Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia

Department of Computer Science

Creation and Extension of Ontologies for Describing Communications in the Context of Organizations

by Ygor Couto Cardoso

Thesis submitted to Faculdade de Ciências e Tecnologia of the Universidade Nova de Lisboa, in partial fulfillment of the requirements for the degree of **Master in Computer Science**

Supervisors:

PhD João Moura Pires

PhD José Júlio Alferes

Monte da Caparica, July 2010

Resumo

O recurso a ontologias é, actualmente, uma área suficientemente sólida e madura para ser considerada uma alternativa eficiente na representação da informação e conhecimento. Com o advento da Web Semântica é expectável que essa alternativa venha a disseminar-se ainda mais num futuro próximo.

No contexto de uma colaboração encetada entre a FCT-UNL e o departamento de I&D de uma empresa de *software* nacional, foi desenvolvida uma nova solução de *software* intitulada ECC – Enterprise Communications Center, que visa a estruturação do processo de comunicação de uma organização de e para o exterior, e emprega processos de classificação automática e de pesquisa conceptual com base num repositório de comunicações. A especificidade é a chave da obtenção de resultados aceitáveis por parte destes processos e, como tal, o uso de ontologias torna-se fundamental na representação do conhecimento existente acerca de um determinado domínio de uma organização.

Este trabalho permitiu dotar esta aplicação de um conjunto de ontologias base que possuem a capacidade de exprimir o contexto geral das comunicações efectuadas no âmbito de uma organização, e de uma metodologia constituída por um conjunto de passos concretos que permitem uma eficaz extensibilidade destas ontologias a qualquer domínio de actividade. Através da aplicação destes passos, garante-se a minimização do esforço de conceptualização e *setup* da plataforma em novas organizações e domínios de actividade.

A adequação do conjunto de ontologias escolhidas e da metodologia de extensão especificada é demonstrada nesta tese através da sua aplicação efectiva a um caso de estudo real, que permite tomar contacto com os diversos tipos de fontes de conhecimento que são consideradas na metodologia e com todo o ciclo de actividades que suportam a sua construção e evolução.

Abstract

The use of ontologies is nowadays a sufficiently mature and solid field of work to be considered an efficient alternative in knowledge representation. With the crescent growth of the Semantic Web, it is expectable that this alternative tends to emerge even more in the near future.

In the context of a collaboration established between FCT-UNL and the R&D department of a national software company, a new solution entitled ECC – Enterprise Communications Center was developed. This application provides a solution to manage the communications that enter, leave or are made within an organization, and includes intelligent classification of communications and conceptual search techniques in a communications repository. As specificity may be the key to obtain acceptable results with these processes, the use of ontologies becomes crucial to represent the existing knowledge about the specific domain of an organization.

This work allowed us to guarantee a core set of ontologies that have the power of expressing the general context of the communications made in an organization, and of a methodology based upon a series of concrete steps that provides an effective capability of extending the ontologies to any business domain. By applying these steps, the minimization of the conceptualization and setup effort in new organizations and business domains is guaranteed.

The adequacy of the core set of ontologies chosen and of the methodology specified is demonstrated in this thesis by its effective application to a real case-study, which allowed us to work with the different types of sources considered in the methodology and the activities that support its construction and evolution.

Agradecimentos

À minha família, *aos meus amigos, discos e livros*. Às restantes fontes de inspiração.

À extrema paciência e compreensão revelada pela minha futura esposa, Irina Oliveira, e a imensurável força que só ela consegue me transmitir, dia após dia.

Aos meus orientadores João Moura Pires e José Júlio Alferes, de cujo gosto pelo ensino, polivalência e competência beneficiei durante estes findos dois anos de mestrado, e especialmente nesta fase final. É com orgulho e satisfação que posso dizer que fui orientado por estes dois professores, e constatar o imenso que aprendi.

Agradeço e congratulo toda a equipa da ITDS, pela excelente oportunidade de desenvolver o nosso trabalho académico num aliciante projecto do "mundo real". Agradeço em especial ao nosso elemento de ligação entre a ITDS e a faculdade, Rui Leal, pelo apoio e pela disponibilidade demonstrada.

Aos meus colegas de curso e de projecto, Ricardo Neves e Bernardo Oliveira, pelo indispensável companheirismo evidenciado diariamente em todas as fases desta jornada. Uma nota especial e carregada de saudosismo aos restantes colegas de curso, em especial à família Bonga.

Index

List of Figures

List of Tables

Chapter 1
Introduction

This chapter presents the motivation, context and goals of this thesis, followed by the thesis structure

.

This chapter introduces the motivations behind the elaboration of this dissertation, namely the study of methodologies of creation and extension of ontologies. It is also shown the context of whereas this thesis has emerged, which is related to a new software solution, the ECC – Enterprise Communications Center, by the Portuguese company ITDS.

1.1. Context

ITDS[1] is a Portuguese software company that has designed and developed the XEO Platform. XEO stands for eXtensible Enterprise Objects and is a platform for business modeling and agile applications development that allows professional teams or business analysts to develop and maintain complex business applications, by modeling their real world's business requirements in the form of "business objects".

One of the main components of this platform is called XEO Outcom. Outcom is basically a component that allows the management and structuring of the communication process of organizations, in terms of communications channels management, definition of communication templates and integration with a documental repository.

Inspired on the XEO Outcom, ITDS has decided to develop a whole new product called XEO ECC – Enterprise Communications Center, which has its general functionality illustrated in figure 1.1. This product is expected to provide an effective solution to manage and centralize all the communications that enter, leave or are made within an organization, as well as analyze and deal with the content they express.

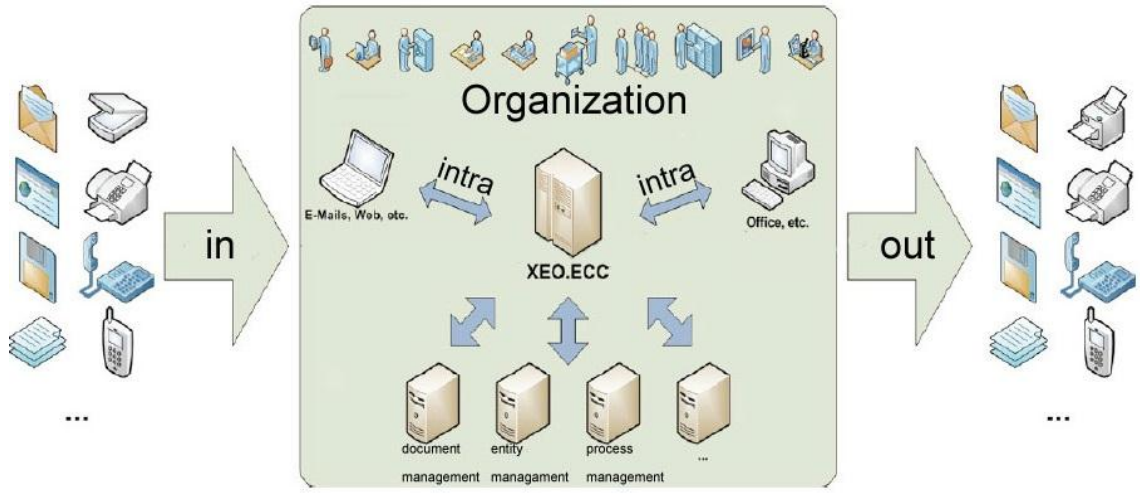

Figure 1.1 XEO ECC General Functionality

[1] www.itds.pt

[2] http://dublincore.org/

Besides managing the organization's communications, this program includes intelligent classification of communications and search and navigation in the communications repository. To support these processes, the company has decided to rely on the use of ontologies.

The main goal is to provide an effective solution for a company to centralize and manage all the communications that enter or leave the organization, independently of the content expressed in the communications.

The ontologies are expected to model and express the communications, their interlocutors and respective roles within the organization, and the organization specific domain context.

Based on that model, the communications are annotated and classified according to the purpose illustrated by the communication content; this process serves as a basis to forwarding communications to each respective department in the organization, for example.

The annotated communications are made available in an archive, and there will be conceived a specific interface to search and navigate through it.

To support these functionalities, it is essential that the ontologies are able to be easily adaptable and extensible to any domain, as this guarantees that the effort of the setup process of the product in different customers is minimized, and the analysis of the communications content will be optimized for every particular case. This requires that a specific core set of ontologies is designed in such a way that its capacity of extensibility is guaranteed, and a comprehensive methodology of achieving that extension is described. Figure 1.2 illustrates the scenario described.

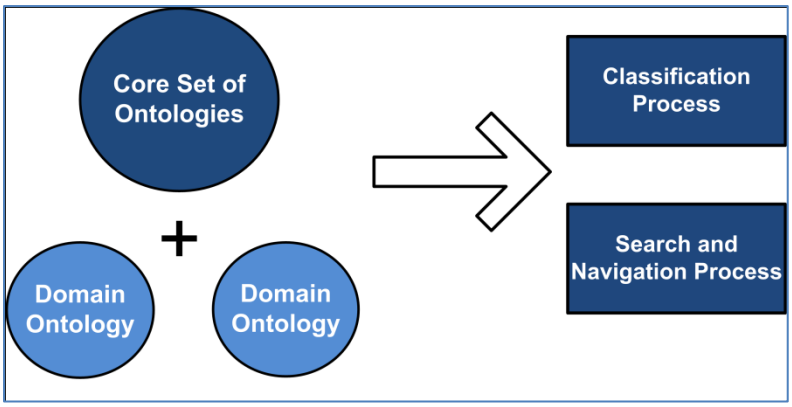

Figure 1.2 Ontologies and their usage on the project

ITDS has obtained a QREN financial incentive to this project, and made an agreement with Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa to associate three MSc students to

it, with the theses themes of ontology development, automatic classification and search and navigation. This thesis is based on the first theme, and its aim is to create a methodology to develop and maintain the core set of ontologies the other processes rely upon.

1.2. Motivation

Ontology Engineering refers to the set of activities that concern the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them.

During the last few years, increasing attention has been paid to ontologies and Ontological Engineering, since they are now widely used in Knowledge Engineering, Artificial Intelligence and Computer Science, in general. It is now widely recognized that constructing a domain model, or ontology, is an important step in the development of knowledge based systems.

As such, one of the things that motivates me the most is the opportunity to deal with such an emerging and fascinating field of study, and to essentially understand the major issues involved in the ontologies development process; the other factor was the possibility of integrating this work into a motivating project, which allied to the company interest in maintaining and supporting its elements academic studies led me to believe that this work can potentially involve future applied research on this field.

1.3. Objective

The objective of this thesis is to propose a core set of ontologies and vocabularies that support the description of the organizations communications, as well as their internal structure, products and services they offer. Based on this core set, a methodology constituted by concrete steps is proposed in order to allow the extension of the ontologies to any specific domain.

1.4. Achievements

Prior to this work, the company had no knowledge-aware applications of any kind, and never had to address these kind of issues on their work. As such, it was with great satisfaction that we verified that this philosophy was well accepted and effectively implemented through this platform.

The core set of ontologies became an effective solution for representing the intended knowledge, and proved to be suitable and adaptable to any specific business domain; on the other hand, the methodology developed, besides serving as a guideline to effectively use the model, is already being used as a reference for consultants trained by ITDS for this matter. As such, this work serves as a successful response for the problems addressed.

1.5. Thesis Structure

The content of this thesis is structured in six chapters:

Chapter one (Introduction) presents the motivation, context and objectives of this thesis, followed by the thesis structure.

In chapter two (State of the Art) it is explained what is an ontology in this context, what are its main applications (with special emphasis on the Semantic Web), and some examples of existing ontologies are given. Further on, it is also explained what constitutes the life cycle of an ontology and all of its development process; this explanation serves as a basis for detailing some of the existing methodologies on the Ontology Engineering field, and to discuss the problem of re-engineering non-ontological resources into ontologies, with the particular cases of translating relational model schemas into RDF and using textual resources to extract ontology concepts.

Chapter three (Proposed Solution) presents the proposed solution for the core set of ontologies and the methodology developed to extend them to any specific domain. In chapter four (Application), these ontologies are exemplified and extended through a specific case of study.

In chapter five (Evaluation), an evaluation of this work is made in terms of satisfaction of requirements, consistency and response to future changes.

The thesis is concluded in chapter six (Conclusion and Future Work) where future work is presented and conclusions are drawn on the design and implementation of the core set of ontologies and the methodology. The annex is only present in the digital copy of this thesis, and includes the full RDF code of the ontologies designed.

Chapter 2
State of the Art

This chapter presents the state of the art in the ontology engineering field, including methodologies and tools to create, extend and maintain ontologies.

This chapter is devoted to presenting what is an ontology in this context, what are its main applications, and to specify and analyze some methodologies that we have at our disposal to create and extend ontologies.

The first section (Ontologies) introduces some of the theoretical foundations of the ontology field, and presents vocabularies, main modeling components, design criteria and their suggested practical usage. This serves as a basis for the design of the **core set of ontologies** presented in chapter 3 of this document.

The second section (Ontologies Repositories and Tools) presents some tools and platforms that are used to build ontologies and that allow their usage on the Semantic Web. The usage of these tools is further exemplified in the **implementation and application phases** of this work, which are described on chapters 3 and 4 of this document.

The third section (Methodologies for Building Ontologies) discusses the ontology development process and the existing methods and methodologies that support the ontology construction from scratch, identifying and discussing specific activities that may emerge during the development process. These methodologies and activities were taken into consideration to develop the **methodology** that is presented in chapter 3 of this document.

The fourth section (Re-engineering Non-Ontological Resources into Ontologies) deals with the particular field of the ontology engineering area which is related with reusing and re-engineering knowledge-aware resources for building ontologies, rather than building them completely from scratch. The aim of this section is to list and compare methods, techniques, and tools for reusing and re-engineering the terminology contained in the available knowledge-aware resources. This serves as a basis to identify what kind of resources can be gathered in the **knowledge elicitation phase** of the methodology, to minimize the effort of extending our core set of ontologies to specific domains, as illustrated in chapter 3 of this document.

Finally, the last two sections (Relational Databases to RDF and Ontology Learning from Unstructured Text) present some research about two particular fields of ontology learning, which are the transformation of relational databases to RDF, and the ontology learning from the analysis of unstructured corpus of text. Although they are not effectively present in the implementation of this work, they were analyzed and may serve as a reference for **future work** suggested in the last chapter of this document.

2.1. Ontologies

Ontologies are the main subject of this work. This section is devoted to presenting what does the concept of ontology mean in this context and where is this concept applied (section 2.1.1). In the second section (section 2.1.2), some examples of existing ontologies and vocabularies that can be used in this project are presented, as described in chapter 3.

2.1.1. What Is an Ontology

The term ontology originates from philosophy. In that context, it is used as the name of a subfield of philosophy, namely, the study of the nature of existence (the literal translation of the Greek word Οντολογία), the branch of metaphysics concerned with identifying, in the most general terms, the kinds of things that actually exist, and how to describe them [85]. For example, the observation that the world is made up of specific objects that can be grouped into abstract classes based on shared properties is a typical ontological commitment.

However, in more recent years, ontology has become one of the many words hijacked by computer science and given a specific technical meaning that is rather different from the original one. Instead of "ontology" we now speak of "an ontology", in this context.

One of the first definitions of what an ontology is was given by Neches et al [60], who defined it as follows: "an ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary." This descriptive definition tells what to do in order to build an ontology, and gives us some vague guidelines: the definition identifies basic terms and relations between terms, identifies rules to combine terms, and provides the definitions of such terms and relations. Note that, according to Neches's definition, an ontology includes not only the terms that are explicitly defined in it, but also the knowledge that can be inferred from it.

