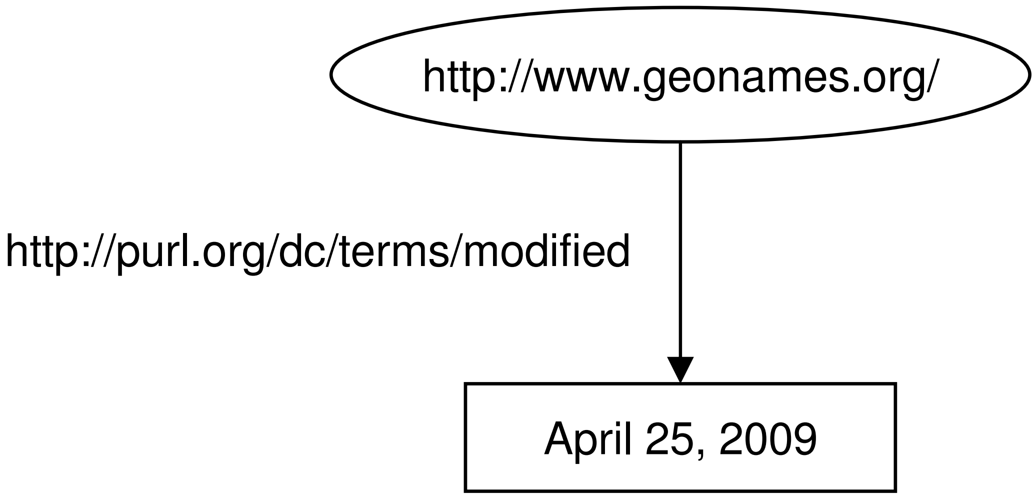

Figure 2.2: Graph data model of a statement represents subject and predicate as URIs and object as a literal.

Statements about GeoNames can be described in RDF using the constructs rdf:Description, rdf:resource, rdf:about, and rdfs:label, as provided below:

```
<?xml version=“1.0”?>
<rdf:RDF
    xmlns:rdf=“http://www.w3.org/1999/02/22-rdf-syntax-ns#”
    xmlns:rdfs=“http://www.w3.org/2000/01/rdf-schema#”
    xmlns:dc=“http://purl.org/dc/terms#”>
 <rdf:Description rdf:about=“http://www.geonames.org”>
  <rdfs:label>GeoNames</rdfs:label>
 <dc:coverage rdf:resource=“http://www.geonames.org/countries”/>
```

```
    <dc:modified>April 25, 2009</dc:modified>
  </rdf:Description>
</rdf:RDF>
```

2.5 OWL

OWL is a Semantic Web language for representing Web documents as full-fledged ontologies. To represent the various descriptions of the document content, OWL provides a richer set of vocabulary (with forty language constructs) than its predecessors, and consequently generates much richer ontologies. In addition, some OWL ontologies can be mapped to specific logic languages, which enable them to use the reasoning support tools used for that language.

Ontologies generated in OWL DL, a sublanguage of OWL, can be mapped to description logic. OWL DL is designed to use the representation, reasoning power, and tools support of description logic. The modeling constructs of the OWL Lite sublanguage of OWL, are a proper subset of OWL DL. For this reason, OWL Lite can also be mapped to a subset of description logic to which OWL DL can also be mapped. Like OWL DL, OWL Lite is able to use the reasoning tools of description logic. OWL Full is another sub-language of OWL, along with OWL Lite and OWL DL. Like RDF, data representation in all three sublanguages of OWL is in triple form: subject, object, and predicate. More detailed descriptions of all three sub-languages of OWL, OWL Lite, OWL DL, and OWL Full, are described below.

OWL Lite

Data representation in OWL Lite is able to use a subset of OWL and RDF(S) vocabulary. It can use thirty-five OWL constructs out of forty, and eleven RDF(S) constructs out of thirty-three (not including the subproperties of the property `rdfs:member`). RDF(S) construct `rdfs:Class` cannot be used to define a class in OWL Lite. Instead, OWL construct `owl:Class` is used for this purpose. Five OWL constructs – `complementOf`, `disjointWith`, `hasValue`, `oneOf`, and `unionOf`, cannot be used in it.

Some of the OWL Constructs that are able to be used in OWL Lite have a limited use. All three cardinality constructs, `cardinality`, `maxCardinality`, and `minCardinality`, can only take the non negative integer quantity, 0 or 1, in their value fields. Constructs `equivalentClass` and `intersectionOf` also have restricted uses. They cannot be used in a triple if the subject or object represents an anonymous class.

The restrictions imposed on OWL Lite make it easy for users who want to publish simple hierarchical structures in OWL, and to do so at reduced cost and time. It offers low computational complexity, but guaranteed computations.

OWL DL

OWL DL representation can use all eleven RDF(S) constructs that OWL Lite can use. Like OWL Lite, it uses only the `owl:Class` construct to define a class. It can also use all forty OWL constructs. However, some of these forty constructs have restricted use in OWL DL. Classes in it cannot be used as individuals, and vice vera. Each individual must be an extension of a class. Even if an individual cannot be classified under any user-defined class, it must be classified under the class of all classes in OWL, that is, the `owl:Thing` class. Individuals cannot be used as properties, and vice versa. Likewise, properties cannot be used as classes, and vice versa.

Properties in OWL DL are differentiated as data type properties and object properties. Object properties connect class instances and data type properties connect instances to data literals.

However, the restrictions applied in OWL DL are there to maintain a balance between expressivity and computational completeness. Even though its computational complexity is higher than OWL Lite, it offers more expressive power than OWL Lite. However, OWL DL's expressiveness is limited to a certain level so that the computations remain complete and decidable. While remaining as a complete and decidable sublanguage, it permits the use of the `intersectionOf` construct to put any number of classes in it. OWL DL also allows the use of any non negative integer in the cardinality restrictions value fields.

OWL Full

OWL Full can use all forty OWL constructs and all eleven RDF(S) constructs without the restrictions imposed on OWL Lite and OWL DL. Moreover, the construct `rdfs:Class` can be used to define a class while `owl:Class` is another choice to define the same thing. These supports make it more expressive than OWL DL. Properties can be assigned to classes, a class can be represented as an individual or a property, and vice versa.

To gain more expressive power, OWL Full sacrifices computational completeness. The computational complexity of this sublanguage is higher than the other two sublanguages. In OWL Full, computations can not be guaranteed to conclude. However, applications that need more expressivity than OWL DL offers, and do not require a guaranteed conclusion, can use OWL Full.

Statements about GeoNames can be represented in OWL using the constructs owl:Ontology, owl:Thing, rdfs:labels, and rdf:resource as provided below:

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:dc="http://purl.org/dc/terms#">
    <owl:Ontology rdf:about=""/>
    <owl:Thing rdf:about="http://www.geonames.org">
      <rdfs:label>GeoNames</rdfs:label>
    <dc:coverage rdf:resource="http://www.geonames.org/countries"/>
      <dc:modified>April 25, 2009</dc:modified>
    </owl:Thing>
  </rdf:RDF>
```

2.6 C-OWL

C-OWL representation can support the use of some mapping constructs beyond OWL and RDF(S) constructs. The mapping constructs are used to represent mapping relations between the concepts, individuals, and properties of two different ontologies. The mapping relations are more specific, more general, equivalent, disjoint, and compatible. C-OWL statements built with the mapping relations are called bridge rules.

Excluding bridge rules, ontologies published in C-OWL can be in one of the sublanguages of OWL. However, two ontologies connected via bridge rules must be in the same sublanguage. An ontology published in an OWL sublanguage, which is connected to a group of one or more ontologies published in that sublanguage via a set of bridge rules, form an ontology called contextual ontology. OWL sublanguages cannot support the representation of a contextual ontology containing a non-empty set of bridge rules. However, C-OWL can represent contextual ontology. Thus C-OWL is more expressive than any sublanguage of OWL. Like OWL, data representation in C-OWL is in triple form.

Statements about GeoNames can be represented in C-OWL using the expressive power of OWL, and their C-OWL representation remains the same, as shown in the OWL example in the previous section. Below is a mapping example taken from the C-OWL paper [8]. In this contextual ontology, two ontologies Wine and Vino, describing the same thing, wine, are mapped. For the detailed description we refer to the C-OWL paper.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:cowl="http://www.example.org/wine-to-vino.map#">
<cowl:mapping>
  <rdfs:comment>Example of a mapping of wine into vino</rdfs:comment>
  <cowl:sourceOntology rdf:resource="http://www.example.org/wine.owl"/>
  <cowl:targetOntology rdf:resource="http://www.example.org/vino.owl"/>
 <cowl:bridgeRule cowl:br-type="equiv">
   <cowl:sourceConcept rdf:resource="http://www.example.org/wine.owl#wine"/>
   <cowl:targetConcept rdf:resource="http://www.example.org/vino.owl#vino"/>
 </cowl:bridgeRule>
 <cowl:bridgeRule cowl:br-type="onto">
   <cowl:sourceConcept rdf:resource="http://www.example.org/wine.owl#RedWine"/>
   <cowl:targetConcept rdf:resource="http://www.example.org/vino.owl#VinoRosso"/>
 </cowl:bridgeRule>
 <cowl:bridgeRule cowl:br-type="into">
   <cowl:sourceConcept rdf:resource="http://www.example.org/wine.owl#Teroldego"/>
   <cowl:targetConcept rdf:resource="http://www.example.org/vino.owl#VinoRosso"/>
 </cowl:bridgeRule>
 <cowl:bridgeRule cowl:br-type="compat">
   <cowl:sourceConcept rdf:resource="http://www.example.org/wine.owl#WhiteWine"/>
   <cowl:targetConcept rdf:resource="http://www.example.org/vino.owl#Passito"/>
 </cowl:bridgeRule>
 <cowl:bridgeRule cowl:br-type="incompat">
   <cowl:sourceConcept rdf:resource="http://www.example.org/wine.owl#WhiteWine"/>
   <cowl:targetConcept rdf:resource="http://www.example.org/vino.owl#VinoNero"/>
 </cowl:bridgeRule>
</cowl:mapping>
</rdf:RDF>
```

2.7 Conclusion

In this chapter we have presented a description of the Semantic Web. The description highlights the design goal, capability, features, and languages of the Semantic Web. A hierarchy of the Semantic Web languages has also been provided. In this hierarchy, it has been demonstrated that, starting from XML, successive languages have emerged to overcome the limitations of their predecessors. We have also provided a brief description of the Web knowledge representation languages RDF, OWL (and the sublanguges OWL Lite, OWL DL, OWL Full), and C-OWL, that can be used to represent information on the Semantic Web. However, none of these languages enable us to encode faceted lightweight ontologies (which were introduced in Section 1.3), and which are outlined in further detail in the next Chapter. For the purpose of representing faceted lightweight ontologies, we designed a new language, C-XML, which is described in Chapter 5.

Chapter 3

Faceted LO

Having provided a brief description of the web knowledge representation languages in the previous chapter, in this chapter we describe some formalisms; the classification schemes and ontologies used for representing knowledge. We then discuss lightweight ontologies, their applications, and the problems these applications face. Finally, we propose faceted lightweight ontologies as the solution to these problems.

The chapter is organized as follows. In Section 3.1 we briefly describe the classification scheme and ontology. Section 3.2 provides a comparison between the classification schemes and ontologies. In Section 3.3, we describe lightweight ontologies. Section 3.4 provides the structure of the background knowledge and Section 3.5 details the faceted lightweight ontology.

3.1 Introduction

A *classification scheme*, or a *classification* for short, is a rooted tree whose nodes are assigned natural language labels and are populated with a (possibly empty) set of documents. Since the invention of classification by Aristotle in the 4th century BC, classifications continue to be used pervasively to represent various kinds of human knowledge. For example, classifications have been used in libraries (DDC[1], LCC[2], and Colon classification[3]), in Personal Knowledge Management (favorites, personal e-mails, and folder hierarchies), and, lately, on the Web (Amazon[4], Google[5], and Yahoo[6]).

As already discussed in Section 1.1, while classifications are extensively used to organize web contents, the Web's evolution to a more formal structure, ontology, can serve this purpose. As core artifacts of the Semantic Web, ontologies provide the formal semantics for information, and thereby enable computers to use inference rules to conduct automated reasoning. The key factor which makes this possible is the capacity of ontologies to be expressed in a formal language suited to automated reasoning.

As we have seen in Section 1.1, lightweight ontologies, proposed in [25], can bridge the gap between informal classifications and formal ontologies. The faceted lightweight ontology, an extension of the lightweight ontology, can be used to overcome the limitations of an ontology; that ontologies built for one purpose can rarely be reused for another purpose. Moreover, the faceted lightweight ontology addresses the problem of lack of background knowledge of ontologies. In this chapter, we provide a formalization of the faceted lightweight ontology that can help the user to automate its use.

3.2 Classifications vs Ontologies

In this section we discuss commonalities and differences between classifications and ontologies. In order to ground our discussion on well-defined terms, we give the definitions of these two kinds of artifacts below.

[1]See http://www.tnrdlib.bc.ca/dewey.html.
[2]See http://www.loc.gov/catdir/cpso/lcc.html.
[3]See http://www.iskoi.org/doc/colon.htm.
[4]See http://www.amazon.com.
[5]See http://www.google.com.
[6]See http://www.yahoo.com.

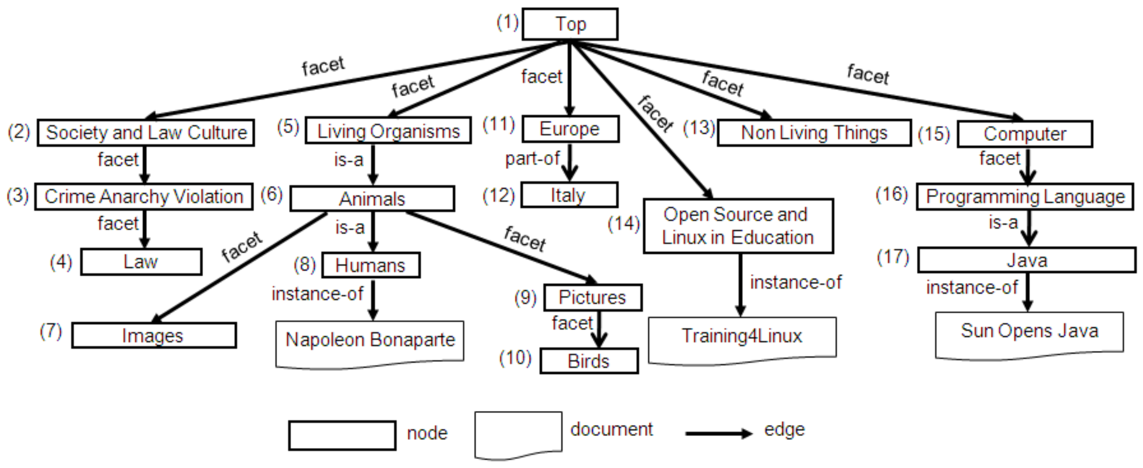

Figure 3.1: An example of a classification with link semantics made explicit.

A *classification* is a 5-tuple $C = \langle N, E, L, D, cl\rangle$ where N is a finite set of nodes, E is a set of edges on N, such that $\langle N, E\rangle$ is a rooted tree; L is a finite set of labels expressed in natural language, such that for any node $n_i \in N$, there is one and only one label $l_i \in L$; D is a set of documents and cl is a function which maps every $d_i \in D$ to a non-empty set of nodes $\{n_i\} \subseteq N$. In Figure 3.1 we show an example of a classification. Although classifications have no explicit formal semantics for edges, in this example we have labeled each edge with the name of a hypothetical relation that may hold between the linked nodes.

An *ontology* is an *explicit specification of a conceptualization* [28]. They are often thought of as directed graphs whose nodes represent *concepts* and whose edges represent formal *relations* between concepts. The backbone structure of the ontology graph is a taxonomy in which all the relations are `sub-class-of`, whereas the remaining structure of the graph supplies auxiliary information about the modeled domain and may include relations like `part-of`, `located-in`, `is-parent-of`, and others [29]. Classes can be associated with instances through the `instance-of` relation. In Figure 3.2 we provide an example of a small ontology.

Even if both ontologies and classifications can often be represented in

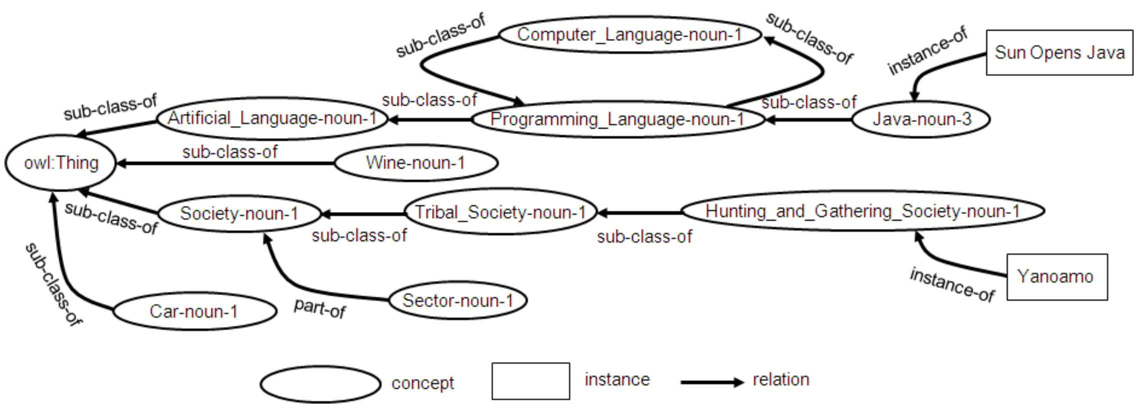

Figure 3.2: An example of an OWL ontology.

the form of a graph, ontologies and classifications remain quite different in their uses, purpose, language, applications, and in other aspects which we summarize as follows:

- **Users:** A typical user of classifications is a human (e.g., a classifier in a library classification), whereas ontologies are primarily used by machines and, as such, are the key enablers of the Semantic Web. Moreover, designing a classification is part of the everyday practice of many computer users, whereas designing a full-fledged ontology (expressed, for example, in OWL-DL) is a difficult and error-prone task even for ontology experts [44].

- **Purpose:** Classifications are primarily used for the organization of (large) document collections into categories and subcategories so that these documents can be easily accessed by humans through browsing the classification tree in a top-down fashion. Ontologies are primarily used for modeling a particular domain so that the resulting model represents a shared view of a group of individuals [42].

- **Language:** As already mentioned, classifications use natural language to describe nodes' categories. Natural language is well understood by humans but is hard to be "understood" and reasoned about by machines due to its ambiguous nature. By contrast, ontologies are codified in a formal language which is unambiguously interpreted by machines. In fact, because ontologies are expressed in a formal language, they are often used for automated *reasoning* about the domain they model. Natural language is used in ontologies to a limited extent (e.g., to describe concept names) and, in general, has no functional value in reasoning operations on ontologies.

- **Nodes:** In an ontology, nodes normally represent atomic concepts (e.g., `car`, `wine`) whose names are shown next to the corresponding nodes when ontologies are visualized. In a classification, a label can represent a rather complex concept (e.g., "Open Source and Linux in Education") or an individual (e.g., "Napoleon Bonaparte").