A few years later, in 1993, Gruber [34] stated that "An Ontology is an explicit specification of a conceptualization". This became the most quoted definition in the literature and by the ontology community. Based on Gruber's definition, many definitions of what an ontology is were proposed; for this work, we use Studer's [72] refinement of Gruber's definition: "An ontology is an explicit and formal specification of a conceptualization".

In general, an ontology describes formally a domain of discourse. Typically, an ontology consists of a finite list of terms and the relationships between these terms. The terms denote important concepts (classes of objects) of the domain. For example, in a university setting, staff members,

students, courses, lecture theaters, and disciplines are some important concepts, as depicted in figure 2.1.

The relationships typically include hierarchies of classes. A hierarchy specifies a class C to be a subclass of another class C' if every object in C is also included in C'. For example, all faculty are staff members. Figure 2.1 shows a hierarchy for the university domain.

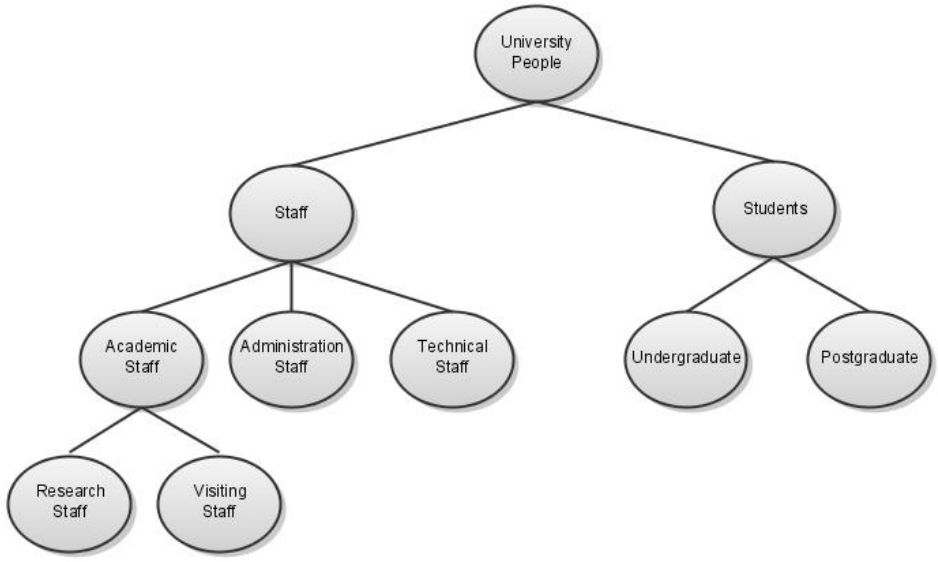

Figure 2.1 A Hierarchy – adapted from [85]

Apart from subclass relationships, ontologies may include information such as:

- Properties (X teaches Y);
- Value restrictions (only faculty members may teach courses);
- Disjointness statements (students and staff are disjoint);
- Specifications of logical relationships between objects (every department must include at least ten faculty members).

2.1.2. Ontologies Languages and Vocabularies

At present, the most important ontology languages for the Web are the following:

- RDF is a data model for objects ("resources") and relations between them; it provides a simple semantics for this data model; and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary description language for describing properties and classes of RDF resources, with a semantics for generalization of hierarchies of such properties and classes.
- OWL is a richer vocabulary description language for describing properties and classes, such as relations between classes (e.g., disjointness), cardinality (e.g., "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes.

RDF

The Resource Description Framework [92] (RDF) is a W3C Recommendation to describe resources through statements. In RDF, a statement (or triple) is composed of three parts: A **resource**, a **property** and a **property value**. In the case where the statement is referred to as a triple, the component names are, respectively, subject, predicate and object.

- A **resource** identifies an object through a URI (in RDF, any object is identified with a URI).
- A **property** describes an attribute of a resource identified by a URI.
- A **property value**, is the value a given property has. This value can be another resource, if a URI is used.

As an example, to describe the title of the Martin Scorcese movie based on the novel by Dennis Lehane, “Shutter Island”, identifying the movie with the URI “http://www.shutterisland.com/#/home” and the title property by the URI of the Dublin Core element *title*, “http://purl.org/dc/elements/1.1/title”, the following statement can be defined:

Resource (Subject): http://www.shutterisland.com/#/home

Property (Predicate): http://purl.org/dc/elements/1.1/title

Property Value (Object): Shutter Island

A RDF document contains a set of RDF statements. To define RDF documents, several syntaxes are available [93], one of them being the RDF/XML syntax [94]. This syntax defines the structure of a RDF document as follows:

A root element “rdf” that must include the namespace declaration with the “rdf” prefix, mapped to the address “http://w3.org/TR/1999/PR-rdf-syntax-19990105#”. Children of this node can only be elements of type “**Description**”.

As an example, the previous statement is depicted in Figure 2.2, in a RDF/XML serialization. It also includes some more information of the movie, including the movie director with the predicate “**DC Creator**” and a film contributor with the predicate “**DC Contributor**” and defining them as references to other elements (using the attribute **rdf:resource**) inside the same document.

In RDF, each predicate **must** be associated to a namespace [95], to make it possible to resolve ambiguities between predicates with the same name, but defined by different entities, and also to know where to search for the meaning of any given predicate.

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:id="http://www.example.com/identifier">

    <rdf:Description rdf:about="http://www.shutterisland.com/#/home">
        <dc:title>Shutter Island</dc:title>
        <dc:creator rdf:resource="http://www.imdb.com/name/nm0000217/"/>
        <dc:contributor rdf:resource="http://www.imdb.com/name/nm0436164/"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.imdb.com/name/nm0000217/">
        <id:name>Martin Scorcese</id:name>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.imdb.com/name/nm0436164/">
        <id:name>Laeta Kalogridis</id:name>
        <dc:description>WGA Writer</dc:description>
    </rdf:Description>

</rdf:RDF>
```

Figure 2.2 RDF Example

Besides being designed as a data model to represent information about resources and to allow that information to be exchanged and processed among applications, RDF was also conceived as an appropriate representation formalism allowing for logical inference. This allows us to extract computationally logical consequences from the provided information.

RDF Schema

RDF Schema [97] is a W3C recommendation for the construction of RDF based vocabularies. Essentially, it allows for the definition of classes, subclasses and properties of those classes, in a very similar way to most Object Oriented (OO) programming languages, such as Java. As such, it allows for the creation of hierarchies of classes for the description of “objects”.

Just to state some of the elementary components of RDF and RDF Schema: in section 2.1.1, it is said that an ontology can have classes; these classes are resources denoting a set of resources, and

this can be specified in RDF by the mean of the property rdf:type (instances have property rdf:type valued by the class). Those sets of resources have by definition the property rdf:type valued by rdfs:Class. On the other hand, all properties (defined in the W3C recommendation or in any schema) have rdf:type valued by rdf:Property. At last, hierarchies of classes can be denoted by the property rdfs:subClassOf.

OWL

The Web Ontology Language (OWL) [99] is a family of knowledge-representation languages to create ontologies. It was designed for usage by applications that need to process the content of information, instead of just presenting it to people. OWL promotes a better interoperability of web content, between machines, than what is supported by formats such as XML, RDF and RDF Schema, supplying an extended vocabulary as well as a formal semantics, allowing for the development of formal ontologies. In OWL we can express, for example, equality relationships between classes (owl:sameAs or owl:equilaventClass), class property restrictions (owl:allValuesFrom or owl:someValuesFrom) or richer properties between classes (owl:transitive, owl:symmetric or owl:inverseOf).

OWL is composed by three sub-languages, progressively more expressive: OWL Lite, OWL DL and OWL Full [85].

OWL Full allows for full expressiveness and syntactic freedom of RDF, using all the OWL language primitives. It also allows for the combination of these primitives in arbitrary ways with RDF and RDF Schema. This includes the possibility (also present in RDF) of changing the meaning of the predefined (RDF or OWL) primitives by applying the language primitives to each other. For example, in OWL Full, we can impose a cardinality constraint on the class of all classes, essentially limiting the number of classes that can be described in any ontology. The advantage of OWL Full is that it is fully upward-compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion. The disadvantage of OWL Full is that the language is so powerful that become undecidable, giving thus no guarantee at all about the end of computation.

In order to regain computational efficiency, OWL DL (short for Description Logic) is a sublanguage of OWL Full that restricts how the constructors from OWL and RDF may be used. Essentially, the application of OWL's constructors to each other is disallowed, thus ensuring that the language corresponds to a well-studied description logic. The advantage of this is that it permits efficient reasoning support. The disadvantage is that we lose full compatibility with RDF. An RDF document

will in general have to be extended in some ways and restricted in others before it is a legal OWL DL document. Every legal OWL DL document is a legal RDF document.

Finally, an even further restriction limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality. The advantage of this is a language that is both easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is, of course, a restricted expressivity.

Dublin Core

The Dublin Core Metadata Initiative (DCMI)[2] is an organization dedicated to promoting the adoption of interoperability standards of metadata description of multimedia content such as documents, images, videos, websites, etc., having specifications of abstract models of description and a specific vocabulary scheme.

The Dublin Core standard includes two levels: Simple and Qualified. Simple Dublin Core comprises fifteen elements; Qualified Dublin Core includes three additional elements (Audience, Provenance and RightsHolder), as well as a group of element refinements (also called qualifiers) that refine the semantics of the elements in ways that may be useful in resource discovery.

The Simple Dublin Core Metadata Element Set (DCMES) consists of 15 metadata elements: title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage and rights. Each Dublin Core element is optional and repeatable. There is no prescribed order in Dublin Core for presenting or using the elements. That grants some flexibility to the use we may give to the model.

The usage of these terms is very popular and nearly standardized for metadata descriptions, and implementations of Dublin Core typically make use of XML and are RDF based.

Simple Knowledge Organization System

Simple Knowledge Organization System (SKOS)[3] is a family of formal languages designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary, relying on RDF and RDF Schema. SKOS has been designed to provide a low-cost migration path for porting existing organization systems to the Semantic Web. SKOS also provides a lightweight, intuitive conceptual modeling language for developing and sharing new KOS's. It can be used on its own, or in combination with more-formal languages such as the Web

[2] http://dublincore.org/
[3] http://www.w3.org/2004/02/skos/

Ontology Language (OWL). SKOS can also be seen as a bridging technology, providing the missing link between the rigorous logical formalism of ontology languages such as OWL and the chaotic, informal and weakly-structured world of Web-based collaboration tools, as exemplified by social tagging applications.

The aim of SKOS is not to replace the original conceptual vocabularies in their initial context of use, but to allow them to be ported to a shared space, based on a simplified model, enabling wider re-use and better interoperability.

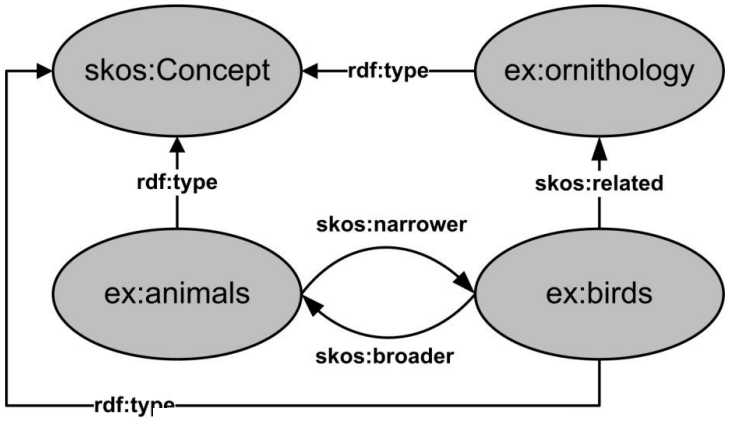

Figure 2.3 SKOS modelling example

In KOS semantic relations play a crucial role for defining concepts. The meaning of a concept is defined not just by the natural-language words in its labels, but also by its links to other concepts in the vocabulary. Mirroring the fundamental categories of relations that are used in vocabularies such as thesaurus, SKOS supplies three standard properties:

- skos:broader and skos:narrower enable the representation of hierarchical links, such as the relationship between one genre and its more specific species, or, depending on interpretations, the relationship between one whole and its parts; an example is illustrated in figure 2.3, with the relation of the concepts “birds” and “animals”.
- skos:related enables the representation of associative (non-hierarchical) links, such as the relationship between one type of event and a category of entities which typically participate in it. Another use for skos:related is between two categories where neither is more general or more specific; an example is illustrated in figure 2.3, with the relation with the concepts “birds” and “ornithology”.

SKOS has become a W3C Recommendation since 18th August 2009, being currently developed by the W3C Semantic Web Deployment Working Group.

The Open Source Business Management Ontology (BMO)

The Business Management Ontology (BMO) is a framework developed by Jenz & Partner[4] in order to provide a means for the semantically rich definition of business processes in a vendor and product neutral format. Separation of concerns has been one of the major factors in the design of the BMO architecture. The emphasis rests on the ability to combine ontologies in a flexible fashion, and ownership of each ontology may be individually assigned.

The generic business domain ontologies allow for the reuse of concepts in different contexts. For example, the "Role" concept is needed in the business process context (tasks are performed by roles) and in the organization-specific context, and potentially in one or more industry-specific context. In essence, the BMO architecture greatly enhances flexibility and facilitates reuse.

While, in principle, an ontology may contain definitions and data (instances), the BMO encourages the separation of model and data. A higher-level ontology may import one or more lower-level ontologies. The separation also prevents for unauthorized modification of ontologies, meaning that a user cannot modify imported ontologies.

At present, the BMO encompasses around 40 ontologies. Table 2.1 briefly describes the major BMO building blocks.

Ontology	Comments
BMO	Contains only instances, no definitions.
Business Integration	Represents the business integration level, which links ontologies together to form a "one-world-view".
Organization-specific	Integrates and extends generic business domain ontologies. For example, organization unit types and roles that are specific to the organization can be defined.
Industry-specific	Multiple industry-specific ontologies may exist in parallel, which caters for multi-industry organizations. Like the organization-specific ontology, each industry-specific ontology integrates and extends generic business domain ontologies. For example, industry-specific concepts and product taxonomies can be defined.
BPM-Integration	Forms the business process management integration layer. It provides associations of core business process concepts with generic business concepts, such as roles, business rules, business documents, etc.
IT-Integration	Represents the IT-Integration layer, which links IT-related (technical) ontologies together.
Generic business domain ontologies	Represents generic business concepts, such as - roles - business rules - business documents - durable information entities - core components (basic business information structures) - resources
BPM (core)	Contains business process-specific concepts required to model process flows for internal (private), public and collaborative business processes.
Business context integration	Forms the basic integration layer.
Business context	Allows for the definition of semantic and speech communities, and defines basic concepts, such as "universe of discourse".
Code lists	Contains code lists that are needed in almost every organization. Examples: countries, languages, currencies.
IT	Contains "technically-oriented" ontologies, such as the business object ontology.

Table 2.1 BMO Ontologies

[4] http://www.jenzundpartner.de/

SPARQL

SPARQL [98] is a RDF document query language designed by the W3C, that since January 2008 is a W3C Recommendation; its name is a recursive acronym that stands for ***S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage*. SPARQL allows for querying multiple sources of information, whether that information is natively in RDF or is supplied by some middleware. The results of a query can be a RDF graph or a set of statements.

Most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. A basic graph pattern matches a sub-graph of the RDF data when RDF terms from that sub-graph may be substituted for the variables and the result is RDF graph equivalent to the sub-graph. SPARQL also allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

The example on figure 2.4 shows a simple SPARQL query to find the title of a book from the given data graph. The query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (?title) in the object position.

Data:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Query Result:

title
"SPARQL Tutorial"

Figure 2.4 - SPARQL Query

SeRQL

SeRQL [100] stands for Sesame RDF Query Language and is a querying and transformation language loosely based on several existing languages, most notably RQL, RDQL and N3. Its primary design goals are unification of best practices from query language and delivering a light-weight yet expressive query language for RDF that addresses practical concerns.

SeRQL syntax is similar to that of RQL though modifications obtain an easier to parse language. Like RQL, SeRQL is based on a formal interpretation of the RDF graph, but SeRQL's formal interpretation is based directly on the RDF Model Theory.

SeRQL supports generalized path expressions, boolean constraints and optional matching, as well as two basic filters: select-from-where and construct-fromwhere. The first returns the familiar variable-binding/table result; the second returns a matching (optionally transformed) subgraph.

SeRQL is implemented and available in the Sesame system, which has been used to accommodate the ontologies developed in this thesis.

2.2. Ontologies Repositories and Tools

This section presents some tools and platforms that are used to build and maintain ontologies. The usage of these tools is further exemplified in the implementation and application phases of this work, as described on chapters 3 and 4 of this document.

2.2.1. Sesame

Sesame [101] is an open source java framework for storing, querying and reasoning with RDF and RDF schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally.

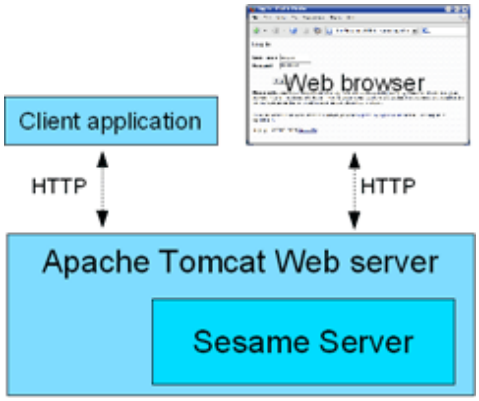

Figure 2.5 Sesame Server

Sesame can be used as a Server in which client applications (or human users) can communicate over HTTP, as illustrated in figure 2.5. Sesame can be deployed as a Java Servlet Application in Apache Tomcat, a webserver that supports Java Servlets and JSP technology.

A central concept in the Sesame framework is the repository. A repository is a storage container for RDF. This can simply mean a Java object (or set of Java objects) in memory, or it can mean a relational database. However, whatever way of storage is chosen, it is important to realize that

almost every operation in Sesame happens with respect to a repository: when you add RDF data, you add it to a repository. When you do a query, you query a particular repository.

Sesame, as mentioned, supports RDF Schema inferencing. This means that given a set of RDF and/or RDF Schema, Sesame can find the implicit information in the data. Sesame supports this simply by adding all implicit information to the repository as well, when data is being added.

The framework includes a Web interface (openrdf-workbench), in order to allow access to repositories through a web browser. In this work, this interface was used to create the repository and to load and query the knowledge edited in Protégé.

When we create a Sesame repository, there are nine template choices for the repository configuration:

- Memory: a memory based RDF repository
- Memory-rdfs: a main-memory repository with RDF Schema inferencing
- memory-rdfs-dt: a main-memory repository with RDF Schema and direct type hierarchy inferencing
- Native Java Store: a repository that uses on-disk data structure
- Native-rdfs Java Store: a native repository with RDF Schema inferencing
- Native-rdfs-dt Java Store: a native repository with RDF Schema and direct type hierarchy inferencing
- Postgre sql: a repository that stores data in a PostgreSQL database
- MySql: a repository that stores data in a MySQL database
- Remote RDF Store: a repository that serves as a proxy for a repository on a Sesame Server

On this work, as illustrated in section 4.5, a native-rdf java store repository was used. A memory based repository would not fit because of the large amount of data expected, and we did not felt the need to use an external database system.

2.2.2. Protégé

Protégé [102] is an open source tool for creating domain models and knowledge-oriented applications through ontologies, developed by the Stanford Center for Biomedical Informatics Research of the Stanford University School of Medicine. The core of Protégé, implements a rich set of knowledge representation structures and actions that allows for creating, managing and visualizing ontologies in several formats.

It has a customizable interface, so that a simple graphic application can be produced for the creation of ontologies (and populating them with information). Protégé can be extended with plugins and a Java API exists so that applications on top of (or connected to) protégé can be built [103].

The protégé platform is available as two products. **Protégé-Frames** and **Protégé-OWL**, explained in the subsections below. Additionally, it is also presented a specific plugin for Protégé version 4, the SKOSed plugin.

Protégé-Frames