- **Edges:** In an ontology graph, edges have well-defined semantics and they usually encode `sub-class-of`, `part-of` and other relations that hold between the two concepts connected by an edge. In a classification, an edge implicitly represents either: (i) a *specification* relation which can be thought of as an `is-a` relation (e.g., an edge from "Animals" to "Humans") or as a `part-of` relation (e.g., an edge from "Europe" to "Italy"); or, (ii) a *facet* relation which encodes the fact that the label of the child node represents an aspect of meaning of the parent node (e.g., an edge from "Animals" to "Images") [18]. It is bad practice to connect two nodes whose labels denote disjoint concepts (e.g., "non-living things" and "living organisms"), as in this case the child node and all its descendants cannot be populated with any document in a meaningful way.

- **Instances:** In an ontology, node instances are representatives of the node class and of all its ancestor classes in the `sub-class-of` hier-

Table 3.1: Comparison between classification schemes and ontologies.

Category	**Classification Schemes**	**Ontologies**
Users	Humans	Machines
Purpose	Organization of (large) document collections	Modeling of a domain
Language	Natural language, e.g. English	Formal language, e.g. OWL
Nodes	Usually represent complex concepts or individuals	Usually represent atomic concepts
Edges	Do not have well defined semantics	Have well defined semantics
Instances	Are not necessarily instances of the class in which they are populated	Are instances of the class in which they are populated
Examples	DDC, LCC, Colon classification	Gene ontology[a], OpenCyc ontology[b], MeSH ontology

[a]http://www.geneontology.org/
[b]http://www.opencyc.org/

archy. They are in the `instance-of` relation with the class(es) they belong to. In a classification, node instances are not necessarily representatives of the class denoted by the node label, and can be documents which are about objects described by the set of labels of the nodes on the progression from the given node to the root. For example, a node labeled “birds” may be populated with pictures of birds if the label of the parent node is “pictures”.

As shown above, classifications and ontologies are quite different and both have their pros and cons with respect to each other. We summarize their distinguishable features in Table 3.1 and, in the next section, we describe lightweight ontologies which bridge the gap between classifications and ontologies.

3.3 Lightweight Ontologies (LO)

A *lightweight ontology* [25] is an ontology representing a backbone taxonomy where only an `is-a` subsumption relation holds between a child node and a parent node. As a result, the concept of the parent node is more general than the concept of the child node. It can be formally defined as a triple $LO = \langle\ LN,\ LE,\ LC\ \rangle$, where: LN is a finite (possibly empty) set of nodes; LE is a set of edges, each of which represents a relation between nodes to form a rooted tree $\langle\ LN,\ LE\ \rangle$; and LC is a finite set of concepts, encoded in a formal language FL, corresponding to nodes such that for each node $ln_i \in LN$ there is one and only one concept $lc_i \in LC$, and $lc_i \sqsubseteq lc_j$, if ln_i is the child node of ln_j.

Depending upon the usage, lightweight ontology can be classified into two kinds: a descriptive lightweight ontology and a classification lightweight ontology [25]. Descriptive lightweight ontologies are mainly used to define the meaning of words in a specific domain. Thesauri is an example of this kind. Classification lightweight ontologies are used to categorize documents containing data items, to make them accessible to users. Web directories are an example of this kind. Each above mentioned kind can be further classified into two kinds: an informal lightweight ontology and a formal lightweight ontology. Here we will attempt to make their notion clear through examples. Taxonomies, controlled vocabularies, business catalogues, faceted classifications, user classifications, Web directories, thesauri, (ordinary) glossaries, XML DTDs, and Database Schemas are examples of the informal lightweight ontology. Frames, data models, and general logics are examples of the formal lightweight ontology.

An informal lightweight ontology can be converted to a formal one, whereas the conversion of a classification lightweight ontology requires one more step than is required by a descriptive lightweight ontology [25]. A descriptive lightweight ontology requires the following steps: if its terms are not organized as a single rooted tree, it is converted into such a tree, and the natural language label of each term is converted into a concept in *FL*. Note that *FL* is the subset of the description logic (DL) language, excluding roles, called the propositional DL language. Beyond these steps, a classification lightweight ontology requires the computation of node concepts, where each node concept is equal to the intersection of the concepts associated with the term labels on the path from the root node to the node itself.

The key applications of formal lightweight ontologies are document classification, semantic search (relevant term or document retrieval), and data integration [25]. We define them in the context of lightweight ontologies as follows. Document classification can be defined as a means of classifying documents into a term/category in a taxonomy, controlled vocabulary, business catalogue, user classification, web directory, or faceted classification. Semantic search can be defined as a means of retrieving relevant categories, documents, or both, from the lightweight ontologies they are classified into. Data integration can be defined as a means of identifying and then utilizing semantic relations (i.e., more general, more specific, equivalent, or disjoint) between terms/categories of the lightweight ontologies for their integration, inter-operation, or merging.

3.4 Background Knowledge (BK)

All the applications described in the previous section depend upon the reasoning on formal lightweight ontologies. The outcome of the reasoning tasks is heavily influenced by the axioms encoded through assertion as well as inference. Asserted axioms are retrieved from a knowledge base initially built with the concepts and axioms imported from WordNet [38], which covers various domains of knowledge. The user of a lightweight ontology

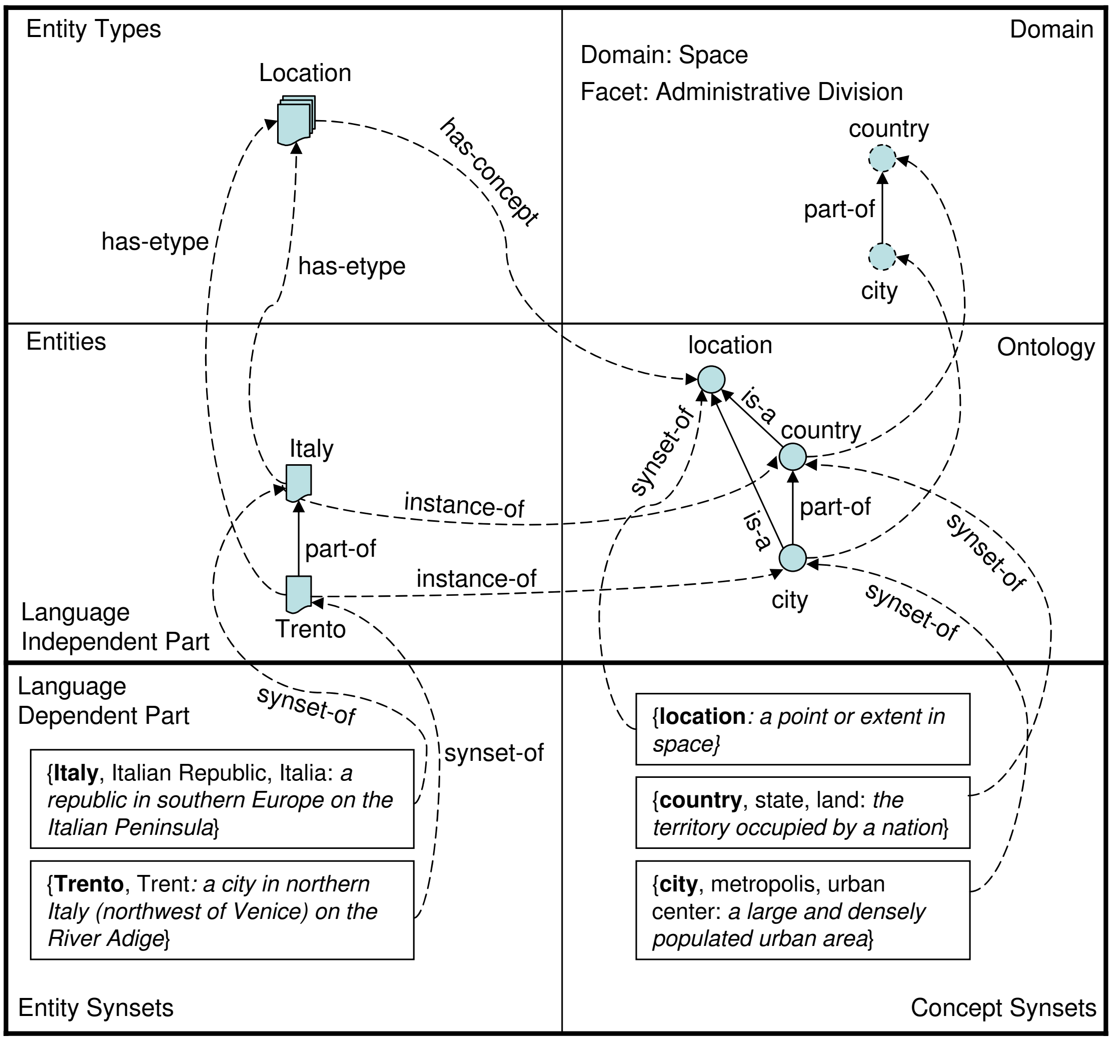
Entity Types
Domain
Location
Domain: Space
Facet: Administrative Division
has-concept
country
part-of
city
has-etype
has-etype
Entities
location
Ontology
is-a
country
synset-of
Italy
instance-of
part-of
is-a
part-of
instance-of
synset-of
city
synset-of
Language Independent Part
Trento
Language Dependent Part
synset-of
synset-of
{**Italy**, Italian Republic, Italia: *a republic in southern Europe on the Italian Peninsula*}
{**Trento**, Trent: *a city in northern Italy (northwest of Venice) on the River Adige*}
{**location**: *a point or extent in space*}
{**country**, state, land: *the territory occupied by a nation*}
{**city**, metropolis, urban center: *a large and densely populated urban area*}
Entity Synsets
Concept Synsets

Figure 3.3: Organization of background knowledge.

might be interested in the domain(s) his/her ontology belongs to. A subset of a knowledge base representing the domain(s) specific knowledge a user is interested in for his/her own ontology is called *background knowledge (BK)*. BK can be modified by users.

As has already been discussed in Section 1.3, and as can be seen in Figure 3.3, BK is organized into two distinct parts: a language-independent part and a language-dependent part. In the language-independent part, knowledge is organized as a set of domains, each domain is grouped into a set of facets, and each facet is constituted by a hierarchy of a set of homogeneous concepts [14]. Instances of concepts are called entities and are grouped into a set of entity types. Each concept can belong to a (possibly empty) set of domains. An entity type can correspond to a concept and a set of entities can be connected to a concept (i.e., its instances). In the language-dependent part, knowledge is organized as a list of words in a given language grouped into synsets. There are two kinds of synsets: concept synsets and entity synsets. Each concept synset is connected to a concept, but each concept may not have a synset representation in a human language (language gaps). Similarly, each entity synset is connected to an entity, but an entity may not have a synset representation in a human language (language gaps).

In the Ontology part of the figure described above location, country, and city represent concepts. All the concepts in the ontology part are shown as circles and all the concepts in the Domain part are shown as dashed circles. Links between the objects within a part are shown as solid straight arrows and links across the parts are shown as dashed curved arrows. In the Entity part, Italy and Trento represent entities, where Italy is an instance of the concept country, Trento is an instance of the concept city, and the relation part-of connects the entities Italy and Trento.

3.5 Faceted Lightweight Ontologies

A *faceted lightweight ontology* [14] is a lightweight ontology in which terms, present in each node label, and their concepts, are available in the BK which is organized as a set of facets. It can be formally defined as a quintuple $FLO = \langle\ LN,\ LE,\ LT,\ LC^{FL},\ BK^{F}\ \rangle$, where LN is a finite (possibly empty) set of nodes, LE is a set of edges representing relations between nodes to form a rooted tree $\langle\ LN,\ LE\ \rangle$, LT is a set of terms, LC^{FL} is a finite set of concepts encoded in a formal language FL, such that for each term $lt_i \in LT$ there is one and only one concept $lc_i \in LC^{FL}$ and BK^F is background knowledge organized as a set of facets F such that $LT \in BK^F$ and $LC^{FL} \in BK^F$.

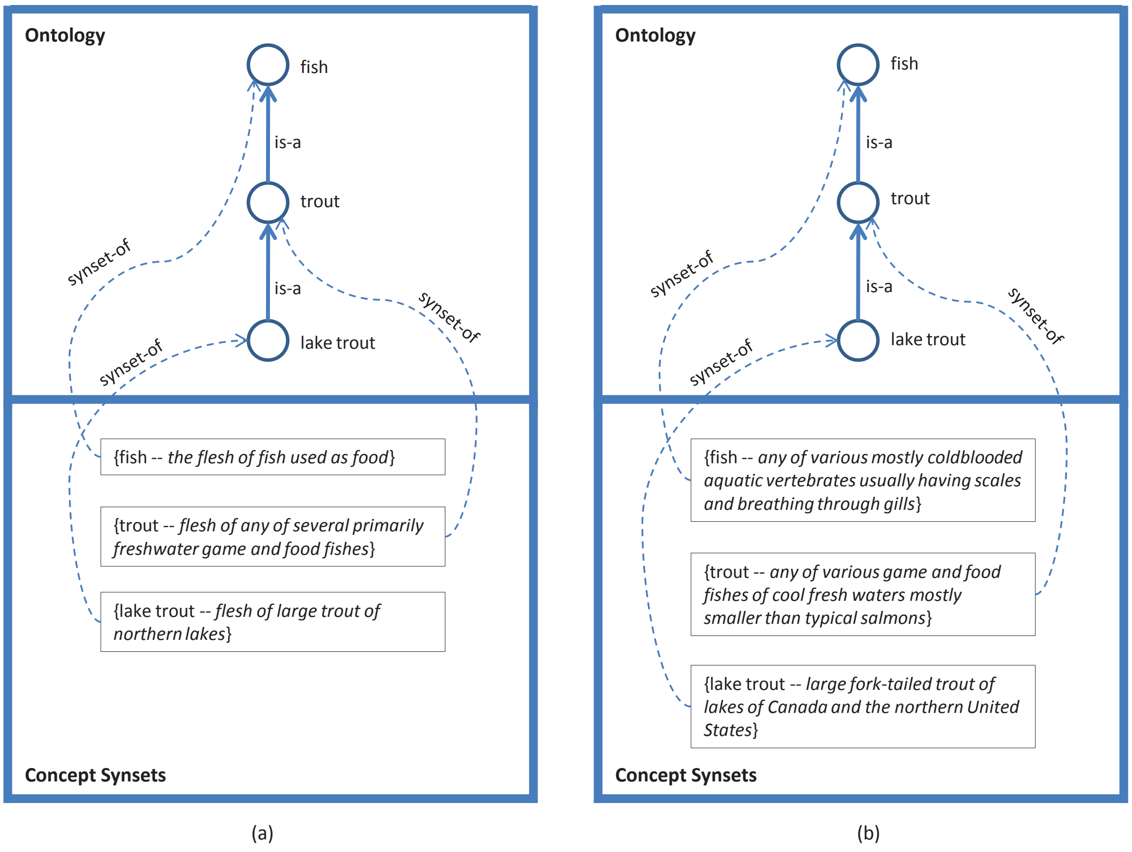
Ontology
fish
is-a
trout
is-a
lake trout
synset-of
synset-of
synset-of
{fish -- the flesh of fish used as food}
{trout -- flesh of any of several primarily freshwater game and food fishes}
{lake trout -- flesh of large trout of northern lakes}
Concept Synsets
(a)
Ontology
fish
is-a
trout
is-a
lake trout
synset-of
synset-of
synset-of
{fish -- any of various mostly coldblooded aquatic vertebrates usually having scales and breathing through gills}
{trout -- any of various game and food fishes of cool fresh waters mostly smaller than typical salmons}
{lake trout -- large fork-tailed trout of lakes of Canada and the northern United States}
Concept Synsets
(b)

Figure 3.4: An example of a faceted lightweight ontology in (a) food domain; and (b) animal domain.

It can be seen from the definition given above that a faceted lightweight ontology comprises background knowledge and a lightweight ontology. Background knowledge plays a major role in faceted lightweight ontologies. Given that there is a term *fish* in a label of a node in the hierarchy of a lightweight ontology. This term represents *aquatic vertebrate* when background knowledge, attached to the lightweight ontology, is in the *animal* domain. The same term represents *the flesh of fish used as food* when the background knowledge is in the *food* domain. Therefore, by replacing the existing background knowledge with a new one selected from a different domain, we enable the same lightweight ontology to be reused for another purpose. Figure 3.4 exemplifies how a faceted lightweight ontology can be used for a variety of purposes. For the sake of simplicity, we have provided only the semantics of the lightweight ontology terms in different domains, instead of the faceted background knowledge hierarchies.

3.6 Conclusion

In this chapter we have provided a brief description of the classification schemes and ontologies, and have presented a comparison between them. We have described lightweight ontologies, their applications, and the problems involved in their applications. We have proposed faceted lightweight ontologies as a solution to overcome these limitations.

Chapter 4

Lightweight and OWL Ontologies

In the previous chapter we provided the definitions of classifications, lightweight ontologies(LO), and ontologies. In this chapter we describe how classifications can be converted into ontologies. We evaluate the conversion approach by providing some experimental results. In the evaluation, among other things, we demonstrate which OWL sublanguage can be used to represent classifications.

In Section 4.1 we describe how to convert classification schemes into OWL ontologies and how the generated OWL ontolgies can be enriched with additional axioms. In Section 4.2, we report the experimental results. Section 4.3 outlines how this work helps to optimize classifications. In Section 4.4 we discuss related work, and we conclude the chapter in Section 4.5.

4.1 LO to OWL Ontology

In this section we demonstrate how a classification, as defined in the previous chapter, can be converted into an OWL ontology. In particular, we show how classification elements, namely labels, nodes, edges, documents, and document-node links, are encoded into OWL structures. Note that encoding classification labels requires conversion from a natural language to a formal language, whereas encoding classification nodes and edges requires only structural manipulation. In Section 4.1.1 we demonstrate how to solve the former problem, and in Section 4.1.2 we demonstrate how to solve the latter one. In Section 4.1.3 we detail how to encode classification documents and document-node links as class instances. Section 4.1.4 demonstrates how the resulting OWL ontology can be enriched with a set of axioms so that it can be better suited for automated reasoning. Finally, in Section 4.1.5, we discuss which subset of the OWL language is required in order to encode classifications into ontologies.

4.1.1 Label to Concept

In the conversion of natural language labels into a formal language we follow the approach presented in [19], which describes how these labels can be converted into a propositional concept language. The underlying idea of this approach is that the senses of words appearing in a label are converted into atomic concepts, whereas punctuation and syntactic relations between words in the label are converted into logical connectives (such as conjunction $\sqcap$ and disjunction $\sqcup$) and parentheses. As discussed in [25], the extension of these concepts is the set of documents about the objects or individuals referred to by the (lexically defined) concepts. As shown in the same article, this interpretation has some advantages: it provides the possibility of representing individuals as concepts, and not as instances (e.g., the extension of concept `George_Bush` is the set of documents about the president George Bush); and it provides the possibility of treating classification edges as the intersection of concepts. In our analysis of natural language labels, we exploit the natural language processing (NLP) pipeline presented in [51]. As opposed to standard approaches to NLP, this pipeline is adapted to be applied on web directory labels. In what follows, we present the main steps of the pipeline and we demonstrate how to complete some of them with the conversion to OWL.