The Protégé-Frames editor provides the tools to easily create an ontology for any domain, as well as its maintenance and data input. Protégé implements a knowledge model that is compatible with the Open Knowledge Base Connectivity protocol (OKBC) [104]. In this model, an ontology is seen as a set of classes that can be organized in a hierarchy (to represent domain concepts in a very similar way as an object-oriented programming language does) and a number of "slots" can be associated to classes in order to describe its properties and relations. It is possible, after that step, to create instances of the ontology that are individual examples of classes that have specific (and distinct) property values.

The ontology creation process is eased by a simple graphic interface, as depicted in Figure 2.6. After creating non-abstract classes in an ontology, it's possible to create instances of those classes, an action Protégé eases by generating specific forms to create those instances (the forms can be further

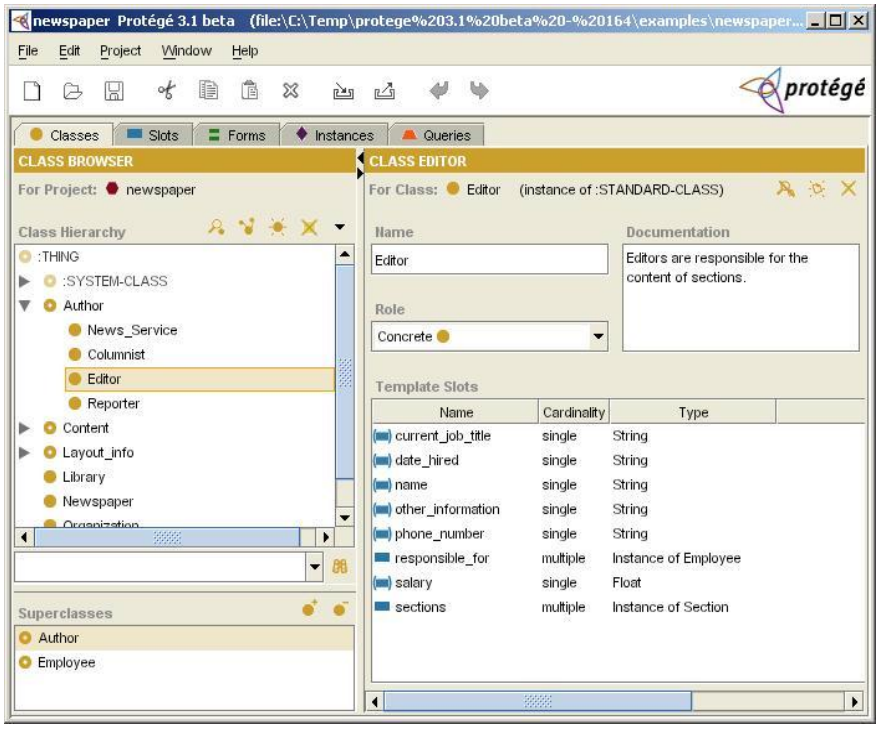

Figure 2.6 Protégé-Frames editor

customized to make it even easier for users to interact with them). During the creation phase of ontologies (or of class instances) all rules are verified, including cardinality rules, relationship rules or restrictions to values of properties and the generated interfaces already have those restrictions into account [105].

Protégé-Frames supply a graphical interface to query instances of the ontology, and allow making selections based on criteria over the values of the slots of a class. Figure 2.7 depicts the interface for querying, using “salary” as a criterion for the search.

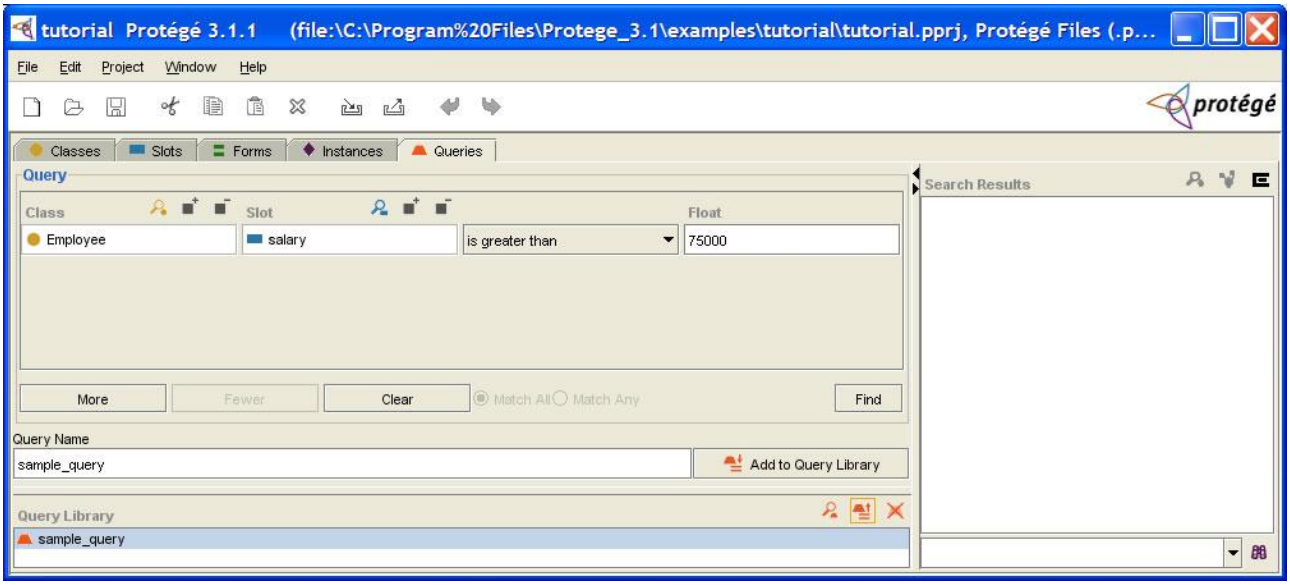

Figure 2.7 Protégé-Frames Query Interface

Protégé-OWL

Protégé-OWL is an extension of Protégé that enables the creation of ontologies based on the Web Ontology Language [106]. Beyond the possibility of creating classes (and restrictions, instances, etc.) offered by Protégé-Frames, Protégé-OWL allows users to create, or load, their ontologies in the RDF/OWL format, allows for defining the properties of classes (specific of OWL) as well as use inference engines to extract new knowledge not explicitly in instances, but achievable through the semantic rules of the ontology. It also supplies an equally simple interface (supported by several plugins) that eases the graphical inspection of an ontology (among many other actions) such as depicted in Figure 2.8.

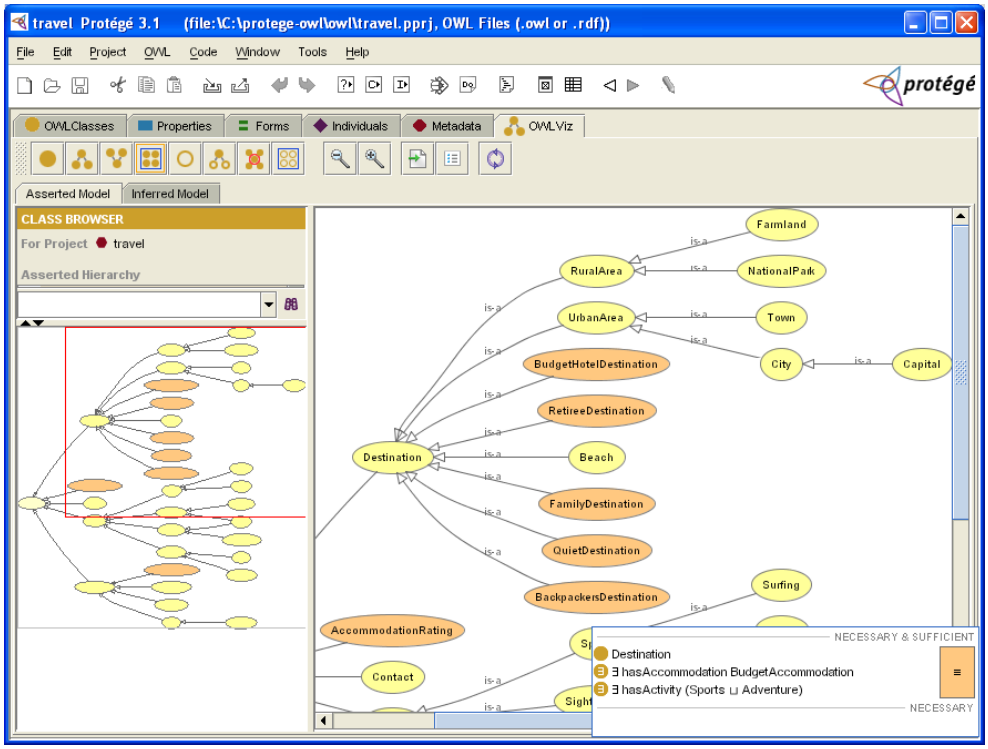

Figure 2.8 Protégé-OWL - Ontology Visualization Plugin - http://protege.stanford.edu/

SKOSed

SKOSEd is a plugin for Protégé 4 that allows the user to create and edit thesauri (or similar artifacts) represented in the Simple Knowledge Organization System (SKOS).

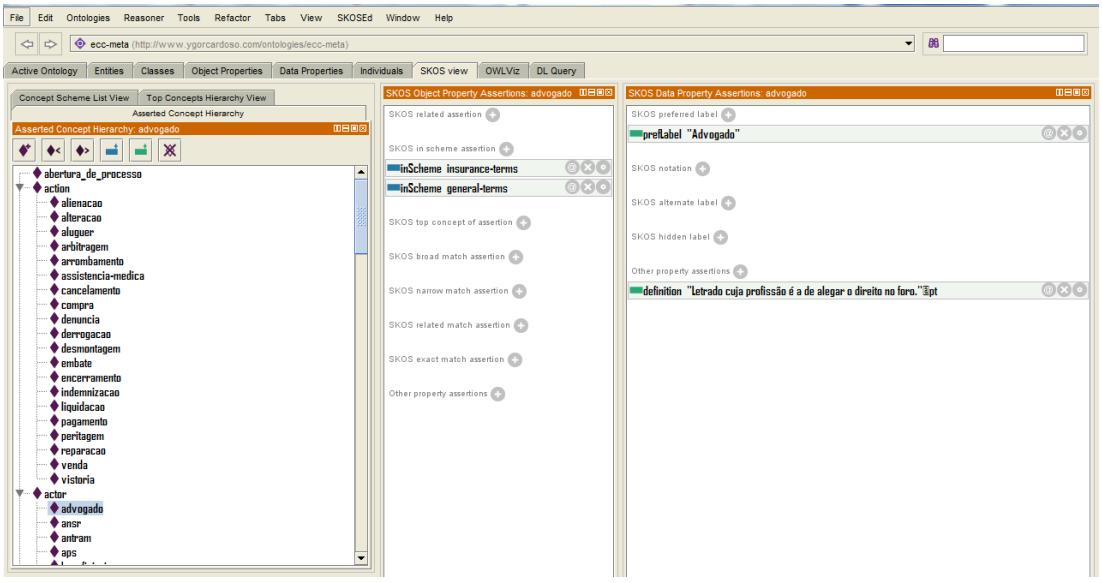

Figure 2.9 Protégé 4 - SKOSed plugin

This plugin provides some Protégé views and menu items that offer additional support to Protégé users wishing to work with SKOS. Central to the plugin is a SKOSEd tab, which is used to store the

main available views. The main view is the Asserted Hierarchy View, which is used to expose the SKOS broader/narrower hierarchy; this view also has drag-and-drop capabilities and a set of convenience buttons for manipulating the hierarchy, as depicted in figure 2.9.

2.3. Methodologies for Building Ontologies

The ontology development process, as identified in the framework of the METHONTOLOGY methodology [48] and based on the IEEE standard for software development [43], consists basically in identifying which activities should be performed when building ontologies. This definition is followed to describe and illustrate the details about these activities, as it not only allows us to identify each activity, but also to group them into general stages of the ontology life cycle. As such, this is also used to contextualize each of the methodologies presented in sections 2.3.2 to 2.3.5, and to elaborate some general conclusions on section 2.3.6.

2.3.1. Ontology Development Process

According to the previous definition, the activities of the ontology development process can be grouped into three different categories [32]: management activities (1), development oriented activities (2) and support activities (3). These activities are illustrated in figure 2.10 and described in detail as follows:

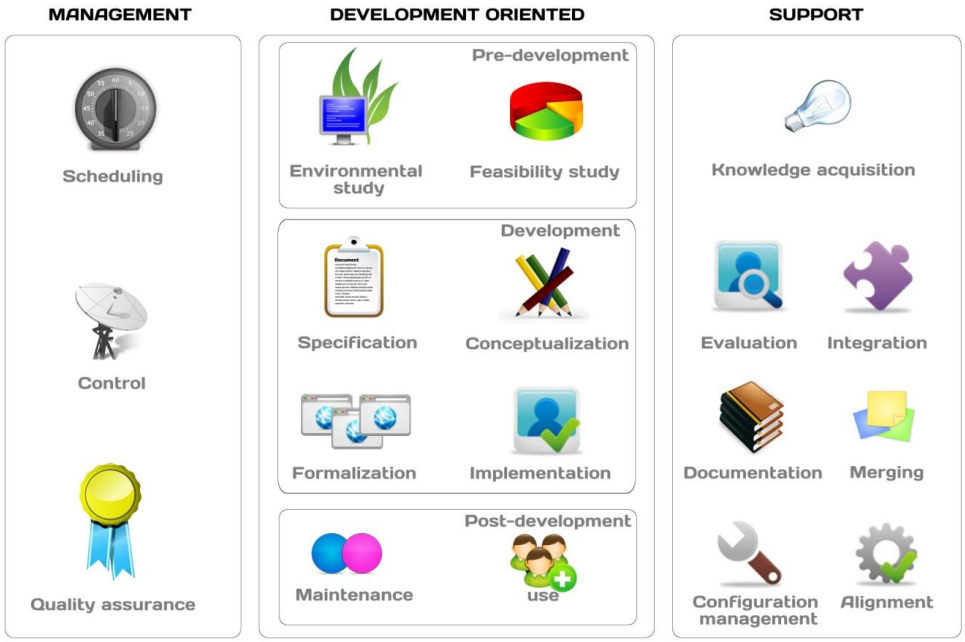

Figure 2.10 Ontology Development Process Activities - adapted from [32]

Ontology Management Activities

Ontology management activities include scheduling, control and quality assurance. The scheduling activity identifies which tasks must be performed, their arrangement, and the time and resources required for their completion. The control activity guarantees that scheduled tasks are completed in the intended manner to be performed. Finally, the quality assurance activity assures that the quality of each and every produced output (ontology, software and documentation) is satisfactory.

Ontology Development Oriented Activities

Ontology development oriented activities can be grouped into pre-development, development and post-development activities, as shown in figure 2.10.

During pre-development, an environmental study documents the ontology environment, regarding the platforms where the ontology will be used, the applications where the ontology will be integrated in, etc. Also during the pre-development activity, the feasibility study assesses whether it is possible and suitable to build the ontology.

Once in the development, the specification activity states the reasons why the ontology is being built, what are its intended uses and who the end-users are. The conceptualization activity structures the domain knowledge as meaningful models at the knowledge level [61]. The formalization activity transforms the conceptual model into a formal or semi-formal model. The implementation activity builds computable models in an ontology language.

During the post-development activity, the maintenance activity updates and revises the ontology, if required. Also during the post-development, the ontology may be (re)used by other ontologies or applications.

Ontology support activities

Finally, ontology support activities include a series of activities performed at the same time as the development-oriented activities, without which the ontology could not be built. They may include knowledge acquisition, evaluation, integration, merging, alignment, documentation and configuration management.

The goal of the knowledge acquisition activity is to acquire knowledge from experts of a given domain or through some kind of automatic or semi-automatic process, something called ontology learning [15].

The evaluation activity [14] makes a technical judgment of the ontologies, of their associated software environments, and of the documentation. This judgment is made with respect to a frame of reference during each stage and between stages of the ontology's life cycle.

The integration activity is required when building a new ontology by reusing other ontologies already available.

Another support activity is merging [62], which consists of obtaining a new ontology starting from several ontologies on the same domain. The merging of two or more ontologies can be carried out either in run-time or design time.

The alignment activity establishes different kinds of mappings (or links) between the involved ontologies. Hence, this option preserves the original ontologies and does not merge them.

The documentation activity details, clearly and exhaustively, each and every one of the completed stages and products generated.

The configuration management activity records all the versions of the documentation and of the ontology code to control the changes.

2.3.2. Uschold and King's method

The first method for building ontologies was proposed by Uschold and King in 1995 [80], and consisted of some guidelines and requirements based on their experience in developing the Enterprise Ontology. The Enterprise Ontology was developed as a part of the Enterprise Project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners IBM, Lloyd's Register, Logica UK Limited, and Unilever.

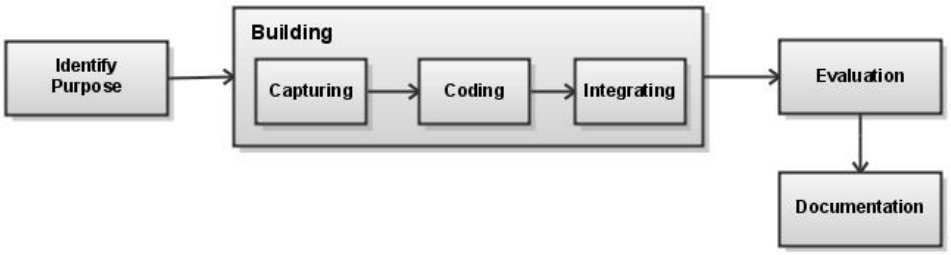

Figure 2.11 Main processes of the Uschold and King's Method [32]

According to Uschold and King's approach, the following stages must be carried out during the ontology development process: (1) identify the purpose of the ontology, (2) build the ontology, (3) evaluate the ontology, and (4) document the ontology. These processes, as shown in figure 2.11, are:

Identifying the purpose and the scope

The goals of this step are to clarify why the ontology is being built and what its intended uses are, the characterization of the range of intended users and the identification of the relevant terms on the domain. The output of this step is a specification that fully outlines the range of information that the ontology must characterize. This may be done by using motivating scenarios and informal competency questions, as in TOVE [90] or by "brainstorming and trimming", i.e. producing a list of potentially relevant concepts and deleting irrelevant entries and synonyms.

Building the ontology

This step breaks into three activities: capturing the ontology (1), coding it (2), and integrating it into existing ontologies (3). The authors have decided to put the main emphasis on the ontology capture phase.

Activity 1: Ontology Capture

In this context, ontology capture is intended as:

- identification of the key concepts and relationships in the domain of interest, i.e., scoping;
- production of precise unambiguous text definitions for such concepts and relationships;
- identification of terms to refer to such concepts and relationships;
- agreeing on all of the above;

To identify the concepts in the ontology, Uschold and Gruninger pointed out three strategies: bottom-up, top-down, and middle out.

The bottom-up strategy involves identifying first the more specific concepts of the domain and then generalize them into more abstract concepts. This approach results in a very high level of detail, but has the drawbacks of increasing the overall effort, the difficulty of spotting commonality between related concepts, and the risk of inconsistencies.

In the top-down approach, we start by identifying the most abstract concepts, and then specialize them into more specific concepts. Using this approach results in a better control of the level of detail, however, starting at the top can result in imposing arbitrary and possibly not needed high level categories.

The middle-out strategy recommends identifying first the core of basic terms, and then specifying and generalizing them properly as required. This strikes in a balance in terms of level of detail, arising

only when necessary, and avoiding some efforts. Starting first with core level concepts makes them appear naturally, and thus more likely to be stable.

Activity 2: Ontology Coding

In this context, coding is meant to be the explicit representation of the conceptualization captured in the above stage in some formal language; this involves committing to some meta-ontology, choosing a representation language and effectively writing the code.

The authors also discuss the possible merge of the capturing and coding activities into the same step. Ontologists, in some practical cases, may start to develop the conceptualization on the fly; they accept that this may be appropriate for some cases, but suggest that in general, it is more beneficial to separate the two.

To conclude the analysis of this activity, they indicate that the principles for designing ontologies enumerated by Gruber [86] should be present in any methodology. These principles are the following:

- Clarity: An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective. While the motivation for defining a concept might arise from social situations or computational requirements, the definition should be independent of social or computational context. A formalism is a means to this end. When a definition can be stated in logical axioms, it should be. Where possible, a complete definition (a predicate defined by necessary and sufficient conditions) is preferred over a partial definition (defined by only necessary or sufficient conditions). All definitions should be documented with natural language.
- Coherence: An ontology should be coherent: that is, it should sanction inferences that are consistent with the definitions. At least, the defining axioms should be logically consistent. Coherence should also apply to the concepts that are defined informally, such as those described in natural language documentation and examples. If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent.
- Extendibility: An ontology should be designed to anticipate the uses of the shared vocabulary. It should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology monotonically. In other words, one should be able to define new terms for special uses based

on the existing vocabulary, in a way that does not require the revision of the existing definitions.

- Minimal encoding bias: The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. An encoding bias results when representation choices are made purely for the convenience of notation or implementation. Encoding bias should be minimized, because knowledge-sharing agents may be implemented in different representation systems and styles of representation.
- Minimal ontological commitment: An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities. An ontology should make as few claims as possible about the world being modeled, allowing the parties committed to the ontology freedom to specialize and instantiate the ontology as needed. Since ontological commitment is based on consistent use of vocabulary, ontological commitment can be minimized by specifying the weakest theory (allowing the most models) and defining only those terms that are essential to the communication of knowledge consistent with that theory.

Activity 3: Integrating existing ontologies

This process refers to the decision of whether not to use already existing ontologies, and how to process their integration. This may be done in parallel with either or both the capture and the coding processes. At the time this methodology was presented, the authors identified this as a difficult problem and one of the biggest challenges in developing a comprehensive methodology, leaving it as an open issue.

Evaluating the ontology

The authors take the definition of evaluation from Gómez-Pérez et al [9] to affirm that:

"To make a technical judgment of the ontologies, their associated software environment, and documentation with respect to a frame of reference (...) the frame of reference may be requirement specifications, competency questions, and/or the real world".

With this statement, the authors have shown that they decided not to tackle in detail the evaluation stage, leaving the study and interpretation of this problem for further works. The work they suggested as a hint was later developed by Gomez-Perez et al [48] in the METHONTOLOGY methodology, as it is illustrated in the respective section dedicated to this methodology, in section 2.3.4.

Documenting the ontology

The authors conclude that it may be desirable to have established guidelines for documenting ontologies, possibly differing according to the type and purpose of the ontology. These guidelines may be, for example, to locate similar definitions together or the create naming conventions, such as using upper or lowercase letters to name the terms, or writing the terms of the representation ontology in uppercase.

They cited Skuce [71], who pointed out that one of the main barriers to effective knowledge sharing is the inadequate documentation of existing knowledge bases and ontologies; he also argued that all important assumptions should be documented both about the main concepts defined in the ontology as well as about primitives used to express the definitions in the ontology.

Conclusions

This is an application independent method that focus in implementing an ontology from scratch.

It does not include any ontology management activities, and the importance of ontology support activities are identified, but they are only mentioned superficially. Regarding development-oriented activities, the main drawback of this method is the lack of a conceptualization process before implementing the ontology. The goal of a conceptualization process is to provide a domain model less formal than the implementation model but more than the definition of the model in natural language. Problems provoked by this lack of a conceptualization process are [32] that:

- Domain experts, human users, and ontologists have many difficulties in understanding ontologies implemented in ontology languages;