1. Sense retrieval. In this step we retrieve the senses of each word in the label from the WordNet lexical database [38]. Apart from this, we identify words which are not found in WordNet.

2. Sense disambiguation. In this step we leave only one sense per ambiguous word following the word sense disambiguation algorithm presented in [51]. The algorithm exploits the structure of the classification, WordNet relations such as hypernymy, and the most frequent sense heuristic to disambiguate the meaning.

3. Building atomic concepts. In this step we convert the disambiguated senses as well as the words which are not found in WordNet into atomic concepts and encode them as OWL classes. Following the approach described in [48], we define the URI scheme to uniquely identify OWL classes generated from WordNet senses as follows:

`Synset- + lexical_form_of_the_word- + POS- + synset_number`

where `synset_number` is the number of the synset[1] to which the sense belongs in WordNet, and `lexical_form_of_the_word` is the lemma of the *first* word in the given synset. This enables us to represent synonymous words as one OWL class and not as multiple classes with equivalence relations defined between them.

For example, the URI for the atomic concept `java` which is generated from the sense *coffee* of the noun java is: `Synset-Coffee-Noun-41492`. An OWL class for this atomic concept is defined as follows:

<owl:Class rdf:ID="Synset-Coffee-Noun-41492"/>

We form URIs for the words which are not found in WordNet as their

[1]In WordNet, a *synset* is a set of one or more synonymous words which is assigned a unique numeric identifier, a gloss, and other metadata [38].

literal representation in the label. For example, word *xyz* is encoded as an OWL class with URI "xyz". This allows us to encode unknown words with the same spelling as one OWL class. Note that the encoding of words into concepts is done in such a way as to build a minimal set of concepts. The main reason for this choice is efficiency.

4. Building complex concepts. In this step we build complex concepts from atomic concepts following the approach discussed in [19]. For instance, a label composed of a sequence of adjectives followed by a noun group is converted into the logical conjunction ($\sqcap$) of the concepts corresponding to the adjectives and to the nouns. In this way, prepositions like "of" and "in" are converted into the logical conjunction, coordinating conjunctions "and" and "or" are converted into the logical disjunction ($\sqcup$), and so on.

We convert complex concepts generated from the labels into classes in OWL. We define the following URI schema to uniquely identify these OWL classes:

`Label- + node_label- + node_number`

where `node_label` is the label of the node without spaces, and where each word starts with a capitalized letter. For example, the URI for the label "Society and Law Culture" of node 2 of the classification given in Figure 3.1 is: `Label-SocietyAndLawCulture-2`. The OWL class for this label is the following:

```
<owl:Class rdf:ID="Label-SocietyAndLawCulture-2">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="Synset-Society-Noun-318"/>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:ID="Synset-Law-Noun-51793"/>
      <owl:Class rdf:ID="Synset-Culture-Noun-38542"/>
    </owl:intersectionOf>
  </owl:unionOf>
</owl:Class>
```

4.1.2 Concept at Label to Concept at Node

As discussed in Section 3.2, edges in a classification represent either a specification or a facet relation, which can be generalized to the following observation: the meaning of a child node consists of what the meaning of its label and the meaning of the parent node have in common. We formalize this observation in the notion of *concept of node* [20, 24, 21, 23], which is defined below:

$$C_i = \begin{cases} l_i^F & \text{if } n_i \text{ is the root of } C \\ l_i^F \sqcap C_j & \text{if } n_i \text{ is not the root of } C\text{, where } n_j \text{ is the parent of } n_i \end{cases} \tag{4.1}$$

where C_i is the concept of node n_i and l_i^F is the concept of label of node n_i. Concepts at nodes are converted into classes in OWL. The URI schema used to uniquely identify OWL classes corresponding to nodes is defined below:

`Node- + node_label- + node_number`

For example, the URI for the root node labeled "Top" with id 1 is: `Node-Top-1`. An OWL class for this root node is built as shown below:

```
<owl:Class rdf:ID="Node-Top-1">
  <equivalentClass rdf:resource="#Label-Top-1"/>
</owl:Class>
```

The URI for node 16 labeled "Programming Language" is: `Node-ProgrammingLanguage-16` and its corresponding OWL class is built as shown below:

```
<owl:Class rdf:ID="Node-ProgrammingLanguage-16">
  <owl:intersectionOf rdf:parseType="Collection">
```

```
    <owl:Class rdf:ID="Label-ProgrammingLanguage-16"/>
    <owl:Class rdf:ID="Node-Computer-15"/>
  </owl:intersectionOf>
</owl:Class>
```

Note that classification edges are implicitly encoded in the definitions of OWL classes representing concepts at nodes. Since these classes are defined as the intersection of the concept at parent node and the concept at label of the child node, then the structure of the classification can be reconstructed by analyzing node class definitions.

4.1.3 Document to Instance

We convert a document into an instance of the OWL Thing class. We assume that each document has a URL and we use it to uniquely identify the corresponding instance in OWL. Moreover, if a document has a title and a description (as web directory documents normally have), then we encode them in rdfs:label and rdfs:comment properties accordingly. For example, a document with URL `http://java-source.net/`, with title "Java Open Source Software", and with description "A directory of open source software focused on Java" is encoded in OWL as shown below:

```
<owl:Thing rdf:about="#http://java-source.net/">
  <rdfs:label>Java Open Source Software</rdfs:label>
  <rdfs:comment>A directory of open source software focused on Java
  </rdfs:comment>
</owl:Thing>
```

We convert document-node links of a document by defining the rdf:type relation from the instance representing the document, to the class(es) representing the node(s) in which the document is classified. For instance, if the above mentioned document is classified in nodes 2 and 4 of the classification shown in Figure 3.1, then these document-node links are encoded as shown below:

```
<owl:Thing rdf:about="#http://www.laweasy.com">
  <rdf:type rdf:resource="#Node-SocietyAndLawCulture-2"/>
  <rdf:type rdf:resource="#Node-Law-4"/>
</owl:Thing>
```

4.1.4 Semantic Enrichment

Since OWL classes, which correspond to word senses, are mapped to synsets in WordNet, we can exploit the relations between synsets and relations between words within synsets in order to enrich the resulting OWL ontologies with additional relations between classes. The enrichment is based on the following two rules:

- **Rule 1:** In WordNet synsets are organized into hierarchies based, for example, on the hypernym (i.e., *is-a* or *is-kind-of*) relation [38]. For instance, the synset denoting "Java" (as "a simple platform-independent object-oriented programming language") has a hypernym synset denoting "programming language" (as "a language designed for programming computers"). If two OWL classes (`cl-1` and `cl-2`) correspond to two senses (`sen-1` and `sen-2`) belonging to two synsets (`syn-1` and `syn-2`), among which there is a hypernym relation defined in WordNet (e.g., `syn-2` is a hypernym for `syn-1`), then we define an `rdfs:subClassOf` relation between these two classes (i.e., `cl-1 rdfs:subClassOf cl-2`) as shown below:

  ```
  <owl:Class rdf:about="#Synset-Java-Noun-41493">
   <rdfs:subClassOf rdf:resource="#Synset-ProgrammingLanguage-Noun-45-219"/>
  ```

</owl:Class>

- **Rule 2:** Antonym relations in WordNet are defined among *words* within synsets (and not among synsets). We translate these relations into `owl:disjo`
`intWith` relations among classes corresponding to senses of the two antonym words. For instance, the antonym of the word “day” in the synset {day, daytime, daylight} is the word “night” in the synset {night, nighttime, dark}. The former synset is the third sense of the noun “day” and the latter synset is the first sense of the noun “night”. Classes, associated with these two senses, are declared to be disjoint as shown below:

  <owl:Class rdf:about=“#Synset-Day-Noun-12826”>

    <owl:disjointWith rdf:resource=“#Synset-Night-Noun-38819”/>

  </owl:Class>

The enrichment of classification OWL ontologies according to the two rules described above enables us to make these ontologies more suitable for reasoning as the underline axiom base grows.

4.1.5 OWL Sublanguage

OWL ontologies, generated from classifications, fall into the OWL Lite or OWL DL subset of OWL. There are two factors which require OWL DL:

- the logical disjunction that may appear after the conversion of natural language labels and which is converted into the `owl:unionOf` construct;

- disjoint axioms that may appear at the semantic enrichment step and which are converted into the `owl:disjointWith` construct.

Table 4.1: Statistics of the dataset.

Dataset	Nodes	Average Branching Factor	Average Subtree Depth	Tokens Per Label	Words with Senses in WordNet	Noun Senses	Adjective Senses
Countries[a]	245	6.26	3	1.07	261	256	5
Europe[b]	75	4.22	3	1.12	86	86	0
Asia[c]	76	4.24	3	1.18	89	88	1
Africa[d]	80	4.31	3	1.15	94	93	1

[a]http://dmoz.org/Regional/Countries/.
[b]http://dmoz.org/Regional/Europe/.
[c]http://dmoz.org/Regional/Asia/.
[d]http://dmoz.org/Regional/Africa/.

Both above mentioned constructs are forbidden in OWL Lite. Note that the conversion to OWL does not require the use of constructs of OWL Full which leaves us within a decidable subset of OWL.

4.2 Evaluation

To evaluate our approach, we selected four subtrees with the maximum depth of 3 from the DMoz web directory. In Table 4.1 we report statistical data of the datasets. There are 476 nodes in the selected subtrees, which have 548 tokens in total, out of which 527 tokens are found in WordNet (i.e., WordNet coverage is 96.17%). Out of the set of words found in WordNet, 223 (i.e., 42.31%) are ambiguous with the average polysemy of 3.36. In our experiments we used WordNet version 2.0.

4.2.1 Correctness

We evaluated the most critical step of the NLP pipeline, i.e., the word sense disambiguation (see Section 4.1.1) algorithm, whose performance results are reported in Table 4.2. The accuracy of this step largely affects the

Table 4.2: Accuracy of the word sense disambiguation algorithm.

Dataset	Ambiguous Tokens	Disambiguation Accuracy(%)
Countries	92	76.54
Europe	38	77.01
Asia	47	80.89
Africa	46	79.13

Table 4.3: Statistics of the generated OWL ontologies.

Ontology	Nodes	Sense Classes	Label Classes	Node Classes	Class Axioms	Individual Axioms	intersectionOf Constructs	unionOf Constructs
Countries	245	261	245	245	873	0	265	4
Europe	75	86	75	75	155	183	76	10
Asia	76	89	76	76	203	125	80	9
Africa	80	94	80	80	212	253	84	9

correctness of the results of reasoning on these OWL ontologies, as shown in Section 4.3.5.

4.2.2 OWL Sublanguage

In Table 4.3 we report statistical data for the generated OWL ontologies. In Table 4.4 we provide details on the kind and number of axioms before and after semantic enrichment.

Table 4.4: Axioms before and after semantic enrichment.

Ontology	Equivalent Class Axioms		SubClass Axioms		Disjoint Class Axioms		Individual Axioms	
	Before	After	Before	After	Before	After	Before	After
Countries	490	490	0	383	0	0	0	0
Europe	152	152	0	3	0	0	183	183
Asia	152	152	0	51	0	0	125	125
Africa	160	160	0	52	0	0	253	253

It is to be noted that most of the constructs in the generated ontologies are valid in OWL Lite. There are only a few `owl:unionOf` constructs that require the use of OWL DL for the representation of these ontologies.

4.3 Optimizing Classifications

In this section we provide some practical examples of reasoning on classification OWL ontologies. For instance, we demonstrate how they can be checked for consistency, how their structure can be rationalized, and how nodes with similar contents to a given node can be found.

4.3.1 Consistency

We used Protégé OWL Plugin [35] and its reasoning capabilities to detect logical inconsistencies within the classification OWL ontologies. We used the reasoning capabilities of both Pellet 1.5 and Fact++ OWL reasoners launched with Protégé. None of the reasoners reported that the classification OWL ontologies were inconsistent.

4.3.2 Rational Forms

Classifications may not be perfect. For this reason we may need to reconstruct a classification based on the "most specific subsumer" relation. Nodes get parents which most specifically describe them (nodes), and are more general. The new structure is called a *rational form* of a classification. The idea behind the rationalization of classifications is to build a classification which better corresponds to a taxonomic structure. The classification given in Figure 4.1(b) is a rational form of the classification given in Figure 4.1(a). Note that classification semantics does not change in the transition from classification to rational form of classification, as the

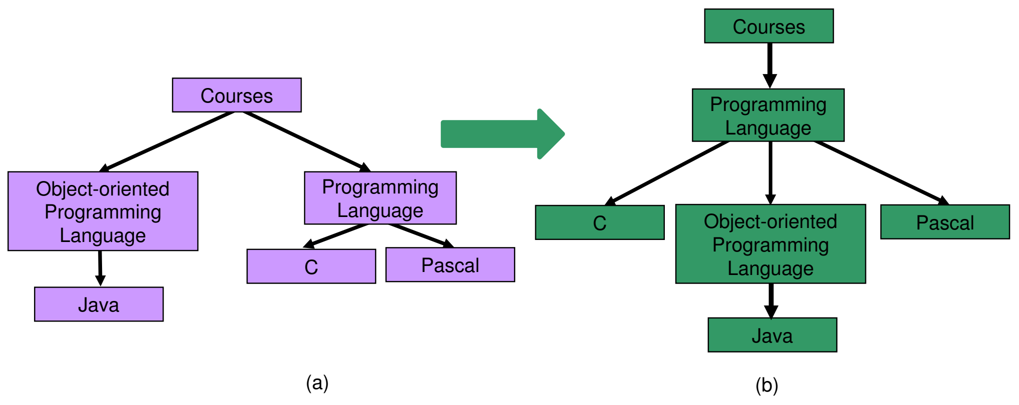

Figure 4.1: (a) Classification; (b) Rational form of the classification given in (a).

Table 4.5: Found relations within and across the four ontologies.

Ontology	Countries		Europe		Asia		Africa	
	$\sqsubseteq$	$\equiv$	$\sqsubseteq$	$\equiv$	$\sqsubseteq$	$\equiv$	$\sqsubseteq$	$\equiv$
Countries	383	490	386	642	392	642	387	650
Europe	386	642	3	152	51	304	52	312
Asia	392	642	51	304	51	152	100	312
Africa	387	650	52	312	100	312	52	160

set of concepts at nodes remains the same.

4.3.3 Minimizing Effort

In Table 4.5 we report the kind and number of found relations within and across the four ontologies. For example, the reasoner found an equivalent relation between node class */Regional/Countries/**Italy*** and node class */Regional/Europe/**Italy***. This is an example of how reasoning on classification OWL ontologies can help web directory editors find interrelated parts of the web directory and thereby improve its organizational structure without manual inspection. Note that no disjointness relations were found because we did not have disjoint axioms in the produced OWL ontologies.

4.3.4 Computing `See-Also` Links

Apart from the four ontologies, we experimented with another classification OWL ontology and we observed that the individuals asserted to the OWL class which corresponds to the classification node */Games and Activities/Kids and Teens/**Football*** are inferred as the individuals of the OWL class which corresponds to the classification node */Sports Athletics Funs/Youth and High School/**Soccer***, and vice versa. This kind of reasoning can be used for finding similar documents populated in different nodes, which will help in building `see-also` links.

4.3.5 Errors

Apart from the correct relations, we also found some incorrect ones. For example, the reasoner found an erroneous more specific relation between node class */Regional/Europe/**Georgia*** and node class */Regional/Countries/**United States***. As discussed earlier, this problem is caused by the lack of accuracy of the word sense disambiguation algorithm. Evaluating the correctness and completeness characteristics of the computed set of relations between ontology classes is outside the scope of this thesis. Interested readers are referred to [20, 16] for a complete account.

4.4 Related Work

The current work is representative of the recent trend in the Semantic Web community towards the use of *lightweight semantics* (as opposed to expressive logic languages) and *lightweight ontologies* [25] (as opposed to full-fledged ontologies), the generation of which can potentially be supported by ordinary users which constitute the long tail of the Semantic Web. The trend has been formed through a number of scientific publications (e.g., see [49, 19, 47, 33]) and is currently supported by a number of R&D projects (e.g., MATURE[2], OpenKnowledge[3]) and systems (e.g., OntoWiki[4]). The current work contributes to this trend by proposing an approach in which classifications, which are often called (informal) lightweight ontologies [25] and whose most representative instantiations on the web are web directories, can be automatically converted into formal OWL ontologies ready to be embedded in Semantic Web applications.

There are a few contemporary lines of work which are close in spirit to our approach. For instance, in [49], the authors propose a method to convert thesauri to OWL ontologies in which they provide a detailed account of how elements of a thesaurus are converted into OWL structures. This approach is based on a manual analysis of thesauri, whereas our approach allows for a fully automatic conversion. Another approach, discussed in [47], comes from the Digital Library community and presents a conceptual structure and transition procedure to support the shift from a traditional knowledge organization system (KOS) and, particularly, a thesaurus, towards a full-fledged and semantically rich KOS. While this approach provides an in-depth analysis of the shortcomings of the traditional KOSs and of the benefits of semantic KOSs, as well as a set of rules for converting thesaurus elements into ontology constructs, it lacks a specification of how a KOS can be converted into an ontology language such as OWL. This ultimate conversion step that has been discussed in detail in this chapter.

The approach described in [32] allows us to convert a hierarchical classification into an OWL ontology by deriving OWL classes from classification labels and by arranging these classes into a hierarchy (based on the rdfs:subClassOf relation) following the classification structure. The ap-

[2]MATURE, Integrated Project (IP), FP7-216356, see http://mature-ip.eu.
[3]OpenKnowledge, STREP, FP6-27253, see http://www.openk.org/
[4]OntoWiki, see http://ontowiki.net/Projects/OntoWiki.

proach is based on some application-dependent assumptions such as one label represents one atomic concept, and that relations between labels can be defined as `sub-class-of` relations in some particular context (e.g., concept “ice” is more specific than concept “non-alcoholic beverages” when considered in the context of procurement). These assumptions do not hold in a general case and are not made in our approach. Apart from this, our approach differs from [49, 47, 32] by being generic. It is therefore suitable for the automatic conversion in OWL of any knowledge representation structure whose core can be represented in the form of a classification, as defined in this chapter.

4.5 Conclusions

In this chapter we have presented a fully automated approach to converting generic classification schemes into OWL ontologies. The proposed approach allows us to leverage classifications, which are the interfaces to knowledge for humans, and ontologies, which are the interfaces to knowledge for machines on the Semantic Web. Furthermore, as shown above, our approach provides an immediate advantage by enabling the user to build better classifications more suited for reasoning. Potentially, the approach allows for the cost-free, seamless integration of a vast amount of classification structures on the web and in personal repositories into the Semantic Web infrastructure, thereby reducing the problem of the lack of semantically rich data. The initial experimental results, reported in this chapter, demonstrate that reasoning on classification OWL ontologies can be used for building practical Semantic Web applications.

Chapter 5

C-XML

In the previous chapter we described how to convert classifications into ontologies. In this chapter we provide the specification of a language named C-XML (Contextualized Markup Language), which can be used for representing faceted lightweight ontologies. We provide the abstract syntax of C-XML and then demonstrate a mapping to XML.

The chapter is organized as follows. In section 5.2.1 we introduce C-XML. In section 5.2 we outline the abstract syntax of C-XML. Section 5.2.1 describes a mapping between C-XML and XML.