- Domain experts are not able to build ontologies in their domain of expertise, so there is a bottleneck in knowledge acquisition.

2.3.3. Gruninger and Fox's Method

Based on the experience of the TOVE project (TOronto Virtual Enterprise) [90], which was developed at the University of Toronto, Grüninger and Fox [35] published a formal approach to build and evaluate ontologies. This methodology has been used to build the TOVE ontologies, which are the pillars of the TOVE Enterprise Design Workbench, a design environment that allows the user to explore a variety of enterprise designs.

The approach starts by defining the ontology's requirements, through interacting with each particular enterprise and collecting the issues that arise among them; those issues are transformed

into natural language questions that the ontology must be able to answer, which the authors call the competency of the ontology.

The second step involves defining a specific terminology for the ontology - its properties, attributes and formal axioms, thus providing the language that is used to express the definitions in the terminology and the constraints required by the application.

The third step is inspired by the development of knowledge based systems using first order logic; the definitions and constraints of the application are specified on the terminology, whenever possible, and those specifications are represented in first order logic and implemented in Prolog. The competency of the ontology is then tested, proving completeness theorems against the competency questions.

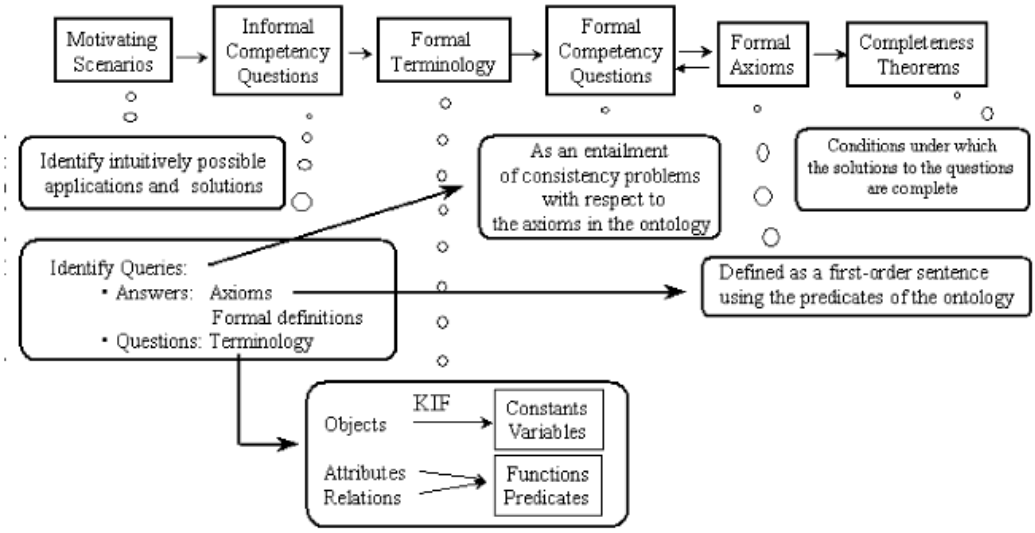

Figure 2.12 Processes in Gruninger and Fox's Methodology – adapted from [32]

The processes identified in this methodology, as shown in figure 2.12, are the following:

Identify motivating scenarios

The development of ontologies is motivated by scenarios that arise in the applications that will use the ontology. Such scenarios describe a set of the ontology's requirements that the ontology should satisfy after being formally implemented. A motivating scenario may also provide a set of intuitively possible solutions to the scenario problems. These solutions give a first idea of the informal intended semantics of the objects and relations that will later be included in the ontology.

Elaborate informal competency questions

Given the set of informal scenarios found, a set of informal competency questions are identified. Informal competency questions are those written in natural language to be answered by the ontology once the ontology is expressed in a formal language. The competency questions play the role of a type of requirement specification against which the ontology can be evaluated. This methodology proposes to stratify the set of competency questions: an ontology is not well-designed if all the competency questions are simple queries, that is, if the questions cannot be decomposed or composed into more specific or general questions, respectively. Competency questions can be split off into more specific (or atomic) competency questions, and the answer to a question can be used to answer more complex questions. Each competency question is useful as a base for obtaining assumptions, constraints, the necessary input data, etc.

Specify the terminology using first order logic

Once informal competency questions have been posed, the ontologist must specify the terminology of the ontology, which will be formally represented by means of concepts, attributes and relations in a first-order logic language or equivalent. From the answers in natural language to the competency questions, the ontologist extracts the knowledge to be included in the formal definitions of concepts, relations, and formal axioms.

Write formal competency questions using first order logic

Informal competency questions are written as an entailment of consistency problems with respect to the axioms in the ontology. Such axioms are defined in the next process.

To specify axioms using first-order logic

The axioms in the ontology specify the definitions of terms in the ontology and constraints in their interpretation; axioms are defined as first-order sentences using the predicates of the ontology.

To specify completeness theorems

Once the competency questions have been formally stated, we must define the conditions under which the solutions to the questions are complete. This forms the basis of completeness theorems for the ontology. Besides evaluating the fulfillment of the ontology requirements or completeness theorems also provide a means to determining the extendibility of the ontology, by making explicit the role that each axiom plays in proving the theorem.

Conclusions

This is a very formal methodology that takes advantage of the robustness of classic logic and can be used as a guide to transform informal scenarios in computable models. It is a well-founded methodology for building and evaluating ontologies, even lacking some management and support activities and though the ontology life cycle is not completely specified.

2.3.4. METHONTOLOGY

METHONTOLOGY [48] was developed within the Ontology Engineering Group at Universidad Politécnica de Madrid (UPM) , and enables the construction of ontologies at the knowledge level. It has its roots in the main activities identified by the software development process [43] and in knowledge engineering methodologies, and it is well suited for building ontologies either from scratch, reusing other ontologies as they are, or by a process of reengineering them.

The framework consists of: identification of the ontology development process with the identification of its main activities; a life cycle based on evolving prototypes; and the methodology itself, specifying the steps for performing the activities, the techniques used, the outcomes and their evaluation.

The ontology development process was proposed in the framework of this methodology and refers to those activities performed during ontology building. This process does not identify the order in which such activities should be performed: that is, the role of the ontology life cycle. METHONTOLOGY proposes an ontology building life cycle based on evolving prototypes because it allows adding, changing, and removing terms in each new version (prototype).

For each prototype, METHONTOLOGY proposes to begin with the schedule activity that identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. After that, the ontology specification activity starts and at the same time several activities begin inside the management (control and quality assurance) and support processes (knowledge acquisition, integration, evaluation, documentation, and configuration management). All these management and support activities are performed in parallel with the development activities (specification, conceptualization, formalization, implementation and maintenance) during the whole life cycle of the ontology.

Once the first prototype has been specified, the conceptual model is built within the ontology conceptualization activity. This is like assembling a jigsaw puzzle with the pieces supplied by the knowledge acquisition activity, which is completed during the conceptualization. Then the formalization and implementation activities are carried out. If some deficiency is detected after any

of these activities, we can return to any of the previous activities to make modifications or refinements.

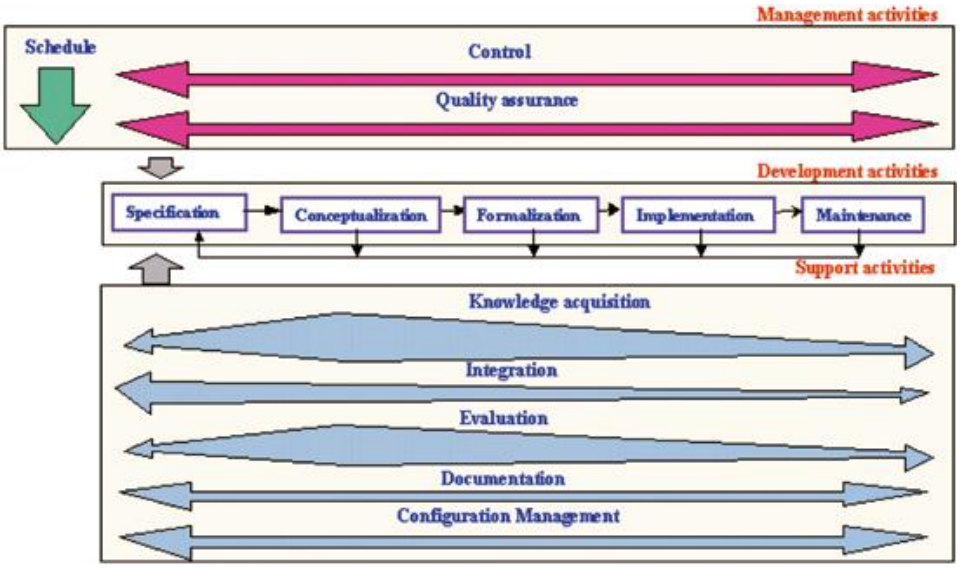

Figure 2.13 Development process of METHONTOLOGY – adapted from [48]

Figure 2.13 illustrates the ontology life cycle proposed in METHONTOLOGY, and summarizes the previous description. Note that the activities inside the management and support processes are carried out simultaneously with the activities inside the development process.

Related to the support activities, the figure also shows that the knowledge acquisition, integration and evaluation are greater during the ontology conceptualization, and that they decrease during formalization and implementation.

The reasons for this greater effort are:

- Most of the knowledge is acquired at the beginning of the ontology construction.
- The integration of other ontologies into the one we are building is not postponed to the implementation activity. Before the integration at the implementation level, the integration at the knowledge level should be carried out.
- The ontology conceptualization must be evaluated accurately to avoid propagating errors in further stages of the ontology life cycle.

The relationships between the activities carried out during ontology development are called intra-dependencies, and they define the ontology life cycle.

METHONTOLOGY also considers that the activities performed during the development of an ontology may involve performing other activities in other ontologies already built or under construction. Therefore, METHONTOLOGY considers not only intra-dependencies, but also inter-dependencies. Inter-dependencies are defined as the relationships between activities carried out when building different ontologies. Instead of talking about the life cycle of an ontology, we should talk about crossed life cycles of ontologies. The reason is that, most of the times and before integrating an ontology in a new one, the ontology to be reused is modified or merged with other ontologies of the same domain.

So, basically, METHONTOLOGY starts by identifying the following activities involved in the development of an ontology:

Specification

This activity intends to identify the purpose of the ontology, its intended users, scenarios of use, level of formality and its scope, which includes the set of terms to be represented, its characteristics and granularity.

This specification must be illustrated in an informal, semi-formal or formal document written in natural language, using a set of intermediate representations or using competency questions.

Knowledge Acquisition

Although this is an independent activity in the ontology development process, it can occur in parallel with the other activities; in fact, most of the knowledge acquisition is made during the requirements specification activity, and decreases as the ontology development process moves forward.

In this activity any type of knowledge source and any elicitation method can be used, although the roles of expert interviews and analyses of texts are specifically discussed in the methodology specification.

Conceptualization

In this activity, the domain knowledge acquired during the knowledge acquisition activity is organized and structured in a conceptual model that describes the problem and its solution in terms of the domain vocabulary identified in the specification activity. Domain terms are identified as concepts, instances, verbs relations or properties and each are represented using an applicable informal representation.

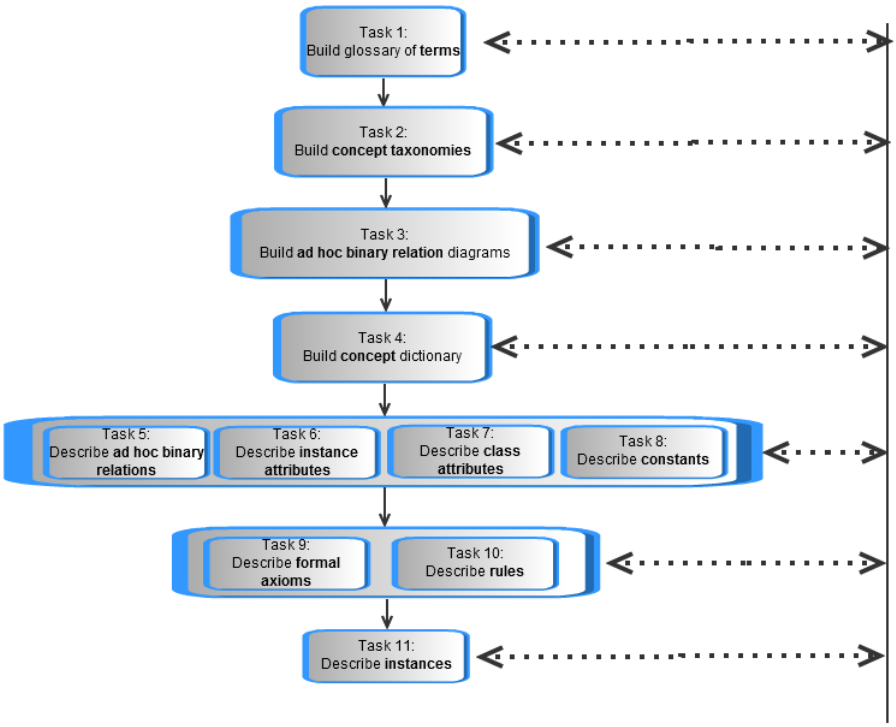

Figure 2.14 Conceptualization activity according to METHONTOLOGY [35]

METHONTOLOGY includes in the conceptualization activity the set of tasks for structuring knowledge, as shown in figure 2.14. The figure emphasizes the ontology components attached to each task, and specifies the order proposed to create such concepts during the conceptualization activity. This process is not sequential as in a waterfall life cycle model, though some order must be followed to ensure consistency and completeness of the knowledge represented. If new vocabulary is introduced, the ontologist can return to any previous task. The tasks are described in detail as follows:

Task 1: The ontologist starts to build the glossary of terms that identifies the set of terms to be included in the ontology (e.g. concepts, instances, attributes that represent concept properties, relations between concepts), their natural language definition, and their synonyms and acronyms.

Task 2: When the glossary of terms contains a considerable number of terms, the ontologist builds concept taxonomies to define the concept hierarchy. The output of this task could be one or more taxonomies where concepts are classified. To build concept taxonomies, the ontologist selects terms that are concepts from the glossary of terms. For this, it is really important to identify in the concept taxonomy sets of disjoint concepts, that is, concepts that cannot have common instances. METHONTOLOGY proposes to use the four taxonomic relations: Subclass-Of, Disjoint- Decomposition, Exhaustive-Decomposition, and Partition.

Task 3: Once the taxonomy has been built and evaluated, the conceptualization activity proposes to build ad hoc binary relation diagrams. The goal of this diagram is to establish ad hoc relationships between concepts of the same (or different) concept taxonomy.

Task 4: Once the concept taxonomies and ad hoc binary relation diagrams have been generated, the ontologist must specify which are the properties and relations that describe each concept of the taxonomy in a concept dictionary, and, optionally, their instances. A concept dictionary contains all the domain concepts, their relations, their instances, and their class and instance attributes. The relations specified for each concept are those whose domain is the concept.

Once the concept dictionary is built, the ontologist should define in detail each of the ad hoc binary relations, instance attributes and class attributes identified on the concept dictionary, as well as the main constants of that domain.

Task 5: The goal of this task is to describe in detail all the ad hoc binary relations included in the concept dictionary, and to produce the ad hoc binary relation table. For each ad hoc binary relation, the ontologist must specify its name, the names of the source and target concepts, its cardinality, its inverse relation and its mathematical properties.

Task 6: The aim of this task is to describe in detail all the instance attributes already included in the concept dictionary by means of an instance attribute table. Each row of the instance attribute table contains the detailed description of an instance attribute. Instance attributes are those attributes whose value(s) may be different for each instance of the concept. For each instance attribute, the ontologist must specify the following fields: its name; the concept it belongs to (attributes are local to concepts); its value type; its measurement unit, precision and range of values (in the case of numerical values); default values if they exist; minimum and maximum cardinality; instance attributes, class attributes and constants used to infer values of the attribute; attributes that can be inferred using values of this attribute; formulae or rules that allow inferring values of the attribute; and references used to define the attribute.

Task 7: The aim of this task is to describe in detail all the class attributes already included in the concept dictionary by means of a class attribute table. Each row of the class attribute table contains a detailed description of the class attribute. Unlike instance attributes, which describe concept instances and take their values in instances, class attributes describe concepts and take their values in the class where they are defined. Class attributes are neither inherited by the subclasses nor by the instances. For each class attribute, the ontologist should fill the following information: name; the name of the concept where the attribute is defined; value type; value; measurement unit and value

precision (in the case of numerical values); cardinality; the instance attributes whose values can be inferred with the value of this class attribute; etc.

Task 8: The aim of this task is to describe in detail each of the constants defined in the glossary of terms. Each row of the constant table contains a detailed description of a constant. For each constant, the ontologist must specify the following: name, value type (a number, a mass, etc.), value, the measurement unit for numerical constants, and the attributes that can be inferred using the constant.

Once that concepts, taxonomies, attributes and relations have been defined, the ontologist should describe formal axioms (task 9) and rules (task 10) that are used for constraint checking and for inferring values for attributes. And only optionally should the ontologists introduce information about instances (task 11).

Integration

With the goal of speeding up the development process or to obtain some uniformity across ontologies, this activity considers reusing definitions already built into other ontologies, instead starting from scratch.

Implementation