5.1 Introduction

C-XML is an acronym for **C**onte**-**Xtualized **M**arkup **L**anguage. It is an XML-based language can be used to represent a set of CVs (controlled vocabulary), a set of Users and their classifications, a set of ETypes (entity types), and a set of AttributeDefs (attribute definitions). It specifies a basic format that enables Classifications, along with the CVs related to them, the Mdocs (meta documents) classified into them, and the Attributes related to the real world documents classified into them, to be exchanged between classification management systems. Although there are many languages (e.g., XML, XML-schema, KIF, CycL, OWL, Ontolingua, DAML-OIL, RDF, and RDF-schema) in the arena of ontology representation languages, none of them entirely suit our goal of presenting CVs, Users, contextual Classifications, Mdocs, Attributes, AttributeDefs, and ETypes. It is for this reason that we realized the need for a new language, and subsequently designed our own language, C-XML, for representing these items. C-XML is also suitable for representing faceted lightweight ontologies. This chapter provides a complete specification of C-XML, by defining its abstract syntax and by demonstrating a mapping from its abstract syntax to the XML syntax.

5.2 Abstract Syntax

5.2.1 Introduction

C-XML is intended to represent the content of CV (controlled vocabulary), Classification, Mdoc (meta document), AttributeDef (attribute definition), and EType (entity type). The Server, and every element included in the Server in the hierarchy of Figure 5.1, are called "object" in C-XML. A CV object consists of Word, Synset, Concept, CChildOf (concept child-of), and CCLink (concept-concept link) objects. As language comes before theory (i.e., classification), and we read from left to right, the CV object is placed on the left. A Classification object consists of Node, NChildOf (node child-of), NNLink (node-node link), and Mdoc objects. An EType object consists of EALink (entity-attribute link) and Service objects.

A C-XML document can be complete or partial. It is complete when all the objects related to the Server and all their attributes are present in the C-XML file. There are two ways in which a C-XML document can be partial:

- Intentional partiality: some attributes of an object are omitted from

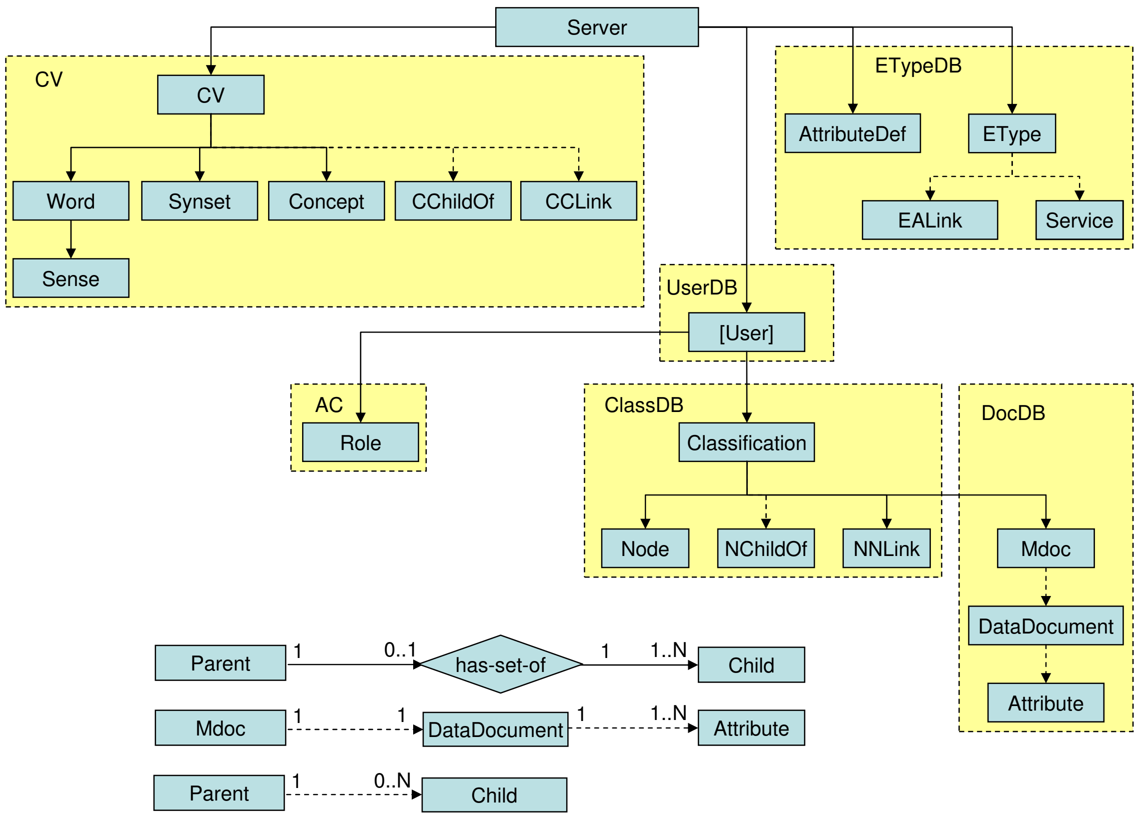

Figure 5.1: C-XML objects hierarchy.

the C-XML document.

- Extensional partiality: some objects are omitted from the C-XML document.

In order to be a valid document, a complete or partial C-XML file must comply with the following rules:

- *Single entry*: any object (identified by its URL) has only a single appearance in the file.

- *Presence*: Server object must appear in a file.

Note that there are two kinds of links in the hierarchy of Figure 5.1. Links represented as solid lines are objects and they have a URL. Links represented as dashed lines are not objects and they do not have a URL. In the legend, has-set-of object represents a link between an object which is connected to the arrowed end of the solid line, called child object, and another object which is connected to the non-arrowed end of the same solid line, called parent object. URLs for the objects connected by solid lines are formed by using the rules defined in this chapter. The URL formation rules are as follows:

- Rule 1: The Server object URL is absolute.
- Rule 2: A has-set-of object URL is formed by taking the corresponding parent object URL, appending the path separator ("/"), and the child object class identifier.
- Rule 3: A child object URL is formed by taking the corresponding child set object URL, appending the path separator ("/"), and the child object URL identifier (e.g., id, label).

As the links represented by dashed lines do not have URLs, objects connected to the arrowed end of the dashed lines do not follow the URL formation rules given above. However, the URL for the DataDocument (data document) object is an absolute URL, which represents a real world document. By following the URL formation rules given above we have instantiated the URL attribute of the objects for the leftmost branch of the tree rooted at Server. Instantiations are as follows:

```
<server url="http://kdtest.science.unitn.it:8180/SWebB">
 <vocabs url="http://kdtest.science.unitn.it:8180/SWebB/vocabs">
  <cv url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
           geography">
   <words url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
               geography/words">
```

```
        <word url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    geography/words/cartography">
         <senses url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                        geography/words/cartography/senses">
          <sense url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                      geography/words/cartography/senses/noun-85992"/>
         </senses>
        </word>
      </words>
    </cv>
   </vocabs>
 </server>
```

anyDataType: Represents any one data type described above.
Note that user-defined types are prefixed with "any", but that anyURI is not a defined type. User-defined data types are specified by means of BNF. Terminals are bold, non-terminals are normal typeface, the first letter of data types is lowercase, and rest is mixed case. For iteration we use {} with * operator (zero or more), + operator (one or more), or ? operator (zero or one). The format of anyDataType is provided below:

anyDataType ::= **int** | **nonNegativeInteger** | **string** | **dataTime** | **boolean** | **anyURI** | **token**

anyConcept: Represents the concept of some attributes
The format of anyConcept is provided below:

anyConcept ::= AtomicConcept | ComplexConcept
AtomicConcept ::= Token"["Synset?"]"
Token ::= **token**
Synset ::= 'c' OffSet {',''c' OffSet}*
OffSet ::= **nonNegativeInteger**

where character constant 'c' means Concept and Offset represents a Sense number in the CV. Note that there is no space before and after the comma, "[", and "]" in BNF definitions come above.

ComplexConcept ::= {"("}? AtomicConcept {("&" | "|") AtomicConcept {")"}?}+

The abstract syntax is specified by means of BNF. Terminals are bold, non-terminals are not bold. Terminals are either datatypes or string literals and non-terminals are either objects or attributes. Objects are in italic typeface, the first letter is capitalized, and the rest is mixed case. The first letter of attributes is lowercase and the rest is mixed case. Moreover, both the objects and attributes are written as tokens.

5.2.2 Server

From the C-XML point of view, the Server is the root object. It contains a url, zero or one Vocabs, zero or one Users or Classifications, zero or one AttributeDefs, and zero or one ETypes.

```
Server        ::=   url            [URL]
                    {Vocabs}?
                    {Users | Classifications}?
                    {AttributeDefs}?
                    {ETypes}?
url           ::=   anyURI
```

5.2.3 Vocabs

In C-XML a Vocabs contains a url and a group of one or more CVs.

```
Vocabs        ::=   url            [URL]
                    {CV}+
```

5.2.4 CV

In C-XML a CV contains a url, zero or one Words, zero or one Synsets, zero or one Concepts, a set of CChildOfs, and a set of CCLinks.

CV	::=	url	[URL]
		{*Words*}?	
		{*Synsets*}?	
		{*Concepts*}?	
		{*CChildOf*}*	
		{*CCLink*}*	

Words

In C-XML a Words contains a url and a group of one or more Words.

Words	::=	url	[URL]
		{*Word*}+	

Word

In C-XML a Word contains a url, a lemma, a set of derived_forms, an optional provenance, a timestamp, and zero or one Senses.

Word	::=	url	[URL]
		lemma	[Basic form]
		{derived_form}*	[Derived form]
		[provenance]	[Source]
		timestamp	[Last time when a Word was changed]
		{*Senses*}?	
lemma	::=	**string**	
derived_form	::=	**string**	
provenance	::=	**string**	

timestamp	::=	**dateTime**	

Senses

In C-XML a Senses contains a url and a group of one or more Senses.

Senses	::=	url	[URL]
		{*Sense*}+	

Sense

In C-XML a Sense contains a syn_url, an optional cased_lemma, and an optional rank.

Sense	::=	syn_url	[URL of the Synset]
		[cased_lemma]	[Cased version of the corresponding word]
		[rank]	[Frequency of use]
syn_url	::=	**anyURI**	
cased_lemma	::=	**string**	
rank	::=	**int**	

Synsets

In C-XML a Synsets contains a url and a group of one or more Synsets.

Synsets	::=	url	[URL]
		{*Synset*}+	

Synset

In C-XML a Synset contains a url, a pos, an optional gloss, an optional provenance, an optional c_url, and a timestamp.

Synset	::=	url	[URL]
		pos	[Part-of-speech]

		[gloss]	[Explanation]
		[provenance]	[Source]
		[c_url]	[Concept URL which corresponds to a Synset]
		timestamp	[Last time a Synset was changed]
pos	::=	**string**	
gloss	::=	**string**	
c_url	::=	**anyURI**	

Concepts

In C-XML a Concepts contains a url and a group of one or more Concepts.

Concepts	::=	url	[URL]
		{*Concept*}+	

Concept

In C-XML a Concept contains a url, a label, an optional definition, an optional provenance, a syn_url, and a timestamp.

Concept	::=	url	[URL]
		label	[Descriptive identifier]
		[definition]	[Definition]
		[provenance]	[Source]
		syn_url	[Synset URL]
		timestamp	[Last time a Concept was changed]
label	::=	**string**	
definition	::=	**string**	

CChildOf

In C-XML a CChildOf contains a source_c_url and a target_c_url.

CChildOf	::=	source_c_url	[Source Concept URL]
		target_c_url	[Target Concept URL]

CCLink

In C-XML a CCLink contains a url, a source_c_url, a target_c_url, an cCRelation, an optional gloss, and an optional provenance.

CCLink	::=	url	[URL]
		source_c_url	[Source Concept URL]
		target_c_url	[Target Concept URL]
		cCRelation	[Concept-Concept Relation which represents a relation between two concepts in a CV]
		[gloss]	[Explanation]
		[provenance]	[Source]
ccrelation	::=	“=”	
		|“!”	

A description of the cCRelation attribute values is given below:

- = : the target concept is semantically equivalent to the source concept.
- ! : the target concept is semantically disjoint with the source concept.

5.2.5 Users

In C-XML a Users contains a url and a group of one or more Users.

Users	::=	url	[URL]
		{*User*}+	

5.2.6 User

In C-XML a User contains a url, a name, and zero or one Classifications.

User	::=	url	[URL]
		name	[Descriptive identifer]
		$\{$*Classifications*$\}?$	
name	::=	**string**	

5.2.7 Classifications

In C-XML a Classifications contains a url and a group of one or more Classifications.

Classifications	::=	url	[URL]
		$\{$*Classification*$\}+$	

5.2.8 Classification

In C-XML a Classification contains a url, a name, an optional description, an optional gloss, a timestamp, an optional root_n_url, zero or one Nodes, a set of NchildOfs, zero or one NNLinks, and zero or one Mdocs.

Classification	::=	url	[URL]
		name	[Descriptive identifer]
		[description]	[Description]
		[gloss]	[Explanation]
		timestamp	[Last time a Classification was changed]
		[root_n_url]	[Root Node URL]
		Nodes	
		NchildOfs	
		NNLinks	
		Mdocs	

description	::=	**string**
timestamp	::=	**dateTime**
root_n_url	::=	**anyURI**

Nodes

In C-XML a Nodes contains a url and a group of one or more Nodes.

Nodes	::=	url	[URL]
		{*Node*}+	

Node

In C-XML a Node contains a url, a label, an optional cocept_at_label, an optional concept_at_node, two optional boolean attributes - is_clabel_aligned and is_cnode_aligned, an optional cn_timestamp, an optional gloss, a timestamp, and an optional default_mdoc_url.

Node	::=	url	[URL]
		label	[Descriptive identifier]
		[concept_at_label]	[Concept at label as defined in [3]]
		[concept_at_node]	[Concept at node as defined in [3]]
		[is_clabel_aligned]	[Is concept at label aligned]
		[is_cnode_aligned]	[Is concept at node aligned]
		[cn_timestamp]	[Last time concept at node was aligned]
		[gloss]	[Explanation]
		timestamp	[Last time a Node was changed]
		[default_mdoc_url]	[URL of an Mdoc which is default to a Node]

concept_at_label	::=	**anyConcept**

concept_at_node	::=	**anyConcept**
is_clabel_aligned	::=	**boolean**
is_cnode_aligned	::=	**boolean**
cn_timestamp	::=	**dateTime**
default_mdoc_url	::=	**anyURI**
is_clabel_aligned	::=	**boolean**
is_cnode_aligned	::=	**boolean**
cn_timestamp	::=	**dateTime**
default_mdoc_url	::=	**anyURI**

NChildOf

In C-XML an NChildOf contains a source_n_url and a target_n_url.

NChildOf	::=	source_n_url	[Source Node URL]
		target_n_url	[Target Node URL]
source_n_url	::=	**anyURI**	
target_n_url	::=	**anyURI**	

NNLinks

In C-XML an NNLinks contains a url and a group of one or more NNLinks.

NNLinks	::=	url	[URL]
		NNLink+	

NNLink

In C-XML an NNLink contains a url, a source_n_url, a target_n_url, an optional kind, an optional nNRelation, a target_cn_timesta-mp, an optional gloss, and a timestamp.

NNLink	::=	url	[URL]
		source_n_url	[Source Node URL]
		target_n_url	[Target Node URL]

		[kind]	[Type]
		[nNRelation]	[Node-Node Relation]
		target_cn_timestamp	[Last time concept at node of a target Node was aligned]
		[gloss]	[Explanation]
		timestamp	[Last time an NNLink was changed]
kind	::=	"**AdoptedChild**"	
		|"**Clink**"	
		|"**SeeAlso**"	
nNRelation	::=	"**>**"	
		|"**<**"	
		|"**=**"	
		|"**!**"	
		|"**Idk**"	
		|"**?**"	
target_cn_timestamp	::=	**dateTime**	

A detailed description of the attributes "kind" and "nNRelation" is given below:

- **Syntactic**: Means that the NNLink encodes some kind of relation that holds from the source node to the target node, and that this kind of relation cannot be given any formal semantics. All links which are of the syntactic kind have "?" in their nNRleation. For example "SeeAlso" is a syntactic link, while link kind is "SeeAlso", nNRelation is "?".

- **Semantic**: Means that the NNLink represents a semantic relation between the related source and the target nodes. In the context of C-XML, links of this kind are the following:
  - **AdoptedChild**: Represents a link where the target node concept is more specific than the source node concept. The source node concept is like the concept of a parent, even though it is not a parent.
  - **Clink**: Clinks stand for Context Links as defined in [4].

  While link kind is “Clink”, nNRelation is one of the following:
  - >: the target node concept is more general than the source node concept.
  - <: the target node concept is more specific than the source node concept.
  - =: the target node concept is semantically equivalent to the source node concept.
  - !: the target node concept is semantically disjoint with the source node concept.
  - **Idk**: the relation between the source node concept and the target node concept is unknown.

Mdocs

In C-XML an Mdocs contains a url and a group of one or more Mdocs.

Mdocs	::=	url	[URL]
		Mdoc+	

Mdoc

In C-XML an Mdoc contains a url, an optional gloss, a timestamp, and a DataDocument.

Mdoc	::=	url	[URL]
		[gloss]	[Explanation]
		timestamp	[Last time an Mdoc was changed]

DataDocument

There are many different kinds of documents such as scientific publications, multimedia, e-mail, web pages, (structured) texts, and so on. Different kinds of documents are often described using different sets of metadata. For example, a scientific publication requires an author and a conference, an audio file requires a genre and a singer, and an e-mail file requires a sender and a receiver. The current version of C-XML supports any number of attributes.
In C-XML a DataDocument contains a url and a group of one or more Attributes.

DataDocument	::=	url	[URL]
		Attribute+	

Attribute

In C-XML an Attribute contains an a_url and a set of values.

Attribute	::=	a_url	[Attribute definition URL]
		{value}*	[Value]
a_url	::=	**anyURI**	
value	::=	**anyDataType**	

5.2.9 AttributeDefs

In C-XML an AttributeDefs contains a url and a group of one or more AttributeDefs.

AttributeDefs	::=	url	[URL]
		AttributeDef+	

5.2.10 AttributeDef

An AttributeDef contains an a_url, a name, a datatype, and an optional description.

AttributeDef	::=	url	[URL]
		name	[Descriptive identifier]
		datatype	[Datatype of an Attribute]
		[description]	[Description]
datatype	::=	**anyDataType**	

5.2.11 Etypes

In C-XML an Etypes contains a url and a group of one or more Etypes.

Etypes	::=	url	[URL]
		Etype+	

5.2.12 Etype

In C-XML an Etype contains a url, a name, a set of EALinks, and a set of Services.

Etype	::=	url	[URL]
		name	[Descriptive identifier]
		EALink*	
		Service*	

EALink

In C-XML an EALink contains an a_url and a boolean attribute is_sma.

EALink	::=	a_url	[Attribute definition URL]
		is_sma	[Is Strictly Mandatory Attribute]
is_sma	::=	boolean	

Service

In C-XML a Service contains a url.

Service	::=	url	[URL]

Note that later on this url will represent the URL of a web service attached to the etype. This url is empty for the time being.

5.3 Mapping to XML

This section demonstrates a mapping from the C-XML abstract syntax given in section 5.2, to the XML syntax. Some general rules of mapping are provided below:

A) C-XML objects are mapped to XML elements.

B) properties of C-XML objects are mapped to attributes of XML.

The mapping table shows mappings between C-XML abstract syntax and XML where: objects are prefixed with “O:”; elements are prefixed with “E:”; properties are prefixed with “P:”; and attributes are prefixed with “A:”. The left column of the table represents either object or property of object, the center column represents its XML equivalent, and the right column represents its XML tag representation.

Table 5.1: Mapping to XML.

C-XML	**XML syntax**	**enclosed in XML**

abstract syntax		**tags**
O:Server	E:server	<server>...</server>
O:Vocabs	E:vocabs	<vocabs>...</vocabs>
O:Vocabs	E:vocabs	<vocabs>...</vocabs>
O:CV	E:cv	<cv>...</cv>
O:Words	E:words	<words>...</words>
O:Word	E:word	<word>...</word>
O:Synsets	E:synsets	<synsets>...</synsets>
O:Synset	E:synset	<synset>...</synset>
O:Concepts	E:concepts	<concepts>...</concepts>
O:CChildOf	E:cchildof	<cchildof>...</cchildof>
O:CCLink	E:cclink	<cclink>...</cclink>
O:Senses	E:senses	<senses>...</senses>
O:Sense	E:sense	<sense>...</sense>
O:Users	E:users	<users>...</users>
O:User	E:user	<user>...</user>
O:Classifications	E:classifications	<classifications>...</classifications>
O:Classification	E:classification	<classification>...</classification>
O:Nodes	E:nodes	<nodes>...</nodes>
O:Node	E:node	<node>...</node>
O:NChildOf	E:nchildof	<nchildof>...</nchildof>
O:NNLinks	E:nnlinks	<nnlinks>...</nnlinks>
O:NNLink	E:nnlink	<nnlink>...</nnlink>
O:Mdocs	E:mdocs	<mdocs>...</mdocs>
O:Mdoc	E:mdoc	<mdoc>...</mdoc>
O:AttributeDefs	E:attributedefs	<attributedefs>...</attributedefs>
O:AttributeDef	E:attributedef	<attributedef>...</attributedef>

O:Attribute	E:attribute	<attribute>...</attribute>
O:ETypes	E:etypes	<etypes>...</etypes>
O:EType	E:etype	<etype>...</etype>
O:EALink	E:ealink	<ealink>...</ealink>
O:Service	E:service	<service>...</service>
O:{CChildOf}*	E:cchildofs	<cchildofs>...</cchildofs>
O:{CCLink}*	E:cclinks	<cclinks>...</cclinks>
O:{NChildOf}*	E:nchildofs	<nchildofs>...</nchildofs>
O:{Attribute}+	E:attributes	<attributes>...</attributes>
O:{Service}*	E:services	<services>...</services>
O:{EALink}*	E:ealinks	<ealinks>...</ealinks>

P:url	A:url	<server url=‘anyURI’>, <vocabs url=‘anyURI’>, <cv url=‘anyURI’>, <words url=‘anyURI’>, <word url=‘anyURI’>, <senses url=‘anyURI’>, <synsets url=‘anyURI’>, <synset url=‘anyURI’>, <concepts url=‘anyURI’>, <concept url=‘anyURI’>, <cclink url=‘anyURI’>, <users url=‘anyURI’>, <user url=‘anyURI’>, <classifications url=‘anyURI’>, <classification url=‘anyURI’>, <nodes url=‘anyURI’>, <node url=‘anyURI’>, <nnlinks url=‘anyURI’>, <nnlink url=‘anyURI’>, <mdocs url=‘anyURI’>, <mdoc url=‘anyURI’>, <attributedefs url=‘anyURI’>, <attributedef url=‘anyURI’>, <etypes url=‘anyURI’> and <etype url=‘anyURI’>
P:lemma	A:lemma	<word lemma=‘string’>
P:derived_form	A:derived_form	<word derived_form=‘string’>

P:provenance	A:provenance	<word provenance='string'>, <synset provenance='string'>, <concept provenance='string'> and <cclink provenance='string'>
P:timestamp	A:timestamp	<word timestamp='dateTime'>, <synset timestamp='dateTime'>, <concept timestamp='dateTime'>, <classification timestamp='dateTime'>, <node timestamp='dateTime'>, <nnlink timestamp='dateTime'> and <mdoc timestamp='dateTime'>
P:syn_url	A:syn_url	<sense syn_url='string'> and <concept syn_url='string'>
P:cased_lemma	A:cased_lemma	<sense cased_lemma='string'>
P:rank	A:rank	<sense rank='int'>
P:pos	A:pos	<synset pos='string'>
P:gloss	A:gloss	<synset gloss='string'>, <cclink gloss='string'>, <classification gloss='string'>, <node gloss='string'>, <nnlink gloss='string'> and <mdoc gloss='string'>
P:c_url	A:c_url	<synset c_url='string'>
P:label	A:label	<concept label='string'> and <node label='string'>
P:definition	A:definition	<concept definition='string'>

P:source_c_url	A:source_c_url	<cchildof source_c_url='anyURI'> and <cclink source_c_url='anyURI'>
P:target_c_url	A:target_c_url	<cchildof target_c_url='anyURI'> and <cclink target_c_url='anyURI'>
P:cCRelation	A:ccrelation	<cclink ccrelation='string'>
P:name	A:name	<user name='string'>, <classification name='string'>, <attributedef name='string'> and <etype name='string'>
P:description	A:description	<classification description='string'> and <attributedef description='string'>
P:root_n_url	A:root_n_url	<classification root_n_url='anyURI'>
P:concept_at_la-bel	A:concept_at_l-abel	<node con-cept_at_label='anyConcept'>
P:concept_at_n-ode	A:concept_at_-node	<node con-cept_at_node='anyConcept'>
P:is_clabel_alig-ned	A:is_clabel_ali-gned	<node is_clabel_aligned='boolean'>
P:is_cnode_align-ed	A:is_cnode_alig-ned	<node is_cnode_aligned='boolean'>
P:cn_timestamp	A:cn_timestamp	<node cn_timestamp='dateTime'>
P:default_mdoc-_url	A:default_mdo-c_url	<node default_mdoc_url='anyURI'>
P:source_n_url	A:source_n_url	<nchildof source_n_url='anyURI'> and <nnlink source_n_url='any-URI'>

P:target_n_url	A:target_n_url	<nchildof target_n_url='anyURI'> and <nnlink target_n_url='anyURI'>
P:kind	A:kind	<nnlink kind='string'>
P:nNRelation	A:nnrelation	<nnlink nnrelation='string'>
P:target_cn_timestamp	A:target_cn_timestamp	<nnlink target_cn_timestamp='dateTime'>
P:a_url	A:a_url	<attribute a_url='anyURI'> and <ealink a_url='anyURI'>
P:value	A:value	<attribute value='string'>
P:datatype	A:datatype	<attributedef datatype='anyDataType'>
P:is_sma	A:is_sma	<ealink is_sma='boolean'>

Each XML document has an optional prolog which has meta-information about the document followed by the root element (no correspondence with abstract syntax).

```
<?xml version= “1.0” encoding = “UTF-8” ?>
  <server>
  </server>
```

Note that the C-XML object Server is encoded as the root element “server” in XML.

5.3.1 Server

Given the Server url=“http://kdtest.science.unitn.it:8180/SWebB”, the abstract syntax of the Server given in subsection 5.2.2 can be mapped either as

```
  <server url=“http://kdtest.science.unitn.it:8180/SWebB”>
    <vocabs>
    </vocabs>
    <users>
    </users>
    <attributedefs>
    </attributedefs>
    <etypes>
    </etypes>
  </server>
```

or as

```
  <server url=“http://kdtest.science.unitn.it:8180/SWebB”>
    <vocabs>
    </vocabs>
    <classifications>
    </classifications>
  <attributedefs>
  </attributedefs>
  <etypes>
  </etypes>
</server>
```

Where:

- **vocabs**: contains a group of one or more CVs.
- **users**: contains a group of one or more Users.
- **classifications**: contains a group of one or more Classifications.
- **attributedefs**: contains a group of one or more AttributeDefs.
- **etypes**: contains a group of one or more ETypes.

5.3.2 Vocabs

Given the Vocabs url="http://kdtest.science.unitn.it:8180/SWebB/vocabs", the abstract syntax of Vocabs given in subsection 5.2.3 can be mapped to the XML as shown below:

```
<vocabs url="http://kdtest.science.unitn.it:8180/SWebB/vocabs">
  <cv>
  </cv>
     ...
  <cv>
  </cv>
</vocabs>
```

Where:
Each cv corresponds to a single classification;

5.3.3 CV

Given the abstract syntax of the object CV, described in subsection 5.2.4, which has the property url="http://kdtest.science.unitn.it:8180/SWebB/-vocabs/unitn", a mapping to XML is provided below:

```
<cv url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn">
  <words>
  </words>
  <synsets>
  </synsets>
  <concepts>
  </concepts>
  <cchildofs>
  </cchildofs>
  <cclinks>
  </cclinks>
</cv>
```

Where:

- **words**: contains a group of one or more Words.
- **concepts**: contains a group of one or more Concepts.
- **synsets**: contains a group of one or more Synsets.
- **cchildofs**: contains a set of CChildOfs.
- **cclinks**: contains a set of CCLinks.

Words

Given the abstract syntax of the object Words, described in subsection 5.2.4, which has the property url= "http://kdtest.science.unitn.it:8180/SWebB/-vocabs/unitn/words", a mapping to XML is provided below:

```
<words url="http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn/
            words">
  <word>
  </word>
     ...
  <word>
  </word>
</words>
```

Word

Given the abstract syntax of the object Word, described in subsection 5.2.4, which has properties

```
url          ="http://kdtest.science.unitn.it:8180/ SWebB/vocabs/
                unitn/words/schools",
lemma        ="school",
derived_form ="",
provenance   ="Feroz" and
timestamp    ="2008-07-10 2:18:02.89",
```

a mapping to XML is provided below:

```
<word
  url          ="http://kdtest.science.unitn.it:8180/ SWebB/vocabs/
                  unitn/words/schools"
  lemma        ="school"
  derived_form =""
  provenance   ="Feroz"
  timestamp    ="2008-07-10 2:18:02.89">
 <senses>
 </senses>
</word>
```

Where:

- **senses**: contains a group of one or more Senses of a Word.

Senses

Given the abstract syntax of the object Senses, described in subsection 5.2.4, which has the property url= “http://kdtest.science.unitn.it:8180/SWebB/-vocabs/unitn/words/schools/senses”, a mapping to XML is provided below:

```
<senses
  url          =“http://kdtest.science.unitn.it:8180/ SWebB/vocabs/
                    unitn/words/schools/senses”
 <sense>
 </sense>
     ...
 <sense>
 </sense>
</senses>
```

Sense

Given the abstract syntax of the object Sense, described in subsection 5.2.4, which has properties

```
  syn_url      =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                  synsets/34183”,
  cased_lemma =“” and
  rank         =“1”,
```

a mapping to XML is provided below:

```
<sense
  syn_url      =“http://kdtest.science.unitn.it:8180/ SWebB/vocabs/
```

```
                    synsets/34183"
  cased_lemma      =""
  rank             ="1"/>
```

Synsets

Given the abstract syntax of the object Synsets, described in subsection 5.2.4, which has the property url="http://kdtest.science.unitn.it:8180-/SWebB/vocabs/unitn/synsets", a mapping to XML is provided below:

```
 <synsets
   url        ="http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                unitn/synsets"
  <synset>
  </synset>
      ...
  <synset>
  </synset>
 </synsets>
```

Synset

Given the abstract syntax of the object Synset, described in subsection 5.2.4, which has properties

```
  url        ="http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn
               synsets/34183",
  pos        ="Noun",
  gloss      ="",
  provenance ="WordNet",
  c_url      ="" and
  timestamp  ="2008-07-10 2:18:03.89",
```

a mapping to XML is provided below:

```
<synset
 url        =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn
                  synsets/34183”
 pos        =“Noun”
 gloss      =“”
 provenance =“WordNet”
 c_url      =“”
 timestamp  =“2008-07-10 2:18:03.89”/>
```

Concepts

Given the abstract syntax of the object Concepts, described in subsection 5.2.4, which has the property url=“http://kdtest.science.unitn.it:8180-/SWebB/vocabs/unitn/concepts”, a mapping to XML is provided below:

```
<concepts url=“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                  unitn/concepts”>
 <concept>
 </concept>
     ...
 <concept>
 </concept>
</concepts>
```

Concept

Given the abstract syntax of the object Concept, described in subsection 5.2.4, which has properties

```
 url        =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                  unitn/concepts/1”,
 label      = “school”,
 definition =“”,
```

provenance = "WordNet",
syn_url = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn/synsets/34183" and
timestamp = "2008-07-10 2:18:03.89",

a mapping to XML is provided below:

```
<concept
  url        = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                unitn/concepts/1"
  label      = "school"
  definition = ""
  provenance = "WordNet"
  syn_url    = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                unitn/synsets/34183"
  timestamp  = "2008-07-10 2:18:03.89"/>
```

CChildOf

Given the abstract syntax of the object CChildOf, described in subsection 5.2.4, which has properties

source_c_url = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn/concepts/1", and
target_c_url = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/unitn/concepts/4",

a mapping to XML is provided below:

```
<cchildof
  source_c_url = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                  unitn/concepts/1"
  target_c_url = "http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                  unitn/concepts/4"/>
```

CCLink

Given the abstract syntax of the object CCLink, described in subsection 5.2.4, which has properties

```
  url           =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/cclinks/1”,
  source_c_url  =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/concepts/1”,
  target_c_url  =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/concepts/4”,
  cCRelation    =“=”,
  gloss         =“”, and
  provenance    =“WordNet”,
```

a mapping to XML is provided below:

```
 <cclink
  url           =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/cclinks/1”,
  source_c_url  =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/concepts/1”,
  target_c_url  =“http://kdtest.science.unitn.it:8180/SWebB/vocabs/
                    unitn/concepts/4”,
  cCRelation    =“=”,
  gloss         =“”, and
  provenance    =“WordNet”/>
```

5.3.4 Users

Given the abstract syntax of the object Users, described in subsection 5.2.5, which has the property url = “http://kdtest.science.unitn.it:8180/SWebB-/users”, a mapping to XML is provided below:

```
<users     url    =“http://kdtest.science.unitn.it:8180/SWebB/users”>
  <user>
  </user>
      ...
  <user>
  </user>
</users>
```

5.3.5 User

Given the abstract syntax of the object User, described in subsection 5.2.6, which has properties
url =“http://kdtest.science.unitn.it:8180/SWebB/users/1”, and
name =“Fausto Giunchiglia”,
a mapping to XML is provided below:

```
<user      url    =“http://kdtest.science.unitn.it:8180/SWebB/users/1”
           name  =“Fausto Giunchiglia”>
  <classification>
  </classification>
       ...
  <classificaiton>
  </classificaiton>
</user>
```

5.3.6 Classifications

Given the abstract syntax of the object Classifications, described in subsection 5.2.7, which has the property url=“http://kdtest.science.unitn.it:8180-/SWebB/users/1/classifications”, a mapping to XML is provided below:
<classifications url=“http://kdtest.science.unitn.it:8180/SWebB/users/1-

```
                    /classifications”>
  <classification>
  </classification>
        ...
  <classification>
  </classification>
</classifications>
```

5.3.7 Classification

Given the abstract syntax of the object Classification, described in subsection 5.2.8, which has properties

```
   url         =“http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1”,
   name        =“unitn”,
   description =“”,
   gloss       =“”,
   timestamp   =“2008-07-10 2:18:03.89” and
   root_n_url  =“http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1/nodes/1”,
```

a mapping to XML is provided below:

```
  <classification
   url         =“http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1”
   name        =“unitn”
   description =“”
   gloss       =“”
   timestamp   =“2008-07-10 2:18:03.89”
   root_n_url  =“http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1/nodes/1”>
  <nodes>
  </nodes>
  <nchildofs>
  </nchildofs>
  <nnlinks>
  </nnlinks>
  <mdocs>
  </mdocs>
</classification>
```

Where:

- **nodes**: contains a group of one or more Nodes of a Classification.
- **nchildofs**: contains a set of NChildOfs of a Classification.
- **nnlinks**: contains a group of one or more NNLinks of a Classification.
- **mdocs**: contains a group of one or more Mdocs of a Classification.

5.3.8 Nodes

Given the abstract syntax of the object Nodes, described in subsection 5.2.8, which has the property url="http://kdtest.science.unitn.it:8180/SWebB/users/1/classifications/1/nodes", a mapping to XML is provided below:

```
 <nodes
  url          ="http://kdtest.science.unitn.it:8180/SWebB/users/1/
                  classifications/1/nodes">
  <node>
  </node>
     ...
  <node>
 </node>
</nodes>
```

Node

Given the abstract syntax of the object Node, described in subsection 5.2.8, which has properties

```
url               =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                     /classifications/1/nodes/1”,
label             =“Doctoral Schools”,
concept_at_label  =“(Doctoral[c12993] & Schools[c34183])”,
concept_at_node   =“Doctoral[c12993] & Schools[c34183]”,
is_clabel_aligned =“true”,
is_cnode_aligned  =“true”,
cn_timestamp      =“2008-07-10 2:18:03.97”,
gloss             =“”,
timestamp         =“2008-07-10 2:18:03.90” and
default_mdoc_url=“”,
```

a mapping to XML is provided below:

```
<node
 url               =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                     /classifications/1/nodes/1”
 label             =“Doctoral Schools”
 concept_at_label  =“(Doctoral[c12993] & Schools[c34183])”
 concept_at_node   =“Doctoral[c12993] & Schools[c34183]”
 is_clabel_aligned =“true”
 is_cnode_aligned  =“true”
 cn_timestamp      =“2008-07-10 2:18:03.97”
 gloss             =“”
 timestamp         =“2008-07-10 2:18:03.90”
```

default_mdoc_url=“”/>

NChildOf

Given the abstract syntax of the object NChildOf, described in subsection 5.2.8, which has properties

```
    source_n_url    =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                        /classifications/1/nodes/1” and
    target_n_url    =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                        /classifications/1/nodes/2”,
```

a mapping to XML is shown below:

```
<nchildof
    source_n_url    =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                        /classifications/1/nodes/1”
    target_n_url    =“http://kdtest.science.unitn.it:8180/SWebB/users/1
                        /classifications/1/nodes/2”/>
```

NNLinks

Given the abstract syntax of the object NNLinks, described in subsection 5.2.8, which has the property url=“http://kdtest.science.unitn.it:8180-/SWebB/users/1/classifications/1/nnlinks”, a mapping to XML is shown below:

```
<nnlinks url=“http://kdtest.science.unitn.it:8180/SWebB/users/1/
                classifications/1/nnlinks”>
  <nnlink>
  </nnlink>
      ...
  <nnlink>
  </nnlink>
```

```
</nnlinks>
```

NNLink

Given the abstract syntax of the object NNLink, described in subsection 5.2.8, which has properties

```
  url                 =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/1/nnlinks/1”,
  source_n_url        =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/1/nodes/4”,
  target_n_url        =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/2/nodes/5”,
  kind                =“CLink”,
  nnrelation          =“>”,
  target_cn_timestamp=“2008-07-10 2:18:03.93”,
  gloss               =“” and
  timestamp           =“2008-07-10 2:18:03.91”,
```

a mapping to XML is shown below:

```
 <nnlink
  url                 =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/1/nnlinks/1”
  source_n_url        =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/1/nodes/4”
  target_n_url        =“http://kdtest.science.unitn.it:8180/SWebB/
                         users/1/classifications/2/nodes/5”
  kind                =“CLink”
  nnrelation          =“>”
  target_cn_timestamp=“2008-07-10 2:18:03.93”
  gloss               =“”
  timestamp           =“2008-07-10 2:18:03.91”/>
```

Mdocs

Given the abstract syntax of the object Mdocs, described in subsection 5.2.8, which has the property url="http://kdtest.science.unitn.it:8180/SWebB/users/1/classifications/1/mdocs", a mapping to XML is shown below:

```
 <mdocs url="http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1/mdocs">
  <mdoc>
  </mdoc>
      ...
  <mdoc>
  </mdoc>
 </mdocs>
```

Mdoc

Given the abstract syntax of the object Mdoc, described in subsection 5.2.8, which has properties

```
  url       ="http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1/mdocs/1",
  gloss     ="" and
  timestamp ="2008-07-10 2:18:03.95",
```

a mapping to XML is shown below:

```
 <mdoc
  url       ="http://kdtest.science.unitn.it:8180/SWebB/users/1/
                    classifications/1/mdocs/1"
  gloss     =""
  timestamp ="2008-07-10 2:18:03.95">
 <datadocument>
 </datadocument>
```

</mdoc>

DataDocument

Given the abstract syntax of the object DataDocument, described in subsection 5.2.8, which has the property url="http://www.disi.unitn.it", a mapping to XML is shown below:
<datadocument url ="http://www.disi.unitn.it">
 <attributes>
 </attributes>
 </datadocument>
Where:

- **attributes**: contains a group of one or more Attributes.