The ontology is formally represented in a language and implemented in an ontology development environment, such as Ontolingua [24], for example. Ontolingua is a set of tools and a service (including an ontology server) that provides a distributed collaborative environment to browse, create, edit, modify, and use ontologies.

Evaluation

Much emphasis is placed on this stage in METHONTOLOGY. The techniques used are largely based on those used in the validation and verification of KBSs. In [12] a set of guidelines is given on how to look for incompletenesses, inconsistencies and redundancies.

Documentation

The final documentation in this methodology is a collation of documents that result from other activities, which means that documentation is an activity to be done during the whole ontology development process: after the specification phase, we may get a requirements specification document; after the knowledge acquisition phase, a knowledge acquisition document; after the conceptualization, a conceptual model document that includes a set of intermediate representations

that describe the application domain; after the formalization, a formalization document; after the integration, an integration document; after the implementation, an implementation document, and during the evaluation, an evaluation document.

Conclusions

METHONTOLOGY is one of the most mature approaches, providing accurate descriptions of each activity and being recommended by Foundation of Intelligent Physical Agents (FIPA)[5] for the ontology construction task.

It is an application-independent methodology that is based on evolving prototypes, that is, the ontology grows according to the needs, which permits modifying, adding, and removing definitions in the ontology at any time. It tackles every group of activities of the ontology development process, and considers the utilization of existing ontologies and other resources to reduce the effort of the knowledge acquisition phase.

2.3.5. OTK (On-To-Knowledge)

The aim of the On-To-Knowledge project [76] is to apply ontologies to electronically available information for improving the quality of Knowledge Management (KM) applications in large and distributed organizations. Some of the partners of this project are the Institute AIFB of the University of Karlsruhe, the Vrije Universiteit of Amsterdam[6], and British Telecom.

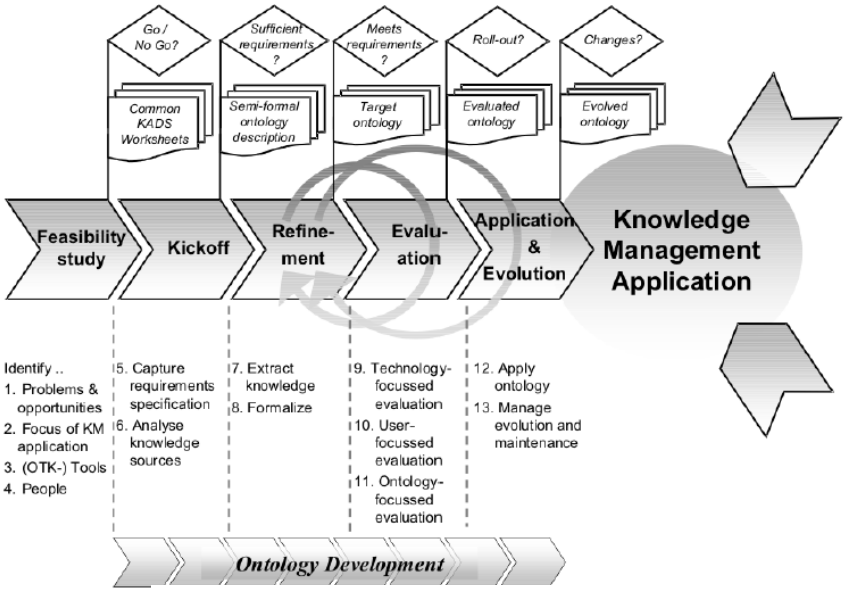

Figure 2.15 OTK Knowledge Meta Process [76]

[5] http://www.fipa.org/
[6] http://www.vu.nl/

The project includes a methodology for building ontologies to be used by Knowledge Management applications; this methodology proposes to build the ontology taking into account how the ontology will be used in further applications. Consequently, ontologies developed with this methodology are highly dependent on the application. Another important characteristic is that On-To-Knowledge proposes ontology learning for reducing the efforts made to develop the ontology.

The methodology also includes the identification of goals to be achieved by knowledge management tools, and is based on an analysis of usage scenarios.

As illustrated in figure 2.15, OTK divides the ontology engineering task into five main processes:

Process 1: Feasibility study

On-To-Knowledge adopts the kind of feasibility study described in the CommonKADS methodology [70]. The feasibility study serves as a basis for the kickoff process and as a decision support for economical, technical and project feasibility, identifying promising focus areas and potential target solutions.

For this analysis, the CommonKADS methodology offers three models: the organization, task, and agent model. The process of building these models proceeds in the following steps:

- Carry out a scoping and problem analysis study, consisting of two parts:
    - Identifying problem/opportunity areas and potential solutions, and putting them into a wider organizational perspective.
    - Deciding about economic, technical and project feasibility, in order to select the most promising focus area and target solution.
- Carry out an impacts and improvements study, for the selected target solution, again consisting of two parts:
    - Gathering insights into the interrelationships between the business task, actors involved, and use of knowledge for successful performance, and what improvements may be achieved here.
    - Deciding about organizational measures and task changes, in order to ensure organizational acceptance and integration of a knowledge system solution.

Process 2: Kickoff

The actual development of the ontology begins in the kickoff phase, resulting in a semi-formal ontology requirements specification document (ORSD). The ORSD describes the domain and goals of

the ontology, design guidelines (e.g. naming conventions), valuable knowledge sources, potential users and use cases, as well as applications supported by the ontology.

Ontology Requirements Specification Document Template	
	Purpose
	The main function or role that the ontology should have.
	Scope
	The general coverage and degree of detail the ontology should have
	Implementation Language
	The formal language that the ontology should have
	Intended End-Users
	The intended end-users expected for the ontology
	Intended Uses
	The intended uses expected for the ontology
	Ontology Requirements
	a. Non Functional Requirements
	The general requirements or aspects that the ontology should fulfill, including optional priorities for each requirement
	b. Functional Requirements: Groups of Competency Questions
	The content specific requirements that the ontology should fulfill, in the form of groups of competency questions and their answers
	Pre-Glossary of Terms
	a. Terms from Competency Questions
	The list of terms included in competency questions and their frequencies
	b. Terms from Answers
	The list of terms included in the answers and their frequencies
	c. Objects
	The list of objects included in the competency questions and in their answers

Table 2.2 ORSD Template –adapted from [87]

The ORSD should lead the ontology engineer to decide about the inclusion or exclusion of concepts in the ontology, and about their hierarchical structure. In fact, this specification is useful to elaborate a draft version containing few but seminal elements. This first draft is called "baseline ontology". The most important concepts and relations are identified on an informal level.

In the kickoff process the developers should also look for potentially reusable ontologies that were already developed.

Process 3: Refinement

The goal here is to produce an application-oriented "target ontology" that is according to the specification given in the kickoff process. The baseline ontology obtained in the previous process is refined by means of interaction with experts in the domain. During the elicitation, the concepts are gathered on one side and the terms to label the concepts on the other. Then, terms and concepts are mapped. The On-To-Knowledge methodology proposes the use of intermediate representations to model the knowledge. If several experts participate in the construction of the ontology, it is necessary to reach an agreement.

Once an agreement has been reached, the ontology is implemented by using an ontology language. Such language is selected according to the specific requirements of the envisaged application. To carry out the formalization, On-To-Knowledge recommends the OntoEdit [75] ontology editor, which generates automatically the ontology code in several languages.

The major decision that needs to be taken to finalize or not this step is whether the target ontology fulfills the requirements specified in the kickoff phase. Typically an ontology engineer compares the initial requirements with the current status of the ontology. This decision is typically based on the personal experience of ontology engineers. The authors suggested that a good rule of orientation is that this first ontology should provide enough "flesh" to build a prototypical application, and this application should be able to serve as a first prototype system for evaluation.

Process 4: Evaluation

The evaluation process serves as a proof of the usefulness of the developed ontologies and their associated software environment. The product obtained is called the ontology based application, and at the end of this process one must decide if this ontology fulfills all evaluation criteria relevant for the envisaged application.

The authors distinguish between three different types of evaluation: technology-focused evaluation, user-focused evaluation and ontology-focused evaluation.

The technology-focused evaluation consists of two main aspects: the evaluation of properties of ontologies generated by development tools, and the evaluation of the technology properties.

The most important point from the author’s perspective regarding user-focused evaluation is to evaluate whether users are satisfied by the KM application. More specifically, whether an ontology based application is at least as good as one already existing application that solve similar tasks.

Beside the above mentioned process oriented and pragmatic evaluation methods, one also needs to formally evaluate ontologies. The author refers the OntoClean approach [37], which is a methodology for evaluating ontologies based on philosophical notions.

Process 5: Application and Evolution

The application of ontologies is the effective usage of ontology based systems. The evolution of ontologies is primarily an organizational process. There have to be strict rules to the update, insert and delete processes of ontologies, clarifying who is responsible for the ontology maintenance and how it should be carried out.

The outcome of an evolution cycle is an evolved ontology, which is typically another version of it. The major decision to be taken is when to initiate another evolution cycle for the ontology.

Conclusions

On-To-Knowledge methodology is also a mature approach, being an application dependent methodology and the one that describes more activities; in fact, it approaches each one of the ontology management, development and support sub-activities, although only detailing the specification activity. It also takes into account the possibility of using existing ontologies and resources.

It is also the only methodology in this state of the art to take into account pre-development oriented activities, such as environment and feasibility studies.

2.3.6. General Conclusions

Given the methodologies details presented, and considering the specific context of our project, the most important questions that arise when deciding upon what methodology we use are the following:

Strategy according to the application

This criterion is related to the degree of dependency of the ontology with the application using it. Considering this criterion, the methodologies and methods can generally be classified into the following types:

- Application dependent. Ontologies are built on the basis of the applications that use them.
- Application semi-dependent. The possible scenarios of ontology use are identified in the specification stage.

- Application-independent. The process is totally independent of the uses of the ontology in applications.

Use of core ontologies

This criterion refers to whether it is possible or not to use a core ontology as a starting point in the development of the domain ontology. On the context of the XEO-ECC project, this is particularly relevant, as extensibility of a core ontology that is one of the crucial requirements identified.

Effort on knowledge acquisition

This issue is related to whether the methodologies approached consider the reutilization of existing resources to acquire knowledge of a given domain. In the XEO-ECC project this is a particularly interesting question when approaching the problem of extending or integrating the core ontology to a specific domain; in most cases, an organization have most of its knowledge relying on some kind of non-ontological resource (e.g. relational databases), and it is advisable to reuse as many resources as possible, as it can strongly reduce the effort of the domain ontology development.

Tools supporting the methodologies

Most of the approaches do not have a specific tool that gives them technology support. Besides, none of the available tools covers all the activities necessary in ontology building.

2.4. Re-engineering Non-Ontological Resources into Ontologies

With the goal of speeding up the ontology development process, ontology engineers have started to reuse as much as possible available ontologies and non-ontological resources such as classification schemes, thesauri, lexicons and folksonomies, which already have some degree of consensus [29]. The reuse of such non-ontological resources necessarily involves their re-engineering into ontologies. Non-ontological resources are highly heterogeneous in their data model and contents: they encode different types of knowledge, and they can be modeled and implemented in different ways.

In this section existing distinctions between types of non-ontological resources are enumerated (section 2.4.1), and software re-engineering is introduced (section 2.4.2) as a related subject to non-ontological resource re-engineering (section 2.4.3). Some methods related to re-engineer specific non-ontological resources are also presented (section 2.4.4), as well as the NeOn project[7] method for re-engineering any kind of non-ontological resources (section 2.4.5).

2.4.1. Types of Non-Ontological Resources

There is a big amount of non-Ontological resources (NORs) that embody knowledge about some particular domains, and that represent some degree of consensus for a user community. These resources may be presented in the form of free texts, textual corpora, web pages, standards, catalogues, web directories, classifications, thesauri, lexicons and folksonomies, among others. Non-ontological resources have related semantics which allows for interpreting the knowledge they contain. Regardless of whether the semantic is explicit or not, the main problem is that the semantics of Non-Ontological Resources is not always formalized, and this lack of formalization prevents using them as ontologies.

The analysis of the literature has revealed that there are several different ways of categorizing non-ontological resources:

[7] http://www.neon-project.org/ NeOn is a project involving 14 European partners and co-funded by the European Commission's Sixth Framework Programme. The aim of NeOn is to create an open infrastructure and associated methodology to support the development life-cycle of semantic applications. It contains an open source multi-platform ontology engineering environment called NeOn toolkit, which aims to provide comprehensive support for various activities in the ontology engineering life-cycle.

- Maedche et al. [50] and Sabou et al. [88] classify NORs into unstructured (e.g. free text), semi-structured (e.g. folksonomies) and structured (e.g. databases) resources.
- Gangemi et al. [28] distinguish catalogues of normalized terms, glossed catalogues (catalogues with natural language glossary), and taxonomies.
- Hodge [42] proposes characteristics such as structure, complexity, relationships among terms, and historical functions for classifying them.

Recently, in 2009, a new approach to this problem has been proposed by Asuncion Perez et al [29], which splits the categorization according to three different features: the type of NOR, which refers to the type of knowledge encoded by the resource; the data model (that is, the design data model used to represent the knowledge encoded by the resource) and the resource effective implementation.

- According to the type of NOR, they classify them into:
    - Glossaries: A glossary is a terminological dictionary that contains designations and definitions from one or more specific subject fields. The vocabulary may be monolingual, bilingual or multilingual.
    - Lexicons: In a restricted sense, a computational lexicon is considered as a list of words or lexemes hierarchically organized and normally accompanied by meaning and linguistic behavior information. An example is WordNet [25], which is the best known computational lexicon of English.
    - Classification schemes: A classification scheme is the descriptive information for an arrangement or division of objects into groups based on characteristics the objects have in common.
    - Thesauri: Thesauri are controlled vocabularies of terms in a particular domain with hierarchical, associative and equivalence relations between terms. Thesauri are mainly used for indexing and retrieval of articles in large databases.
    - Folksonomies: A folksonomy is the result of personal free tagging of information and objects (anything with a URI) for one's own retrieval. An example of the use of folksonomies is the del.icio.us website.
- Regarding the different ways for representing the knowledge encoded by the resource, several data models are identified for classification schemes, as shown in figure 2.16:
    - Path Enumeration [13]: A path enumeration model is a recursive structure for hierarchy representations defined as a model which stores for each node the path (as

a string) from the root to the node. This string is the concatenation of the nodes code in the path from the root to the node. Figure 2.16a shows this model.

- Adjacency List [13]: An adjacency list model is a recursive structure for hierarchy representations comprising a list of nodes with a linking column to their parent nodes. Figure 2.16b shows this model.
- Snowflake [53]: A snowflake model is a normalized structure for hierarchy representations. For each hierarchy level a table is created. In this model each hierarchy node has a linked column to its parent node. Figure 2.16c shows this model.
- Flattened [53]: A flattened model is a denormalized structure for hierarchy representations. The hierarchy is represented using one table where each hierarchy level is stored on a different column. Figure 2.16d shows this model.

Path Enumeration	Category Name	Category Description
1	Category1	Category1Desc
11	Category11	Category11Desc
111	Category111	Category111Desc
12	Category12	Category12Desc
121	Category121	Category121Desc
2	Category2	Category2Desc
...	...	...

a) Path Enumeration

Category Code	Category Name	Parent Category
1	Category1	Null
2	Category2	Null
3	Category3	1
4	Category4	1
5	Category6	3
6	Category7	4
...	...	...

b) Adjacency List

First Level Categories Entity		
Category Code	Category Name	Category Description
1	Category1Level1	Category1Level1Desc
2	Category2Level1	Category2Level1Desc
...	...	...

Second Level Categories Entity			
Category Code	First Level Category	Category Name	Category Description
1	1	Category1Level2	Category1Level2Desc
2	1	Category2Level2	Category2Level2Desc
...	...	...	...

Third Level Categories Entity			
Category Code	Second Level Category	Category Name	Category Description
1	1	Category1Level3	Category1Level3Desc
2	2	Category2Level3	Category2Level3Desc
...	...	...	...

c) snowflake

Flattened Entity					
First Level		Second Level		Third Level	
Category Code	Category Name	Category Code	Category Name	Category Code	Category Name
1	Category1Level1	1	Category1Level2	1	Category1Level3
1	Category1Level1	2	Category2Level2	2	Category2Level3
2	Category2Level1	...	...	...	...
...	...	...	...	...	...

d) flattened

Figure 2.16 Classification Schemes Data Models [78]

- According to the implementation, NORs can be classified into databases, XML files, flat files or spreadsheets.

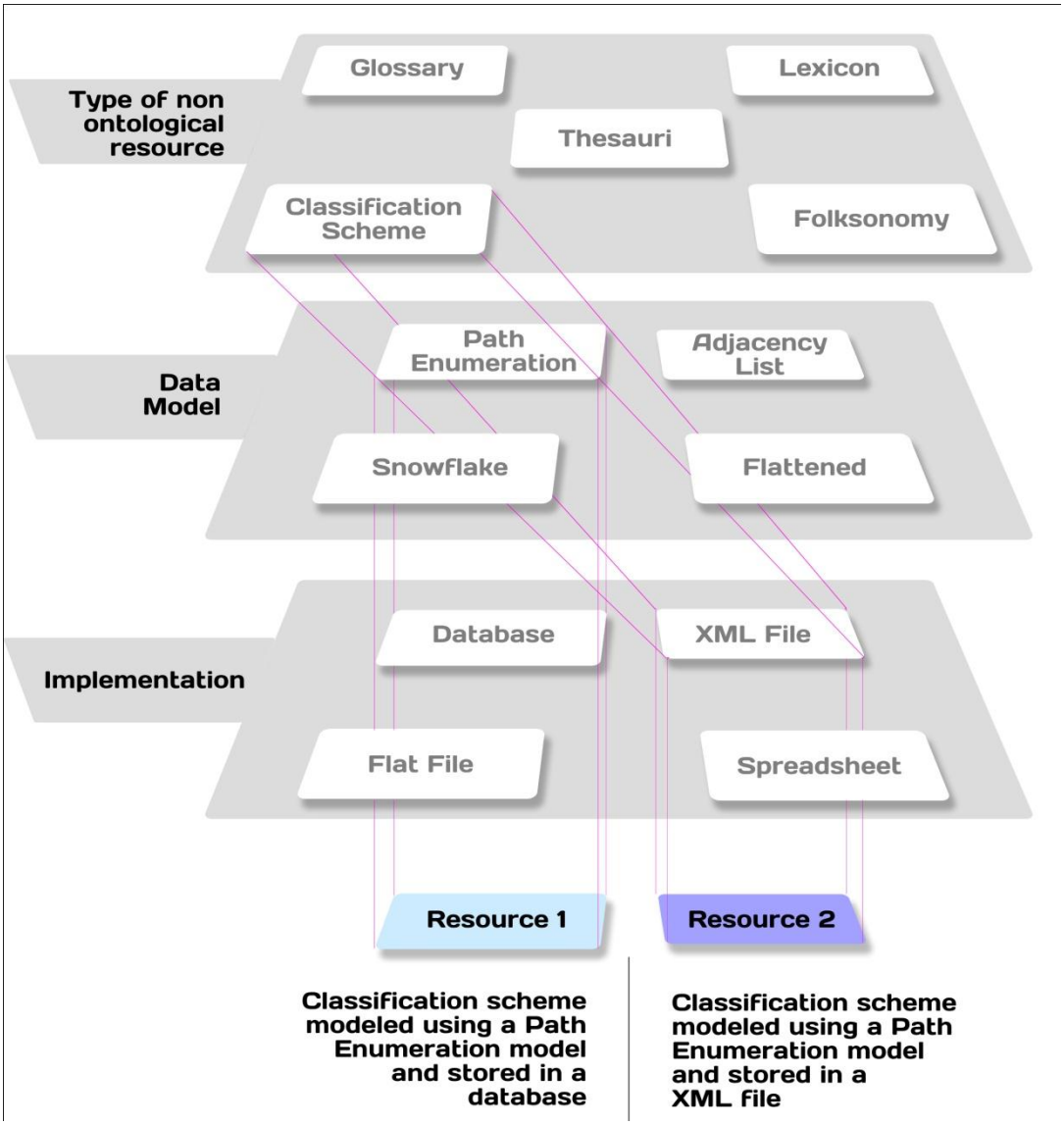

Figure 2.17 Non-Ontological Resources Categorization [83]

Figure 2.17 shows how a given type of NOR can be modeled following one or more data models, each of which could be implemented in different ways at the implementation layer. As an example, the figure shows a classification scheme modeled using a path enumeration model. In this case, the classification scheme is implemented in a database and in an XML file.

2.4.2. Software Re-engineering

Software re-engineering [19] is defined as the examination of the design and implementation of an existing legacy system, and the application of the different techniques and methods to redesign and reshape that system into hopefully better and more suitable software. Software re-engineering main activities are:

- Reverse engineering is the process of analyzing a subject system to identify the system components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction.
- Alteration, also called restructuring, is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior.
- Forward engineering is the traditional process of moving from high level abstractions and logical, implementation-independent designs to the physical implementation of a system.

Re-engineering patterns [66] are patterns that describe how to change a legacy system into a new, refactored system that fits current conditions and requirements. Their main goal is to offer a solution for re-engineering problems. They are also on a specific level of abstraction. They describe a process of re-engineering without proposing a complete methodology, and they can sometimes suggest a type of tool that one could use.

2.4.3. Non-Ontological Resource Re-engineering

Non-ontological resource re-engineering, as defined in the Glossary of Activities in Ontology Engineering [74], refers to the process of taking an existing non-ontological resource and transforming it into an ontology.

The research in NOR re-engineering has been mainly centered on the transformation of standards [41], thesauri and lexicons [88], XML files [30], hierarchical classifications [41], folksonomies [88], relational databases [5], and spreadsheets [39]. These works only concentrate on the re-engineering process of the type and implementation of NOR.

Characteristics of the Transformation Process

In [88], two approaches for the non-ontological resource transformation are distinguished. One of them consists in transforming resource schema into an ontology schema, and then resource content into instances of the ontology. The other one transforms resource content into an ontology schema. Gomez-Lopez et al [29] later identified a third transformation approach, which consists in

transforming the resource content into instances of an existing ontology. Figure 2.18 illustrates these three approaches.

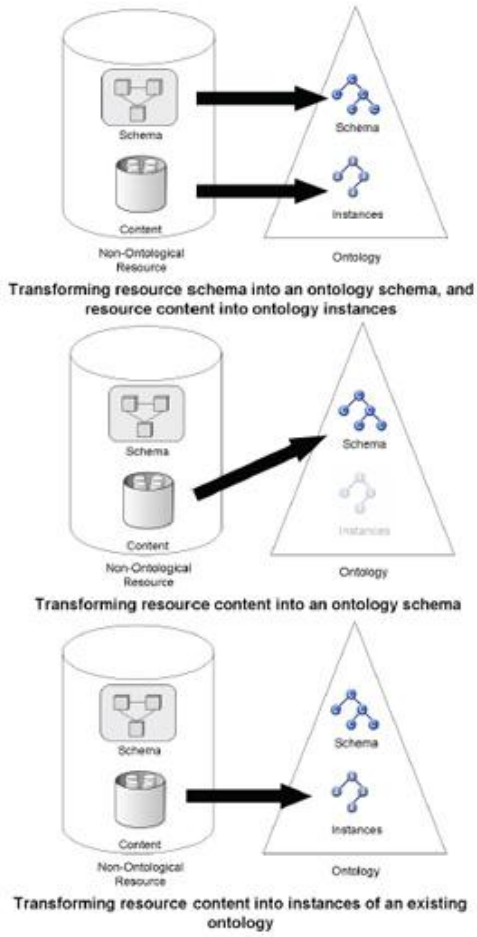

Figure 2.18 Non Ontological Resource Transformation Approaches [29]

Also regarding the transformation process, in the NeOn project study of methods and tools supporting re-engineering [83] other specific characteristics have been identified:

- The transformation process can follow a one-step transformation of the resource, that is, converting the overall non-ontological resource into an ontology, or an incremental transformation, in which specific components of the resource are converted into an ontology, without applying a whole transformation.
- The transformation process can be automatic, semi-automatic or manual.
- The transformation process is carried out by using either an ad-hoc wrapper, or a formal specification of the conversions between entities of the resources (a non-ontological resource and ontology) with an associated transformation condition that defines complex rules (in

which case it is necessary a processor or interpreter). The formal specification of the conversions can be declarative or not.

- The transformation process performs a full conversion of the resource. Full conversion implies that all queries that are possible on the original source are also possible in the resulting ontology

Characteristics of the Resulting Ontology

Regarding the resulting ontology, in the NeOn project [83] identifies the following characteristics have been identified:

- The generated ontology components are classes, attributes, relations, or instances.
- The ontology implementation languages are OWL, RDF(S).
- The research work generates a single ontology or several ontologies. It is not distinguished if the ontologies generated are interconnected or not.

2.4.4. Non-ontological Resource Re-engineering Methods

In this section I will enumerate some of the methods identified for non-ontological resources conversion. I will briefly describe some details about the ones that have a specific tool or implementation associated to it.

Transforming Classification Schemes into Ontologies

The two main methods [83] for transforming classification schemes are GenTax [41] and Hakkarainen et al.'s method [38]. The GenTax approach and specific tool is discussed below, as it is a method that tackles classifications schemes in general; the Hakkrainen et al's methods is a more specific method, coined to study the semantic relationship between the ISO 15926-26 model and OWL DL [83].

GenTax

GenTax is a method presented by Hepp et al. in [41] for semi-automatically deriving consistent RDF(S) and OWL ontologies from hierarchical classifications, thesauri and informal taxonomies. These authors subsume all three types (taxonomies, thesauri, and hierarchical classifications) under the term hierarchical categorization schema; the three types have in common that they include a set of categories and some form of a hierarchical order. They have implemented a preliminary tool, named SKOS2GenTax, to support their method. Their prototype consists of a Java program that expects the informal categorization schema to be stored in a RDBMS. The program accesses the categories via an ODBC link.

GenTax consists of the following steps:

- To pre-process and create a formal representation of the resource.
- To derive classes from each category and set an ad-hoc relation among classes according to a given context.
- To derive a class from each category and set a taxonomic relations among them.
- To generate the ontology in an ontology language.

This method produces one single ontology. The ontology components generated are classes and relations. The ontology is expressed in OWL-DL or RDF(S).

Methods for Transforming Folksonomies into Ontologies

The two main methods [83] for transforming folksonomies are T-ORG [1], by Abbasi et al., and [49] by Maala et al. The T-ORG approach and specific tool is discussed below; The Maala et al. approach depicts a conversion process from Flickr[8] tags to RDF descriptions so, given its specificity, I decided to leave it just as a reference and not to specify its details.

T-ORG

Abbasi et al. [1] present a mechanism to transform a set of tags of a given folksonomy into instances of an existing ontology. However, they do not mention at all the implementation of the resource.

The purpose of this method is to organize resources by classifying their tags into concepts of the ontology. This process is done by selecting concepts from single or multiple ontologies related to the correspondent categories. The authors use lexico-syntactic patterns and Google API for searching the appropriate categories of the tags. This method follows the approach of transforming the resource content into instances of an existing ontology, and their authors have implemented the T-ORG tool to support this method. However, the method does not tackle the internal data model of the folksonomy. On the other hand, how the resource data is represented and accessed for the transformation it is not described. This method does not keep the resource provenance information; therefore, the resulting ontology does not keep the reference to the non-ontological resource.

The method employs an ad-hoc wrapper for discovering the conversions between the ontologies and the tags. This method consists in:

[8] http://www.flickr.com/

- Selecting the ontology. The user selects the ontologies relevant to the categories. Concepts from these ontologies are used as categories. The authors rely on the Swoogle[9] search engine for the selection of ontologies.
- Pruning and refining the ontology. Ontologies must be pruned and refined for the desired categories. Unwanted concepts are pruned, whereas redundant and conflicting concepts are refined, and missing concepts are added to the given ontology. It is not specified if this is a manual or automatic process.
- Classifying the tags. The authors propose a new classification algorithm for classifying the tags, namely, the T-KNOW algorithm. This algorithm classifies the tags into categories using its pattern library (lexico-syntactic patterns), and categories extracted from a given ontology and Google search results.
- Browsing the resources. After classifying each tag, resources may be browsed according to the categories assigned to their tags.

This method manages several ontologies and the ontology components generated are instances. The method does not use any specific ontology language, but light weight ontologies instead.

Methods for Transforming XML Files into Ontologies

The three main methods [83] for transforming XML files into ontologies are presented in [3, 30, 21]. In [30], García et al. have conceived a specific implementation of their method, which is described below.

García et al.

García et al. in [30] introduce a method to create an ontology from the XML schema and populate it with instances created from the XML data.

This method follows the approach for transforming the resource schema into the ontology schema, and then resource content into instances of the ontology. It uses a formal specification of the conversions between entities of the resource and the ontology. The method consists of the following steps:

- XSD2OWL Mapping. In this step the semantics implicit in the schema is captured. This semantics is determined by the combination of XML Schema constructs. This step is quite transparent and captures a great part of XML Schema semantics. To check the resulting

[9] http://swoogle.umbc.edu

ontologies OWL validators have been used; the XSD2OWL tool has also been used. For checking the resulting ontologies, the user interaction is needed.

- XML2RDF Mapping. In this step a structure-mapping approach has been selected. This approach is based on translating XML metadata instances to RDF instances that instantiate the corresponding constructs in OWL. To do this, the XML2RDF tool has been used

This method has been applied to MPEG-7[10] XML Schemas generating a MPEG-7 ontology[11]. The only adjustment that has been made to the automatically generated ontology has been done to resolve a name collision between OWL class and a RDF property. The tools regarding the two conversion steps are available in the ReDeFer Project[12] webpage.

The method produces one single ontology. The ontology components generated are classes, attributes, relations, and instances and they are expressed in RDF/OWL Full.

Methods for Transforming Flat Files into Ontologies

The main method [83] to transform a flat file is presented in [27] and is described in this section, in which the specific tool ConvertToRdf is also discussed.

Foxvog et al.

Foxvog et al. [27] present a method to transforming Electronic Data Interchange (EDI)[13] messages into ontologies. EDI is intended to handle all aspects of business transactions such as ordering, acknowledgements, pricing, status, scheduling, shipping, receiving, invoices, payments, and financial reporting. Hundreds of standard message types are defined with specified formats.

This method follows the approach of transforming a resource schema into an ontology schema, and resource content into ontology instances. Such transformation is performed semi-automatically with an ad-hoc conversion program developed for that purpose. This method does not tackle the internal data model of the resource, nor does it describe how the resource data is represented and accessed for the transformation. It does not provide the resource provenance information, so the resulting ontology does not keep the reference to the flat file.

[10] http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm
[11] http://rhzomik.upf.edu/ontologies/mpeg7ontos
[12] http://rhizomik.net/html/redefer/
[13] http://www.ifla.org/VI/5/reports/rep4/42.htm#chap2

The method for transforming ASC X12 [14]messages into ontologies consists in:

- Syntactic transformation. In this step it is necessary to define and encode a vocabulary, i.e. create a set of classes, for specifying the formats of Transaction Sets, Data Segments, Data Elements and Code Sets.
- Semantic transformation. In this step it is possible to create separated ontologies for different Transaction Sets. Also, classes or individuals are created for each Data Element Code that is applicable for the chosen group of Transaction Sets. Classes or relations are created for each applicable Data Element. Relations or rules are created for each Data segment.

The method produces several ontologies. The ontology components generated are classes, attributes, relations, and instances and they are expressed in OWL Full, CycL, and WSML.

ConvertToRdf

ConvertToRdf[15] is a tool for automatically converting delimited text data into RDF via a simple mapping mechanism. The input resources are delimited text files. This tool supports the approach to transform resource content into instances of an existing ontology, performs a semi-automatic conversion and employs a formal specification of the conversions between entities of the resource and the ontology. ConvertToRdf contemplates syntactic transformation aspects, and how symbols are structured in the file and ontology formats.

It also tackles semantic transformation aspects, and the semantic interpretation of the resource elements when defining transformations to ontology elements. The tool produces one single ontology, and the resultant ontology instances are expressed in RDF.