Attribute

Given the abstract syntax of the object Attribute, described in subsection 5.2.8, which has the property a_url="http://kdtest.science.unitn.it:8180/SWebB/attributedefs/author", a mapping to XML is shown below:
<attribute
 a_url="http://kdtest.science.unitn.it:8180/SWebB/attributedefs/author"
 value="Fausto Giunchiglia; Ilya Zaihreau; Feroz Farazi"/>
Note that value attribute contains multiple values separated by semicolon (;).

5.3.9 AttributeDefs

Given the abstract syntax of the object AttributeDefs, described in subsection 5.2.9, which has the property url="http://kdtest.science.unitn.it:8180-/SWebB/attributedefs", a mapping to XML is shown below:
 <attributedefs

```
  url      =“http://kdtest.science.unitn.it:8180/SWebB/attributedefs”>
 <attributedef/>
    ...
 <attributedef/>
</attributedefs>
```

5.3.10 AttributeDef

Given the abstract syntax of the object AttributeDef, described in subsection 5.2.10, which has properties

```
  url         =“http://kdtest.science.unitn.it:8180/SWebB/
                    attributedefs/author”,
  name        =“Author”,
  datatype    =“STRING” and
  description =“”,
```

a mapping to XML is shown below:

```
  <attributedef
  url         =“http://kdtest.science.unitn.it:8180/SWebB/
                    attributedefs/author”
  name        =“Author”
  datatype    =“STRING”
  description =“”/>
```

5.3.11 ETypes

Given the abstract syntax of the object ETypes, described in subsection 5.2.11, which has the property url=“http://kdtest.science.unitn.it:8180/SWebB/etypes”, a mapping to XML is shown below:

```
 <etypes
     url=“http://kdtest.science.unitn.it:8180/SWebB/etypes”>
```

```
  <etype/>
     ...
  <etype/>
 </etypes>
```

5.3.12 EType

Given the abstract syntax of the object EType, described in subsection 5.2.12, which has properties
url =“http://kdtest.science.unitn.it:8180/SWebB/etypes/PDF”, and
name =“PDF”,
a mapping to XML is shown below:

```
<etype
  url    =“http://kdtest.science.unitn.it:8180/SWebB/etypes/PDF”
  name   =“PDF”>
 <ealinks>
 </ealinks>
 <services>
 </services>
</etype>
```

EALink

Given the abstract syntax of the object EALink, described in subsection 5.2.12, which has the properties a_url =“http://kdtest.science.unitn.it-:8180/SWebB/attributedefs/author”, and is_sma =“true”, a mapping to XML is shown below:

```
<ealink
  a_url   =“http://kdtest.science.unitn.it:8180/SWebB/
                attributedefs/author”>
  is_sma  =“true”/>
```

Service

Given the abstract syntax of the object Service, described in subsection 5.2.12, which has the property url="later on this url will represent the URL of a web service attached to the etype", a mapping to XML is shown below:

```
<service
      url    ="later on this url will represent the URL of a web service
                  attached to the etype"/>
```

5.4 Conclusion

In this chapter we have provided the hierarchy of all the objects that can be represented in C-XML and have presented the abstract syntax of C-XML. We have also described the constituents of a C-XML document and have outlined the rules that must be followed to produce a valid C-XML document. Finally, we have demonstrated a mapping between C-XML and XML.

Chapter 6

WordNet as BK

In this chapter we describe how to import knowledge from WordNet to the Background Knowledge (BK) of the faceted lightweight ontologies. We report every kind of relation available between synsets in WordNet, and we show how to accommodate the synsets and relations in the BK.

The chapter structure is as follows. In Section 6.1, after analyzing the relations between synsets in WordNet, we present their import procedure. In Section 6.2 we provide a few observations about import. Section 6.3 reports evaluation results, and we conclude the chapter in Section 6.4.

6.1 Importing WordNet 2.1

In this section we describe and exemplify all the lexical and semantic relations that exist in WordNet 2.1. We also provide rules to set precision/recall of a given relation, and provide a description of the import procedure.

6.1.1 WordNet Relations

A relation in WordNet can be represented as a triple < *source_category, relation, target_category*>, e.g., n @ n, where n represents a noun, and

@ represents a hypernym relation. We use a for adjective and adjective satellite, v for verb, and r for adverb. There are 26 relations in WordNet 2.1 that fall into two basic kinds: lexical relation and semantic relation. Among these 26 relations, 4 are only lexical, 15 are only semantic, and the rest are both lexical and semantic. For the sake of simplicity and readability, we provide a representative word per synset rather than the whole synset. We provide one example per relation, and each example is followed by every possible combination of source and target categories.
All these relations are provided below:

- 4 relations are **only lexical**:
  - Antonym (!): e.g., *never* is an Antonym of *always.* (i) *n ! n* (ii) *a ! a* (iii) *r ! r* (iv) *v ! v.*
  - Derivationally related form (+): e.g., *personhood* derived from *person.* (i) $n + n$ (ii) $n + a$ (iii) $n + v$ (iv) $a + v$ (v) $a + n$ (vi) $r + a$ (vii) $a + r$ (viii) $v + a$ (ix) $v + n$.
  - Participle of verb (<): *applied* is a Participle of verb of *apply.* (i) $a < v$.
  - Pertainym or Derived from adjective (\): *smartly* is a Pertainym of *smart.* (i) $a \backslash a$ (ii) $a \backslash n$ (iii) $r \backslash a$.
- 15 relations are **only semantic**:
  - Attribute (=): *measure* is an Attribute of *standard.* (i) $n = a$ (ii) $a = n$.
  - Similar to (&): *ample* is Similar to *abundant.* (i) *a & a.*
  - Hypernym (@): *sleep* is a Hypernym of *nap.* (i) *n @ n* (ii) *v @ v.*
  - Instance hypernym (@i): *battle* is an Instance hypernym of *Battle_of_Britain.* (i) *n @i n.*

- Hyponym ($\sim$): *no_man's_land* is a Hyponym of *land*. (i) $n \sim n$ (ii) $v \sim v$.
- Instance hyponym ($\sim$i): *Berlin_airlift* is an Instance hyponym of *airlift*. (i) $n \sim i\ n$.
- Member holonym (#m): *orthography* is a Member holonym of *punctuation*. (i) *n #m n*.
- Member meronym (%m): *eta* is a Member meronym of *Greek_alphabet*. (i) *n %m n*.
- Part holonym (#p): *lion* is a Part holonym of *mane*. (i) *n #p n*.
- Part meronym (%p): *wishbone* is a Part meronym of *bird*. (i) *n %p n*.
- Substance holonym (#s): *blood* is a Substance holonym of *blood_plasma*. (i) *n #s n*.
- Substance meronym (%s): *oxygen* is a Substance meronym of *ozone*. (i) *n %s n*.
- Cause (>): *stay_up* is a Cause of *keep_up*. (i) $v > v$.
- Entailment (*): *snore* is an entailment of *sleep*. (i) $v * v$.
- Verb group ($): *preen* and *dress* are connected by a Verb group relation. (i) *v $ v*.

• 7 relations are **both lexical and semantic**:

- Also see ($\wedge$): *abundant* and *ample*. (i) $a \wedge a$ (ii) $v \wedge v$.
- Domain of synset - TOPIC (;c): *cooking* to *egg*, *botany* to *herbaceous*. (i) *n ;c n* (ii) *a ;c n* (iii) *r ;c n* (iv) *v ;c n*.
- Member of this domain - TOPIC (-c): *take a hit* to *drug*. (i) *n -c n* (ii) *n -c a* (iii) *n -c r* (iv) *n -c v*.

- Domain of synset - REGION (;r): *UK* to *rugby.* (i) *n ;r n* (ii) *a ;r n* (iii) *r ;r n* (iv) *v ;r n.*
- Member of this domain - REGION (-r): *ghost town* to *west.* (i) *n -r n* (ii) *n -r a* (iii) *n -r a* (iv) *n -r v.*
- Domain of synset - USAGE (;u): *comparative* to *fewer.* . (i) *n ;u n* (ii) *a ;u n* (iii) *r ;u n* (iv) *v ;u n.*
- Member of this domain - USAGE (-u): *polycillin* to *trade name.* (i) *n -u n* (ii) *n -u a* (iii) *n -u r* (iv) *n -u v.*

6.1.2 Rules for Precision/Recall

We measure how much a source concept is contained in a target concept in a relation, and vice versa. We put the measures as precision/recall in the hierarchical and associative relational tables. The values of precision/recall, typically in the range [0,1], can be expressed as a pair (N1, N2), where: N1 and N2 are real numbers; N1 represents how much of the source concept is contained in the target concept; and N2 represents how much of the target concept is contained in the source concept.

In the current import, the value of N1 and N2 can be either 1 or 0 or -1, where:

(i) 1 means source/target concept is fully contained in the target/source concept,

(ii) 0 means source/target concept is not contained in the target/source concept and

(iii) -1 means source/target concept has a (non empty) intersection with the target/source concept but we do not know how much of the source/target concept is contained in the target/source concept.

Given three concepts A, B, and REL, where A is the source concept, B is the target concept, and REL is the concept of the relation which connects

A and B, the established rules for measuring the values of precision/recall for both the source and target are provided below:

1. If the REL is EQUIVALENCE, then the values are (1, 1)
2. If the REL is MORE_GENERAL, then the values are (-1, 1)
3. If the REL is ANTONYM, then the values are (0, 0)
4. If the REL is (NON EMPTY) INTERSECTION, then the values are (-1, -1)

6.1.3 Import Procedure

There is one sense index file, four synset files, and four exception files in WordNet. From now on we will refer to all the synset files as a single synset file and to all the exception files as a single exception file. Each line of the synset file represents a unique synset, and each line of the exception file represents an exceptional form of a word.

1. Parse the sense index file to retrieve the sense rank of each sense and create an index of the sense ranks in the memory where index key schema is sense_lemma#synset_offset. Go to the first line of the synset file.
2. Parse the current line of the synset file to retrieve gloss, synset offset, pos, and all the synonymous word senses of each synset.
3. Out of the information gathered in the previous step, create a synset and convert this synset into a concept. In addition, create a record for this concept into the relational table concept.

4. Create a record of the synset, which corresponds to the concept created above, in the relational table synset. Assign the pos value retrieved in step 2 as the pos value of the synset in the same table. In addition, create an index of the synset and the corresponding concept in the memory, where the index key schema is synset_pos#synset_offset.

5. Create a record in the synset description relational table with the gloss retrieved in step 2.

6. Take the lemma of each synonymous word sense retrieved in step 2 and create a record of the lower cased lemma in the relational table word. Also create a record for every cased lemma in the relational table sense. Assign the sense number retrieved in step 1 as the sense rank in the same table. Check the exception file, if the word has an exceptional form. If the exceptional form of the word is available, create a record in the relational table word_form with this exceptional form.

7. If all the lines of the synset file are parsed, go to step 8. Otherwise, go to the next line and repeat steps 2 to 7.

8. Convert all 26 relations into concepts and create records for them in both the relational table concept and in the memory. Go to the first line of the synset file.

9. Parse the current line of the synset file to retrieve the synset (source synset) and all the relations with the other synsets (target synsets).

10. Retrieve the concept of the source synset, the concept of each target synset, and the concept of the relation that exists between these two synsets. If the relation is a hierarchical relation (consider hypernym, hyponym, holonym, and meronym relations as hierarchical

relations), create a record of the relation in the relational table hierarchical_relation. Otherwise, create a record in the relational table associative_relation.

11. If all the lines of the synset file are parsed, the import is over. Otherwise, go to the next line and repeat step 9 to 11.

The following decisions act as a complement to steps 8 and 10 given above.

1. Both Lexical and Semantic Relations: Except 'Also see', all the relations in both the lexical and semantic relation category are the Domain of synset relations and their inverses. In the Domain of synset relations, a domain concept usually represents the domain of the other concept in the relation; e.g., the concept egg falls into cooking domain. However, except for the 'Also see' relation, we do not import relations that are both lexical and semantic (Domain relations).

2. Lexical Relations: Unlike semantic relations, lexical relations are defined between a word of one synset and a word of another synset; e.g., two synsets, {natural object} and {artifact, artefact}, have anotonym relations between natural object sense and artifact sense. The current data structure of the BK provides infrastructure for importing relations between synsets. However, it does not provide support for importing relations between words. We approximate them as relationships between the corresponding concepts.

3. Inverse Relations: In WordNet, some relations have their inverse form; e.g., hyponym is an inverse form of hypernym. The WordNet database retains both the relation and the inverse form of this relation. However, instead of importing both hypernym and hyponym, we import relations in only one direction; e.g., we keep hyponym. We suggest

maintaining an additional relation either in the business logic or in the backend as a separate table from data structure for each inverse relation; e.g., we maintain the relation <hyponym, inverse, hypernym>, where hyponym is a source concept, inverse is a relational concept, and hypernym is the target concept. We generate the inverse relation of a relation by replacing the source with the target and by replacing the relation with the inverse form. However, in some cases the relation and the inverse form are the same; e.g., antonym, similar, and related to. In this case, we import both directions.

4. Hierarchical Relations: For each hierarchical relational table entry, we put the more specific concept of a relation as a source concept, and the other one as a target concept.

5. Transitivity of relations: There are some relations in WordNet, which are transitive; e.g., hypernym, holonym, etc. The transitivity of some of the relations (e.g., hyponym), can be exemplified as heifer (young cow) is a hyponym of cow and cow is a hyponym of animal; by computing transitivity we find that heifer is a hyponym of animal. We import transitive relations in a hierarchical relation table and the rest in an associative relational table. We only consider hyponym, instance hyponym, part meronym, and their inverses, as transitive relations.

6. There are two kinds of relational table, hierarchical relational table and associative relational table, in the concept part of the BK. We classify the relations and put them into the corresponding relational table of the BK as demonstrated below.

Table 6.1: Relations in WordNet and BK.

Relation	**Concept**	**Inverse**	**Hierar-chical**	**Associ-ative**	**Precisio-n/Recall**
Antonym	**Antonym**, opposite word, opposite	**Antonym**, opposite word opposite	no	yes	0,0
Derivationally related form	Related, **Related to**	Related, **Related to**	no	yes	-1,-1
Participle of verb	**Verb**	**Participle**, participal	no	yes	-1,-1
Pertainym or Derived from adjective	**Original**	**Derived**	no	yes	-1,-1
Attribute	Property, **attribute** dimension	Property, **attribute** dimension	no	yes	-1,-1
Similar to	**similar**	**similar**	no	yes	1,1
Hyponym	**Hyponym**, subordinate, subordinate word	**Hypernym** superordinate, superordinate word	yes	no	1,-1
Instance Hyponym	Example, illustration, **instance**, representative	**Class**, category, family	yes	no	1,-1
Member Meronym	**Member**	**Group**, grouping	no	yes	-1,-1

Part Meronym	**Part**, portion, component part, component, constituent	Holonym, **whole name**	yes	no	1,-1
Substance Meronym	**Substance** component part,	**Compound whole name**	no	yes	-1,-1
Cause	**Cause**	Consequence, **effect**, outcome, result, event, issue, upshot	no	yes	-1,-1
Entailment	Deduction, **entailment**, implication	**Entail**, implicate	no	yes	-1,-1
Verb group	Group, **aggroup**	Group **aggroup**	no no	yes yes	-1,-1 1,1
Also see	Consider, take, deal, **look at**	Consider, take, deal, **look at**	no	yes	-1,-1

Note 1: in the table given above, we correlate each relation to a concept. Some relations directly map to WordNet synset, e.g., ‘Antonym’. For relations that do not directly map to the WordNet synset, e.g., ‘Derivationally related form’, we engineer to generate their corresponding concept from the concepts available in the BK. In some cases, we add new concepts to

the BK for encoding the concept of the relation. In some cases we extend the synset for concepts; e.g., we will add "is-a" word to a Hyponym synset and mark it as the preferred term.

Note 2: We import the hyponym and not the hypernym. This way, the use of the relationships, in particular the part/substance/member meronyms, is more intuitive. We provide some examples of how to read the relations in Table 6.2:

Table 6.2: The source and the target in the relations.

Source	**Target**	**Relation**
Dog	Canine	Is-a **hyponym** of
In English: *Dog is a hyponym of Canine.*		
Napoleonic Wars	War	Is-a(n) **instance** of
In English: *Napoleonic Wars is an instance of War.*		
Rome	Italy	Is-a **part meronym** of
In English: *Rome is a part meronym of Italy.*		
Stay-up	Keep-up	Is-a **cause** of

Note 3: Except for similar and antonym relations, all the associative relations have (-1, -1) as their precision/recall value pairs.
Note 4: We place the whole sysnet in the concept column with the main word in bold. For all holonym kinds, there are numerous very similar senses in WordNet. For this reason we provide their full synset along with a description, as in the following:

(i) Part: **part**, portion, component part, component, constituent (something determined in relation to something that includes it)
(ii) Substance: **substance** (the real physical matter of which a person or thing consists)
(iii) Member: **member** (anything that belongs to a set or class)

In addition, we provide the full synset and description of compound and similar synsets, for the reason that each of them has more than one synset with exactly the same elements.
(i) Similar: **similar** – ((of words) expressing closely-related meanings)
(ii) Compound: **compound** – (consisting of two or more substances, ingredients, elements, or parts; "soap is a compound substance"; "housetop is a compound word"; "a blackberry is a compound fruit")

6.2 Open Issues

Relations between categories: The micro grammar provided in section 6.1.2 can be used to both constrain and infer the source and target categories of the relations during import. For example, there are some relations in WordNet which only connect verbs; e.g., Cause, Entailment, and Verb group. In contrast, there are some relations which connect only nouns; e.g., holonym, and meronym. We can provide constraint to the import system such that relations will only be imported if the source and target category match with the given micro grammar. We also can infer the source and target categories if we know the relation between them. In some cases we impose constraints on, and infer, the categories, from the kind of relation that exists between the concepts. This grammar can be used in tasks such as importing thesauri into BK so as to infer the category of the terms used.

6.3 Final Evaluation

We developed an import system following the procedure given in section 6.1.3. We then imported WordNet 2.1 to the BK. The statistics of the imported objects are provided in Table 6.3.

Table 6.3: The statistics of the imported objects.

Name of the Object	**Number of instances imported**
Concept	117,597
Hierarchical Relation	105,647
Associative Relation	112,174
Synset	117,597
Word	147,252
Sense	207,019
Word_form	4,728

According to the statistics of WordNet 2.1, there are 117,597 synsets and 207,019 word-sense pairs. The same results were found after import, as can be seen from the table. For this reason, we can conclude that our designed system is complete.

6.4 Conclusion

In this chapter we have reported the various kinds of synsets and relations available in WordNet. We then described a procedure to import these synsets and relations to the BK, and presented our observations. Finally, we have reported the results of our evaluation of the import work.

Chapter 7

GeoWordNet as BK

In this chapter we describe the import procedure of the knowledge available in GeoNames that is used for the enrichment of BK, which we built in the previous chapter. At this step, of import the BK containing WordNet is fully integrated with GeoNames. As a result, a new enriched level of BK is produced which we call GeoWordNet, and which consists of millions of entity synsets and hundreds of thousands of concept synsets.

The chapter is organized as follows. In Section 7.1 we describe the criteria for selecting the knowledge source. In Section 7.2 we briefly describe the facet. Section 7.3 provides the list of the GeoNames classes selected to build the faceted hierarchy. Section 7.4 presents the main research issues. In Section 7.5 we describe the GeoNames import procedure, and we report the statistics of the import in Section 7.6. Section 7.7 presents some open issues we faced during import and Section 7.8 concludes the chapter.

7.1 Selection Criteria

A fundamental step in each process of knowledge integration is to evaluate candidate ontologies by identifying possible missing knowledge (i.e., in terms of concepts and relations), and by identifying which knowledge is to be removed, reallocated, or changed [41]. It is also fundamental to identify knowledge which is functional to the goal it is meant to serve [39], which in our case is to classify, search, and match classification nodes and documents.

The motivation of this work is to enrich BK with geo-spatial information, as they can improve the performance of the applications, which are connected to real life [2]. Adding geo-spatial information to lexical knowledge bases is exemplified in the works [1], [50], which added GEMET[1] to WordNet, and GNS and GNIS[2] to WordNet, respectively. However, in our case GeoNames has been selected as the best source of geo-spatial information. It has nine top level classes, and each of the classes has a number of subclasses. Classes, their descriptions, and statistics of the subclasses are provided in the Table 7.1. GeoNames contains approximately 7 mil-

Table 7.1: Classes and Feature Classes in GeoNames.

Feature Class	**Description**	**Number of subclasses**
A	administrative divisions of a country. It also represents states, regions, political entities and zones.	16
H	water bodies, e.g., ocean, sea, river, lake, stream, etc.	137
L	parks, areas, etc.	49
P	populated places, e.g., capitals, cities, towns, small towns, villages, etc.	11
R	roads and railroads	23
S	spots, buildings and farms	242
T	mountains, hills, rocks, valleys, deserts, etc.	97
U	undersea areas	71
V	forests, heaths, vineyards, groves, etc.	17
Total Number of Subclasses		663

[1]See http://www.eionet.europa.eu/gemet/about

[2]See http://earth-info.nga.mil/gns/html/index.html and http://geonames.usgs.gov respectively

lion entities distributed in the 663 distinct sub-classes, as shown in the table above. There is therefore an implicit instance-of relation between a class and its corresponding instances. Classes from GeoNames will be naturally integrated as concepts in (the ontological part of) the BK, while their instances will be integrated as entities, together with their corresponding relevant attributes (i.e., latitude and longitude). At this stage, the Location e-type must be defined. In addition, a significant portion of the locations in GeoNames are connected via part-of relations; e.g., Italy is part-of Europe. To record this information, the corresponding entities have to be connected by the topological relation [12] "part-of". Here we have two options; either we create a specific attribute or we extend the data structures to explicitly codify such a relation. We choose the former option. A fundamental decision that needs to be made is which locations we want to import. Our choice is to import into BK all the classes, as concepts, and the locations, as entities, which are explicitly used for the construction of a facet (as described in the section below).

7.2 Facets

As already described in Section 3.4 a facet is a group of hierarchically related concepts. According to the given principles of division from the facet theory, siblings in a facet hierarchy must share the same characteristic(s) [14]. A typical example is the Administrative Division facet. For instance, in Figure 7.1, the hierarchy can be characterized as *Continent* $->$ *Country* $->$ *Province* $->$ *District*. It seems clear to us that geographical locations which belong to feature class A and P can be used for the construction of the Administrative Division facet, which is known as a facet for Space (following the traditional Library and Information Science terminology). Similar facets will be built from the other feature classes.

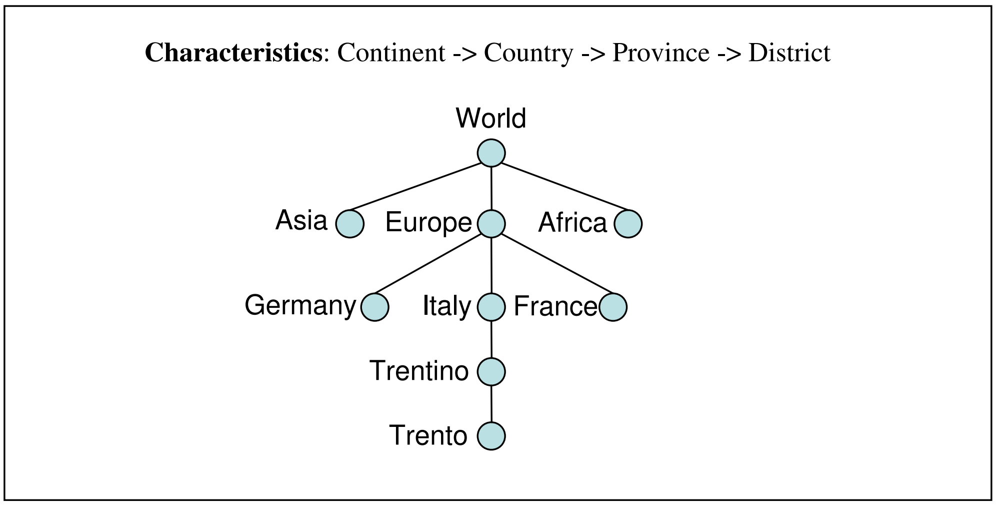
Characteristics: Continent -> Country -> Province -> District
World
Asia
Europe
Africa
Germany
Italy
France
Trentino
Trento

Figure 7.1: A simplified example of Administrative Division facet.

Notice however, that facets can be heterogeneous objects, in the sense that they can contain both concepts and entities. This is particularly true in the Geographical domain. Therefore, each facet is potentially distributed between the ontological and the entity part of the BK, as described in Figure 7.2. Once they are imported, the issue of how to visualize and maintain these facets has to be addressed. This issue is beyond the scope of this thesis, however.

7.3 Classes Selected

In the following we provide the list of the classes selected for the construction of the facets, according to the criteria given in the previous sections.

7.3.1 The Administrative Division facet

The Administrative Division facet is constituted by the non-empty classes from feature classes A and P. They are reported in Table 7.2.

In GeoNames, continents are listed in the "readme.txt" file. In addition, notice that in GeoNames there is no explicit class associated with

Table 7.2: Classes Selected for the Administrative Division facet.

CONT (continent), PCLI (country), PPLC (capital of a political entity), ADM1 (first-order administrative division), ADM2 (second-order administrative division), ADM3 (third-order administrative division), ADM4 (fourth-order administrative division), ADMD (administrative division), PPLG (seat of government of a political entity), PPLA (seat of a first-order administrative division), PPLS (populated places), PPL (populated place), PPLL (populated locality), PPLR (religious populated place), PPLX (section of populated place), PPLW (destroyed populated place), PPLQ (abandoned populated place), STLMT (Israeli settlement)

city, as there are three separate files for cities (divided according to their population; i.e., "cities1000.txt", "cities5000.txt", and "cities15000.txt"). We consider city as an additional class and identify instances of this class from these files. In order to build the facet, we decided to redefine such classes as the following:

CONTINENT $\sqsubseteq$ CONT (continent)

COUNTRY $\sqsubseteq$ PCLI (country)

ADMINISTRATIVE_DIVISION $\sqsubseteq$ ADM1 (first-order administrative division) $\sqcup$ ADM2 (second-order administrative division) $\sqcup$ ADM3 (third-order administrative division) $\sqcup$ ADM4 (fourth-order administrative division) $\sqcup$ ADMD (administrative division)

CITY $\sqsubseteq$\{all the locations explicitly listed the three city files\}

CAPITAL $\sqsubseteq$ PPLC (capital of a political entity)

POPULATED PLACE $\sqsubseteq$ PPLG (seat of government of a political entity) $\sqcup$ PPLA (seat of a first-order administrative division) $\sqcup$ PPLS (populated places) $\sqcup$ PPL (populated place) $\sqcup$ PPLL (populated locality) $\sqcup$ PPLR (religious populated place) $\sqcup$ PPLX (section of populated place) $\sqcup$ PPLW (destroyed populated place) $\sqcup$ PPLQ (abandoned populated place) $\sqcup$ STLMT (Israeli settlement) PPLC (capital of a political entity) $\sqcup$ CITY

LOCATION $\sqsubseteq$ CONTINENT $\sqcup$ COUNTRY $\sqcup$ ADMINISTRATIVE_DI-

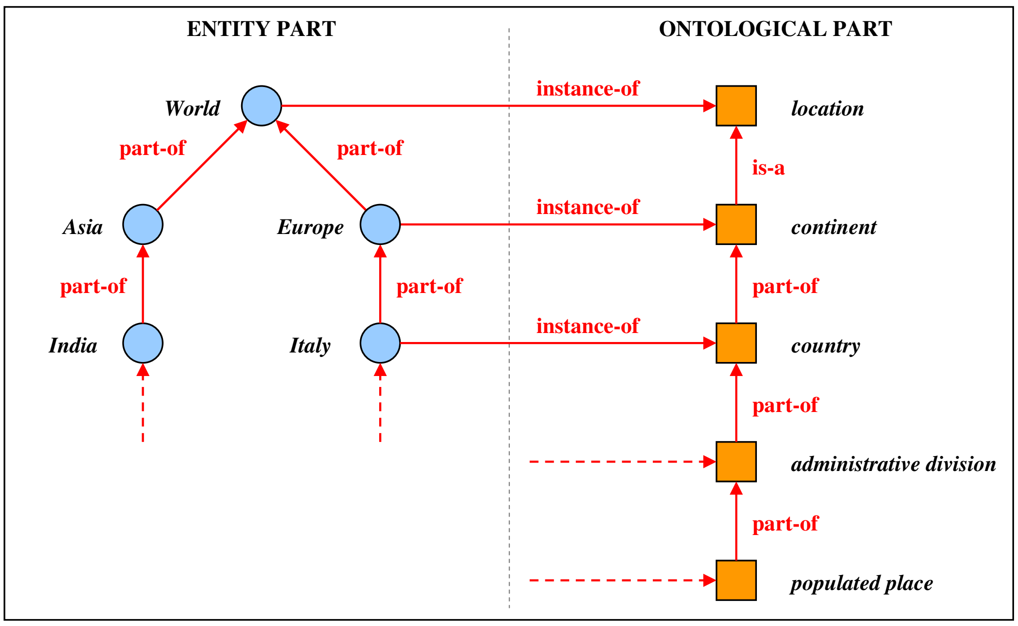

Figure 7.2: The Administrative Division facet.

VISION ⊔ CAPITAL ⊔ CITY ⊔ POPULATED PLACE

In this way, the characteristics used to construct the Administrative Division facet are always uniform, as depicted in Figure 7.2 (on the right).

The characteristics used to construct the Administrative Division facet constitute the backbone structure of the conceptual side of the facet (on the right). Analogously, the corresponding instances will constitute the entity side of the facet (on the left). Some final notes:

(a) GeoNames classes are designed to be disjointed;
(b) POPULATED PLACE includes all the classes from feature class P, plus the CITY class;
(c) LOCATION is constituted by all the classes from feature classes A and P, plus the CITY class;
(d) CAPITAL is more specific than CITY, which is more specific than ADMINISTRATIVE DIVISION;
(e) POPULATED PLACE is part of an ADMINISTRATIVE DIVISION,

which is part of a COUNTRY, which is part of a CONTINENT;
(f) All the concepts in bold in Table 7.2 and the corresponding semantic relations between them, have to be encoded in the ontological part of the BK;
(g) World is the entity root of the facet, while LOCATION is the concept root of the facet.

7.4 Research Issues

In this section, we list the main research issues addressed in: building facet hierarchies; codifying classes and locations; regarding homonymy; the relations imported; storing linguistic information in multiple languages; the identification of preferred terms; regarding glosses in multiple languages; sense ranking; and regarding provenance information.

7.4.1 Issue in building facet hierarchies

Description: We select information from GeoNames for the specific purpose of building facet hierarchies.
(a) What are the facets that can be built using GeoNames? (b) What are the relevant locations to import? (c) Which characteristics have to be used to construct them?

Solution: At the moment we only use locations from the feature classes A and P. As described in the previous section, locations are used for the construction of the Administrative Division facet.

Drawbacks and open problems: Even if more facets can be built, we currently build only the Administrative Division facet. We need to analyze the other feature classes to build additional facets. For instance, a possible candidate is the Water Body facet which can be built using locations from the feature class H.

Notice that even if all the GeoNames classes are assumed to be disjointed, there are clear cases in which this is somewhat artificial. For instance, Rome as Capital and Rome as Populated Place are considered distinct locations in GeoNames, and therefore they are assigned different IDs.

7.4.2 Issue in codifying classes and locations

Description: Which are the relevant GeoNames classes and locations to import? How to codify them in the BK?

Solution: As in the research issue above, we import everything that proves to be functional to facets. Therefore, we import a class, as a concept, or a location, as an entity, in the BK when it is directly (it corresponds to a node) or indirectly involved (a class whose instances correspond to a node) in a facet.

First of all, most of the names for both classes and instances have multiple senses in the BK (originally imported from WordNet) and are therefore subject to disambiguation. Since we identified only a few relevant classes (as in the issue above), and they are all already present in the BK, we decided to manually disambiguate them by selecting the right sense for each class name.

For the Administrative Division facet they are the following (the word, in bold, followed by the WordNet gloss):

- **continent**: *one of the large landmasses of the earth*
- **country**: *the territory occupied by a nation*
- **administrative division**: *a district defined for administrative purposes*
- **city**: *a large and densely-populated urban area*

- **capital**: *the capital city of a nation*
- **populated place**: A populated place is a kind of place. It is also populated. The sense of populated is *furnished with inhabitants* and the sense of place is *a point located with respect to surface features of some region*
- **location**: *a point or extent in space*

On the other hand, this is unnecessary for instances because currently there are no spatial entities in the entity part of BK. The small portion of them already in the BK will be removed, since they are wrongly codified as concepts. So, we create an entity in the BK for each location to import, without caring about the overlap with already existing entities in the BK.

Drawbacks and open problems: We need to analyze how the concepts are related in WordNet and reorganize them according to the set of semantic relations we need. We need to remove the entities (not only the spatial ones) imported from WordNet, from the conceptual part of the BK.

7.4.3 Homonymy Issue

Description: There are plenty of places with the same name. For example, Miramare is a Populated Place in Provincia di Forli (Administrative Division in Italy) and also a Populated Place in Provincia di Trieste (Administrative Division in Italy). How do we differentiate them?

Solution: It is quite safe to assume that two countries as well as two Administrative Divisions at the same level (e.g., regions or provinces) in the same country cannot have the same name. Therefore, homonymous locations can be differentiated by their parents. In the example above, the former Miramare has parent Provincia di Forli and the latter has parent Provincia di Trieste.

Drawbacks and open problems: Even though we have not encountered any such cases until now, it is conceivable that two homonymous locations might have the same parent and therefore cannot be differentiated.

7.4.4 Issue in the relations imported

Description: Which kind of relations do we import?

Solution: We import the part-of (part-meronym) relations between entities and instance-of (instance-hyponym) relations between entities and corresponding concepts. For example, Italy is part-of Europe and Italy is an instance-of Country. Such information is codified in the GeoNames file "allcountries.txt".

However, in GeoNames, physical locations such as lakes, rivers, and mountains are associated with countries. We can say that they are implicitly related by a part-of relation. However, there are several problems with this choice:

- In many cases they belong to more than one country (i.e., very long rivers). However, what GeoNames does is to associate different IDs with physical locations which cross national borders, sometimes with different names for the same physical entity (i.e., they refer to the name used in that country).
- Having multiple parents is not allowed in the theory of facets.
- They are only associated with countries and not with more/less specific concepts. Therefore, questions such as which cities a given river crosses cannot be answered.

As a consequence, these kinds of relations (useful for reasoning) need to be identified in a different way. The final choice is to explicitly encode the part-of only for the Administrative Division facet.

Drawbacks and open problems: We suggest using a geographical database for the identification of the parthood of the other locations, such as mountains and lakes, and hide the computation in a matching service between entities.

7.4.5 Issue in storing linguistic information in multiple languages

Description: GeoNames is a multilingual resource in which each location can have different names in different languages. How then to encode multilingual information in the BK?

Solution: In the current GeoNames import we only consider English and Italian. Each location has an English name, but only a small portion of them also have an Italian name. As previously stated, for each location we create an entity in the BK, regardless of the language.

Current data structures do not support the creation of synsets for entities. Rather, they support the creation of specific language-dependent "string values". For each entity, we create corresponding English string values for the Name attribute in (the linguistic part of) the BK, which also includes possible alternative English names provided by GeoNames. When one or more alternative Italian names are available, we also create one or more Italian string values for that location.

Drawbacks and open problems: Only two languages (English and Italian) are covered in the current import. Unfortunately the coverage is relatively poor for Italian and other non-imported languages.

7.4.6 Preferred terms identification issue

Description: The first term which appears in the list of string values for the Name attribute of an entity can be considered the preferred term for that entity. Except for the English ASCII name (a column in the "allcountries.txt” file), all the names of the locations in GeoNames are listed as alternative names. A location can have multiple alternative names. In the list of alternative names, preferred terms are marked only for some of the locations.

How do we select a preferred term for the rest of the locations?

Solution: A. For the English names that:

- have the preferred term marked, we select this term as the preferred term and we place it in the first position of the English string values of the Name attribute generated for this location. We place the English ASCII name in the second position.
- do not have the preferred term marked, we select the English ASCII name as the preferred term and we place this term in the first position of the English string values of the Name attribute generated for this location.

In both cases, we place the rest of the terms from the list of the English alternative names in the consecutive positions on a First Come First Serve (FCFS) basis.

B. For the Italian names that:

- have the preferred term marked, we select this term as the preferred term and we place it in the first position of the Italian string values of the Name attribute generated for this location.
- do not have the preferred term marked, we select the first term, as the preferred term, from the list of the Italian alternative names on FCFS basis.

In both cases, we place the rest of the terms from the list of the Italian alternative names in the consecutive positions on an FCFS basis;

Drawbacks and open problems: When not explicitly marked, we cannot be sure that what we have selected is effectively the preferred name for the location. For instance, according to such rules, the preferred name for Italy (the country) will be Italian Republic.

7.4.7 Issue regarding glosses in multiple languages

Description: A gloss is typically used to provide a description about a term or a set of terms (i.e. a synset) to enable them to be better understood by user and therefore for a correct disambiguation. How are the glosses generated for the locations imported from GeoNames?

Solution: We use the information encoded in the part-of hierarchy to automatically generate glosses. Notice, however, that for gloss generation we use the complete set of classes, as in Table 7.2.

Moreover, notice that for the purpose of generating the gloss we consider the classes ADMD (administrative division), ADM1 (first-order administrative division), ADM2 (second-order administrative division), ADM3 (third-order administrative division), and ADM4 (fourth-order administrative division) as administrative division.

(a) Gloss generation schema for English can be defined as shown below:
Child concept + " is " + article + class concept + " in " | " of " + [the] + parent concept
Note that [the] means optional use of "the" and | (pipe) means either in or of is used in the gloss.

Moreover, the above schema is expanded by appending "(administrative division in " + country + ")" if the parent concept represents an administrative division. We refer to the schema without expansion as the base schema, and the one with expansion as the expanded schema.
(b) Gloss generation schema for Italian can be defined as shown below:

Child concept + “ è ” + article + class concept + “ in ” | “ nel ” + parent concept
Note that “è”, “in”, and “nel” are the Italian translations of “is”, “in”, and “in the”, respectively.

Moreover, the schema above is expanded by appending “(divisione amministrativa in ” + country + “)” if the parent concept represents an administrative division. Here, “divisione amministrativa in” is an Italian translation of “administrative division in”.

Drawbacks and open problems: Sometimes the generated glosses are not very meaningful. For example, Trento is an administrative division in Trento (administrative division in Italy). In this example, both the source concept and target concept name is Trento, and in both cases Trento is an administrative division, although at different levels (province and municipality in this case). It is our belief that by leaving the level number we do not improve the gloss.

7.4.8 Issue in sense ranking

Description: Words in GeoNames can have multiple senses. Should the entities be ranked like we do for concepts?

Solution: Entity ranks can be computed by taking into account the class of the word (the name of the location). In particular, we can apply the following rule: the higher the class location is, in the facet hierarchy, the higher the rank. We follow the order of the classes given for the characteristics (see the research issue: Building the facet hierarchies). We consider the Continent class as the highest class, Country as the second highest class, and so on. Here, the intuition is that instances of the classes at higher levels are better known than the instances of the classes at lower levels of the hierarchy.

For instance, it is quite common to have a city with the same name as its administrative division. In this case, we assign the higher rank to the administrative division (e.g., we have Rome as a province in Italy, which includes the city Rome, which is also the capital of Italy).

Drawbacks and open problems: In the facet hierarchy the instance at a lower level being better known than the instance at a higher level can occur. For example, the state Georgia in USA is better known than the country Georgia in Eastern Europe.

7.4.9 Provenance information issue

Description: Provenance information is useful to keep track of the original locations in GeoNames, and can be used to update, synchronize, and verify the knowledge in the BK. How can the provenance information be maintained in the BK, however?

Solution: We propose the creation of a special attribute for the spatial entities whose value is their GeoNames ID. Using this ID, it is possible (and not too costly) to synchronize the information in the BK with the information in GeoNames by running a daemon which monitors daily changes. Provenance information may be lost if GeoNames decides to change the policy of the IDs in the future.

7.5 Import Procedure

7.5.1 The intermediate schema

Before importing relevant classes and corresponding locations, we created a set of relational tables, which constitute what we call the *intermediate schema*, to store the whole content of GeoNames. They are the following:

Class (id, name, gloss),

Location (geoNamesId, name, class, latitude, longitude),
AlternativeNameENG (geoNamesId, name),
AlternateNameITA (GeoNamesid, name), and
Part-of (SourceId, TargetId).

The Class table stores the GeoNames classes. Location stores all the GeoNames locations and their relevant attributes. In particular, we store the English name in the Location table, the alternative English names in the AlternativeNameENG table, and the alternative Italian names in AlternateNameITA table. In the Part-of table, SourceId and TargetId are the source and the target location GeoNames IDs, respectively.

After creating the intermediate schema, we then focus on the procedure to load data into it. The GeoNames file “allcountries.txt” contains geo-spatial information about all the countries of the world. Each line contains a single GeoNames location. The file “alternatenames.txt” contains alternative names of the locations in different languages. Each line contains an alternative name. Classes and related information are in the file “featureCodes.txt”. Continents are listed in the “readme.txt” file that also contains a complete description of the format of the files distributed by GeoNames. We follow the steps below:

1. Parse the alternatenames.txt file to retrieve the alternate names, GeoNames ID, and language code for each GeoNames name, and create an index of the alternate names in memory where the index key is the GeoNames ID. We consider only the English and Italian names. Go to the first line of the “allcountries.txt” file.

2. Parse the current line of the “allcountries.txt” file to retrieve the GeoNames ID, name, latitude, and longitude of the location. Re-

trieve its corresponding class and build the gloss as described in the previous section.

3. Out of the information retrieved in the previous step, create an entry in the table Class (if the retrieved class name is not yet available in that table) and create an entry in the table Location. Depending upon the availability, create an entry in AlternativeNameENG (alternative names in English) and AlternateNameITA (alternative names in Italian). Finally, create an entry in the table Part-of, where source is the currently processed location ID, and target is the location ID of the parent (one level higher in the hierarchy).

4. If all the lines of the "allcountries.txt" file are parsed, the import is over, otherwise go to the next line and repeat steps 2 to 4.

Tables created at this stage can be used to import new concepts and entities in BK.

7.5.2 Importing from the intermediate schema to BK

Data from the intermediate schema is used for the construction of the facets by means of their basic constituents; i.e., the concepts, the entities, and the relations between them. The concepts and conceptual relations between them are codified by hand, while the entities and relations are imported from the intermediate schema to BK following the macro steps outlined below.
For the Administrative Division facet:

1. Build the entity part of the facet hierarchy by selecting the instances of the classes which forms the part-of hierarchy in the intermediate schema. Classes and instances are available in the relational tables Class and Location, respectively.

2. Create an entity in the BK for each location in the hierarchy.

3. Create a corresponding English and Italian (when alternative names in Italian are available) string value for the Name attribute.

4. Check the AlternativeNameENG table to see if the entity has an alternative name in English. Append a value to the Name attribute for English for each alternative name. Similarly, check AlternativeNameITA and add a value to the Name attribute for Italian for each alternative name.

5. Create an instance-of entry between the entity and the corresponding class concept as described in the previous section (the class that represents the characteristic).

6. Create a part-of entry if the entity is part of another entity in the table Part-of;

7. Generate a gloss for each entity generated in the previous steps.

For the locations which do not belong to the Administrative Division facet:

1. Create an entity in the BK for each location.

2. Create a corresponding English and Italian (when alternative names in Italian are available) string value for the Name attribute.

3. Check the table AlternativeNameENG to see if the entity has an alternative name in English. Append a value to the Name attribute for English for each alternative name. Similarly, check AlternativeNameITA and add a value to the Name attribute for Italian for each alternative name.

4. Create an instance-of entry between the entity and the corresponding class concept as described in the previous section (the class that represents the characteristic).
5. In the gloss put the name of the concept (e.g., city, populated place, etc.) the entity belongs to.

7.6 Statistics

In the following table we report statistical data about the imported classes, locations, part-of relations, and alternative names in the intermediate schema.

Table 7.3: Number of imported objects in the intermediate schema.

Category	**Number of Imported Entities**
Class	664[a]
Locations	6,907,417
Part-of	6,890,382
Alternative English Names	87,539
Alternative Italian Names	11,996

[a]The 663 GeoNames classes, plus a special NULL class which is not included in the set.

We then import the data from the intermediate schema to the BK as described in Section 7.5.2. The statistics relating to the Administrative Division facet are reported in Table 7.4, and the statistics relating to the whole GeoNames import are reported in Table 7.5.

7.7 Open Issues

1. *Removing existing entities from the conceptual part of the BK.* In this import we have several entities (which correspond to uppercased common names) in BK, which we need to remove.

Table 7.4: Objects imported in the Administrative Division faceted hierarchy of the GeoWordNet.

Category	**Number of imported entities**
Instances	2,265,284
Relation between instances	2,265,283
English Synsets	2,265,298
Italian Synsets	6,433
English Words	2,264,380
Italian Words	5,642
English Senses	2,265,298
Italian Senses	6,433
Facets	1[a]
Domains[b]	1

[a]The Administrative Division facet
[b]Space

2. *Building the whole set of spatial facets.* In the current import we build only the Administrative Division facet. However, more facets can be built; e.g., the Water Body facet (including Rivers, Lakes) or the Raised Areas of Land facet (including Hills, Mountains, etc.)

3. *Encoding of names in the "allcountries.txt" GeoNames file.* The names are not encoded uniformly. Sometimes an ASCII version is available, sometimes the UTF-8 encoding is available, and sometimes both are available. However, in most of the cases we only have the ASCII version. This is a problem in non-Windows environments.

4. *The right amount of instances and concepts to be managed.* We import the GeoNames locations as instances of concepts and the classes as concepts. We build the part-of hierarchy and constitute the Administrative Division facet. For the relevant classes we reuse the concepts already present in the BK. Notice that all the locations are imported in the entity part of the BK (this requires the creation of the corre-

Table 7.5: Objects imported from the GeoNames to the GeoWordNet.

Category	**Number of imported entities**
Instances	6,907,417
Relation between instances	6,890,382
English Synsets	6,907,774
Italian Synsets	10,433
English Words	6,901,538
Italian Words	9,718
English Senses	6,907,774
Italian Senses	10,433
Facets	1
Hierarchical Relations in Facets	2,265,283
Domains	1

sponding concept in the ontological part for their classes). However, this is a huge amount of knowledge. How do we select the right amount of partial knowledge from BK? A possible solution could be to select geo-entities, instances, and concepts according to a principle of locality, namely according to the region of the user (for instance, in the context of the Live Memories[3] project we could only select those which are relevant for Trentino).

7.8 Conclusion

In this chapter we have demonstrated the building procedure of GeoWordNet. We have also described the criteria for selecting knowledge sources. According to the criteria, we have selected GeoNames as the source of knowledge for our purpose. We have provided a GeoNames import procedure and have described the facets used for organizing imported knowledge. In addition, we have outlined the main research issues we encountered in

[3]See http://www.livememories.org/

this import. Finally, we have reported the statistics of the import.

Chapter 8

Conclusion

In this thesis we have discussed formal lightweight ontologies, their applications, and their problems. We have outlined two central problems: the inaccuracy of natural language processing in the generation of formal lightweight ontologies from informal ones; and the fact that extraction of axioms in building such ontologies is influenced and limited, respectively, by the poor coverage of background knowledge. We have also introduced the difficulties associated with the reuse of ontology. In addition, we have discussed scalability and dynamics in the management of ontologies. We have proposed faceted lightweight ontologies as a solution to the problems relating to formal lightweight ontologies. We have developed GeoWordNet, specialized on geo-spatial information, to use as Background Knowledge in the faceted lightweight ontologies.

We have provided a formalization of the notion of the faceted lightweight ontology, in order to identify a suitable language for its representation. In the formalization, we have demonstrated that each faceted lightweight ontology has two parts: a formal lightweight ontology and background knowledge. We have discussed formal Web languages to identify their capabilities to represent faceted lightweight ontologies. We have introduced a specification of C-XML which can be used for representing faceted lightweight ontologies. A formal lightweight ontology is often generated from an informal one. C-XML can also be used to represent informal lightweight ontologies, such as classification schemes. Encoding informal lightweight ontologies in C-XML provides the solution to the problem of dynamics.

We have generated some faceted lightweight ontologies in which the source informal ontologies were encoded in C-XML. We have provided GeoWordNet as background knowledge to the generated ontologies. In our experiments, we have found satisfactory performance improvements in the formal lightweight ontology applications.

Our future work includes a detailed study to identify available knowledge sources in all possible domains. This is to be undertaken in order to build a richer knowledge base which can provide better coverage than is currently possible, so that users can generate a faceted lightweight ontology in every possible domain. It also includes the development of the necessary algorithms and tools for importing identified new sources.

Bibliography

[1] M. Angioni, R. Demontis, and F. Tuveri. Enriching wordnet to index and retrieve semantic information. In *Proc. of 2nd Int. Conf. on Metadata and Semantics Research*, 2006.

[2] S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData - adding a spatial dimension to the web of data. In *Proc. of 7th International Semantic Web Conference (ISWC)*, pages 731–746, 2009.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] S. Bechhofer, F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[5] D. Beckett. RDF/XML Syntax Specification (Revised). Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[6] T. Berners-Lee. Weaving the web. In *Orion Business Books*, 1999.

[7] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, (284(5)):34–43, May 2001.

[8] P. Bouquet, F. Giunchiglia, F. V. Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *International Semantic Web Conference*, pages 164–179, 2003.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[10] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[11] J. J. Carroll and J. D. Roo. OWL Web Ontology Language Test Cases. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[12] M. J. Egenhofer and M. P. Dube. Topological relations from metric refinements. In *Proc. of the 17th ACM SIGSPATIAL Int. Conference on Advances in GIS*, 2009.

[13] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. Technical report, World Wide Web Consortium (W3C), October 28 2004. Recommendation.

[14] F. Giunchiglia, B. Dutta, and V. Maltese. Faceted lightweight ontologies. In *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*, pages 36–51, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] F. Giunchiglia, F. Farazi, L. Tanca, and R. D. Virgilio. The semantic web languages. In *Semantic Web Information management,*

a model based perspective. Roberto de Virgilio, Fausto Giunchiglia, Letizia Tanca (Eds.), Springer, 2009.

[16] F. Giunchiglia, V. Maltese, and A. Autayeu. Computing minimal mappings. In *ISWC Ontology Matching Workshop*, Washington DC, USA, 2009.

[17] F. Giunchiglia, V. Maltese, F. Farazi, and B. Dutta. Geowordnet: a resource for geo-spatial applications. In *ESWC*, Heraklion, Greece, 2010.

[18] F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Towards a theory of formal classification. In *Proceedings of the AAAI-05 Workshop on Contexts and Ontologies: Theory, Practice and Applications (C&O-2005)*, Pittsburgh, Pennsylvania, USA, 2005.

[19] F. Giunchiglia, M. Marchese, and I. Zaihrayeu. Encoding classifications into lightweight ontologies. In *Journal on Data Semantics (JoDS) VIII*, Winter 2006.

[20] F. Giunchiglia and P. Shvaiko. Semantic matching. *Knowledge Engineering Review*, 18(3):265–280, 2003.

[21] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *OTM Conferences (1)*, pages 347–365, 2005.

[22] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Discovering missing background knowledge in ontology matching. In *ECAI*, pages 382–386, 2006.

[23] F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Meaning Coordination and Negotiation workshop, ISWC*, 2004.

[24] F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *ESWC*, pages 272–289, 2005.

[25] F. Giunchiglia and I. Zaihrayeu. Lightweight ontologies. In *The Encyclopedia of Database Systems*. Springer, 2008.

[26] F. Giunchiglia, I. Zaihrayeu, and F. Farazi. Converting classifications into owl ontologies. In *Proceedings of Artificial Intelligence and Simulation of Behaviour Convention Workshop on Matching and Meaning*, Edinburgh, UK, 2009.

[27] J. Grant and D. Beckett. RDF Test Cases. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[28] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[29] N. Guarino. Some ontological principles for designing upper level lexical resources. In *First International Conference on Lexical Resources and Evaluation*, volume 2830, Granada, Spain, May 1998.

[30] P. Hayes. RDF Semantics. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[31] J. Hefflin. OWL Web Ontology Language Use Cases and Requirements. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[32] M. Hepp. Representing the hierarchy of industrial taxonomies in OWL: The gen/tax approach. In *Proceedings ISWC Workshop on Semantic Web Case Studies and Best Practices for eBusiness (SWCASE05)*, 2005.

[33] M. Hepp and J. de Bruijn. Gen tax: A generic methodology for deriving OWL and RDF-S ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies. In *ESWC*, 2007.

[34] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[35] H. Knublauch, M. A. Musen, and A. L. Rector. Editing description logic ontologies with the Protégé OWL plugin. In *Description Logics*, 2004.

[36] F. Manola and M. Miller. RDF Primer. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[37] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[38] G. Miller. *WordNet: An electronic Lexical Database*. MIT Press, 1998.

[39] A. Newell. The knowledge level. *Artificial Intelligence*, (18(1)):87–127, 1982.

[40] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Guide Semantics and Abstract Syntax. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[41] H. S. Pinto and J. P. Martins. A methodology for ontology integration. pages 131–138, 2001.

[42] H. S. Pinto, S. Staab, and C. Tempich. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In *ECAI*, pages 393–397, 2004.

[43] A. Rector and C. Welty. Simple part-whole relations in OWL Ontologies. Technical report, World Wide Web Consortium (W3C), August 11 2005. Working Draft.

[44] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *EKAW*, pages 63–81, 2004.

[45] P. Shvaiko, A. Ivanyukovich, L. Vaccari, V. Maltese, and F. Farazi. A semantic geo-catalogue implementation for a regional sdi. In *INSPIRE Conference*, Krakow, Poland, 2010.

[46] M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[47] D. Soergel, B. Lauser, A. Liang, F. Fisseha, J. Keizer, and S. Katz. Reengineering thesauri for new applications: The agrovoc example. *J. Digit. Inf.*, 4(4), 2004.

[48] M. van Assem, A. Gangemi, and G. Schreiber. RDF/OWL representation of WordNet, W3C working draft, June 2006.

[49] M. van Assem, M. R. Menken, G. Schreiber, J. Wielemaker, and B. J. Wielinga. A method for converting thesauri to RDF/OWL. In *International Semantic Web Conference*, pages 17–31, 2004.

[50] R. Vorz, J. Kleb, and W. Mueller. Towards ontology-based disambiguation of geographical identifiers. In *Proc. of the 16th WWW Conference*, 2007.

[51] I. Zaihrayeu, L. Sun, F. Giunchiglia, W. Pan, Q. Ju, M. Chi, and X. Huang. From web directories to ontologies: Natural language processing challenges. In *ISWC/ASWC*, pages 623–636, 2007.