Transforming Spreadsheet Files into Ontologies

The three main tools for transforming spreadsheet files are TopBraid Composer, described previously in this section, Excel2rdf, and RDF123. The remaining two are described next.

Excel2rdf

Excel2rdf[16] is a Microsoft Windows program that converts Excel files into valid RDF. The input resource is an Excel spreadsheet. This tool supports the approach to transform resource content into instances of an existing ontology and performs a semi-automatic conversion with an ad-hoc wrapper.

[14] http://www.x12.org/|
[15] http://www.mindswap.org/~mhgrove/convert/
[16] http://www.mindswap.org/~Erreck/excel2rdf.shtml

This tool contemplates syntactic transformation aspects, and how symbols are structured in the spreadsheet and ontology formats.

The tool generates one single ontology. The resultant ontology instances are expressed in RDF.

RDF123

RDF123 is a highly flexible open source tool for semi-automatically transforming spreadsheet data to RDF that works on CSV files and also Google spreadsheets. This tool was presented by Han et al. in [39] and it was motivated by the fact that spreadsheets are easy to understand and use, offer intuitive interfaces and have representational power adequate for most purposes. Also the liberty that people take with spreadsheets will sometimes require different rows to be translated with differing schemas.

This tool follows the approach used to transforming resource content into instances of an existing ontology. RDF123 defines a formal specification of the conversions between entities of the resource and more than one ontology. It intends to create instances of existing ontologies. Every row of a spreadsheet will generate a row graph, and the RDF graph produced for the whole spreadsheet is the merge of all row graphs, eliminating duplicated resources and triples.

RDF123 consists of the following two components:

- RDF123 application, which is the component whose main purpose is to give users an interactive and easy to use graphical interface for creating the map graph and outputting the map graph in RDF syntax. It also supports a full work cycle of translating a spreadsheet into RDF and importing a CSV file into a graphical spreadsheet editor and translating the spreadsheet into RDF by applying the map graph. This application is composed of three internal frames: (1) the prefix definition frame which works as a prefix library; (2) the spreadsheet editor which enable users to open a CSV file, edit the file in a similar way to Excel, and save the file; and (3) the interactive graph editor that allows users to create and remove a vertex/edge, drag a vertex, and change properties of a vertex/edge.
- RDF123 Web Service, which aims to provide a public service that translates online spreadsheets into RDF. This component also functions as the host of RDF documents URIs coming from online spreadsheets.

The tool enables to keep data in its original format, which provides two benefits: the same data can be available in different domains just by associating it with different map files; and when the

ontology or the spreadsheet evolves and changes, to make the data adapt to that change it is only necessary to modify the map file, instead of regenerating the hard coded RDF document.

It should be added that this tool produces more than one ontology and that the resulting ontology instances are expressed in RDF.

2.4.5. NeOn Method for Re-engineering Non-ontological Resources

In a nutshell, the NeOn approach for NOR re-engineering [83] considers as input a pool of NORs and patterns for re-engineering NORs. NORs, as mentioned before, include lexica, classification schemes, thesauri, etc. Regarding patterns for re-engineering NORs, they provide solutions to the problem of transforming NORs into ontologies. These patterns are included in the NeOn project patterns library [68].

General Model for Non-Ontological Resource Re-engineering

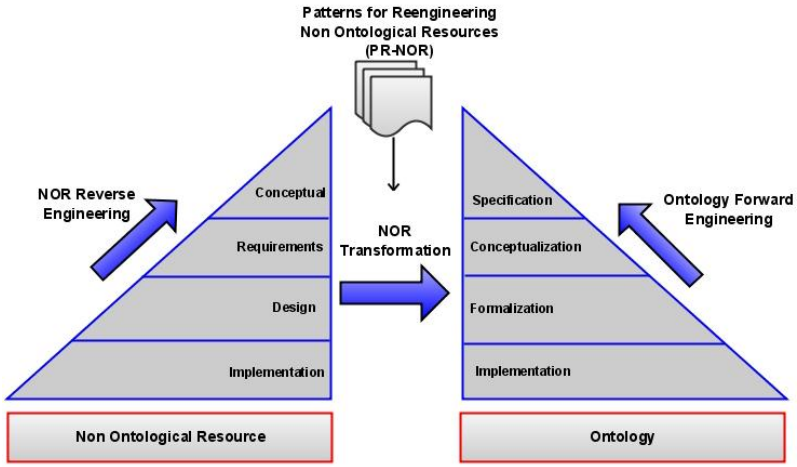

Figure 2.19 Re-engineering Model for Non-Ontological Resources [83]

Based on the software re-engineering model presented in [88], they propose a model for NOR re-engineering, as illustrated in figure 2.19.

The NOR re-engineering process consists of the following activities, which are defined in a Glossary of Activities in Ontology Engineering [74]:

- Non-Ontological Resource Reverse Engineering, whose goal is to analyze a NOR to identify its underlying components and create representations of the resource at the different levels of abstraction (design, requirements and conceptual). Since NORs can be implemented as XML files, databases or spreadsheets, among others, they are considered as software resources, and therefore, it is used the software abstraction levels shown in Fig. 2.19 within this activity.

Here the requirements and the essential design, structure and content of the NOR must be recaptured.

- Non-Ontological Resource Transformation, whose goal is to generate a conceptual model from the NOR. The use of Patterns for Re-engineering Non-Ontological Resources (PR-NOR) is proposed to guide the transformation process. First, the non-ontological resource type has to be identified. Second, the internal data model of the non-ontological resource has to be identified as well. Third, the semantics of the relations between the non-ontological resource entities have to be identified; these semantics can be a) subClassOf, b) an ad-hoc relation like partOf or c) a mix of subClassOf and ad-hoc relations. Next, a pattern for re-engineering non-ontological resources has to be searched according to the type of non-ontological resource, the internal data model and the semantics of the relations between the non-ontological resource entities. Finally, the selected re-engineering pattern has to be applied to transform the non-ontological resource into a conceptual model.
- Ontology Forward Engineering, whose goal is to output a new implementation of the ontology on the basis of the new conceptual model. Here the ontology levels of abstraction are used to depict this activity because they are directly related to the ontology development process.

Patterns for Re-engineering Non-Ontological Resources

According to [68], the use of re-engineering patterns for transforming non-ontological resources into ontologies has several advantages. The most representative are:

- To improve the efficiency of the re-engineering process.
- To make the transformation process easier for both ontology engineers and domain experts.
- To improve the reusability of non-ontological resources.

Patterns for re-engineering non-ontological resources (PR-NOR) define a procedure to transform the NOR components into ontology representational primitives. To this end, patterns take advantage of the NOR underlying data model. The data model defines how the different components of the NOR are represented.

According to the non-ontological resource categorization presented in section 2.4.1, the data models can be different even for the same type of non-ontological resource. A process can be defined for every data model, with a well-defined sequence of activities to extract the non-ontological resources components and then to map these components to a conceptual model of an ontology. Each of these processes can be expressed as a pattern for re-engineering non-ontological resources.

The resulting ontologies proposed by the patterns for re-engineering non-ontological resources are modeled following the recommendations provided by some other ontological patterns such as logical and architectural patterns [77]. The current inventory of NeOn Ontology Modeling Components consider as Architectural Patterns the following ones: taxonomy, lightweight ontology and modular architecture. The patterns for re-engineering non-ontological resources deal only with taxonomies and lightweight ontologies.

Moreover, the patterns for re-engineering non-ontological resources define the transformation process but they do not provide an algorithm neither an implementation of the process. It is planned to include the algorithms and implementations later on in a framework which will implement the transformation process. A section to generate ontologies following the Linking Open Data[17] recommendations is also expected to be included.

Next, we present the proposed template of the NeOn project used to describe the patterns for re-engineering non-ontological resources (PR-NOR), adapted from the tabular template for ontology design patterns of [77].

The template adapted and the meaning of each field is shown in Table 2.3.

[17] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

Slot	Value
General Information	
Name	Name of the pattern
Identifier	An acronym composed of component type + abbreviated name of the component + number
Component Type	Component Type Pattern for Re-engineering Non-Ontological Resource (PR-NOR)
Use Case	
General	Description in natural language of the re-engineering problem addressed by the pattern for re-engineering NOR
Example	Description in natural language of an example of the re-engineering problem.
Pattern for Re-Engineering Non-Ontological Resource	
INPUT: Resource to be Re-engineered	
General	Description in natural language of the NOR
Example	Description in natural language of an example of the NOR
Graphical Representation	
General	Graphical representation of the NOR
Example	Graphical representation of the example of the NOR
OUTPUT: Designed Ontology	
General	Description in natural language of the ontology created after applying the pattern for re-engineering the NOR
Graphical Representation	
(UML) General Solution Ontology	Graphical representation, using the UML profile [BH06], of the ontology created for the NOR being re-engineered.
(UML) Example Solution Ontology	Example showing a graphical representation, using the UML profile [BH06], of the ontology created for the NOR being used
PROCESS: How to Re-engineer	
General	Description in natural language of the general re-engineering process, using a sequence of activities.
Example	Description in natural language of the re-engineering process applied to the NOR example, using the above sequence of activities.
Relationships (Optional)	
Relations to other modeling components	Description of any relation to other PR-NOR patterns or other ontology design patterns.

Table 2.3 Pattern for Re-engineering Non-Ontological Resource Template [77]

Non-ontological Resources Re-engineering Process

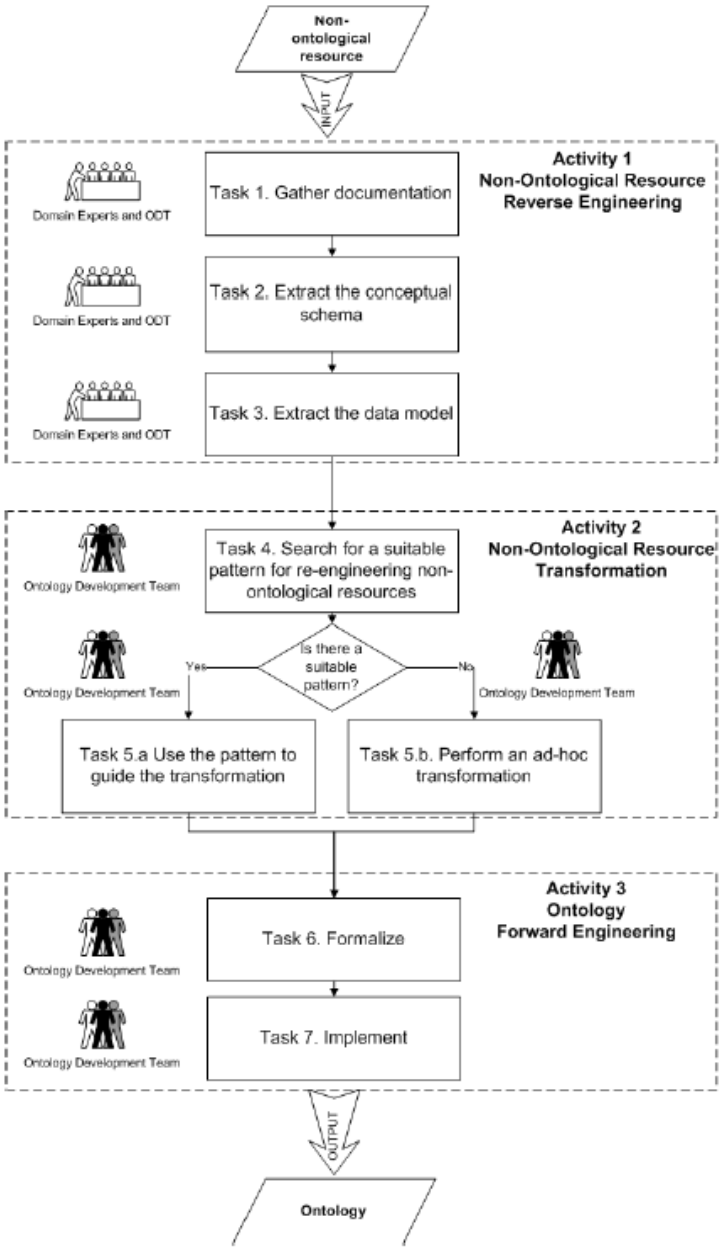

Figure 2.20 Re-engineering process for Non-Ontological Resources [78]

The non-ontological resource re-engineering process consists of the activities depicted in the figure 2.20. This process is based on the one described in [83]. The NeOn project follows the same process, but with the use of the set of patterns for re-engineering non-ontological resources described here.

- 1. Non-Ontological Resource Reverse Engineering, whose goal is to analyze a non-ontological resource to identify its underlying components and create representations of the resource at the different levels of abstraction (design, requirements and conceptual).
    - Task 1. Gather documentation. The goal of this task is to search and compile all the available documentation about the non-ontological resource including purpose, components; data model and implementation details.
    - Task 2. Extract the conceptual schema of the non-ontological resource. The goal of this task is to identify the schema of the non-ontological resource including the conceptual components and their relationships. If the conceptual schema is not available in the documentation, the schema should be reconstructed manually or by using a data modeling tool.
    - Task 3. Extract the data model. The goal of this task is to find out how the conceptual schema of the non-ontological resource and its content are represented in the data model. If the non-ontological resource data model is not available in the documentation, the data model should be reconstructed manually or by using a data modeling tool.
- 2. Non-Ontological Resource Transformation, whose goal is to generate a conceptual model from the non-ontological resource. They propose the use of Patterns for Re-engineering Non-Ontological Resources (PR-NOR) to guide the transformation process.
    - Task 4. Search for a suitable pattern for re-engineering non-ontological resource. The goal of this task is to find out if there is any applicable re-engineering pattern to transform the non-ontological resource into a conceptual model. To search for a suitable pattern for reengineering non-ontological resource the NeOn library of patterns can be used, according to the non-ontological resource, the data model, and the semantics of the relations between the non-ontological resource entities. First, the non-ontological resource type has to be identified. Second, the internal data model of the non-ontological resource has to be identified as well. Third, the semantics of the relations between the non-ontological resource entities have to be identified, these semantics can be a)subClassOf, b) an ad-hoc relation like partOf or c) a mix of subClassOf and ad-hoc relations. Finally, a pattern for re-engineering non-ontological resources can be searched according to the type of non-ontological

resource, the internal data model and the semantics of the relations between the non-ontological resource entities.

- Task 5.a. Use patterns for re-engineering to guide the transformation. The goal of this task is to apply the re-engineering pattern obtained in task 4 to transform the non-ontological resource into a conceptual model. If a suitable pattern for re-engineering non-ontological resource is found then the conceptual model is created from the non-ontological resource following the procedure established in the pattern for re-engineering.
- Task 5.b. Perform and ad-hoc transformation. The goal of this task is to set up an ad-hoc procedure to transform the non-ontological resource into a conceptual model, when a suitable pattern for re-engineering was not found. This ad-hoc procedure may be generalized to create a new pattern for re-engineering non-ontological resource.

- 3. Ontology Forward Engineering, whose goal is to generate the ontology. We use the ontology levels of abstraction to depict this activity because they are directly related to the ontology development process.
  - Task 6. Formalize. The goal of this task is to transform the conceptual model obtained in task 5.a or 5.b into a formalized model, according to a knowledge representation paradigm as description logics, first order logic, etc.
  - Task 7. Implement. The goal of this task is the ontology implementation in an ontology language.

2.4.6. General Conclusions

Undoubtedly, the re-engineering of non-ontological resources constitutes a process that can save us effort when approaching the problem of modeling existing knowledge into ontologies. In this section we have seen how to categorize the resources we may use, and which existing methods approach the problem of performing the effective transformation. In chapter 3, we define more precisely which data sources we can consider in the specific case of our methodology, and what decisions do we have to make in order to properly re-use them.

The NeOn approach provides a good view of the general process of re-engineering non ontological resources, which always involves the understanding of the underlying conceptual schema and data model, and the respective re-implementation into an ontology. As we have seen, this process may be supported by existing ontologies, or can be a full transformation of the resource into an ontology schema or ontology instances. In our problem, it is expectable that we deal with both situations, as we may use the core set of ontologies to support and “guide” the transformation, but model the domain specific ontology schema with some independency.

2.5. Relational Databases to RDF

Most of the data in today's world is stored in relational databases, in diverse RDBMS's. For example, as reported in [17], about 77.3% data on the current Web is stored in relational databases. On the other hand, in order to achieve the Web of Data envisioned by Tim Berners-Lee, there is a strong need to access and publish that kind of data in a Semantic Web fashion. As such, the achievement of interoperability between relational databases has been the focus of a body of research work in diverse domains, and has led to the implementation of generic mapping tools as well as domain-specific applications.

In organizations, the current scenario is very similar, with companies keeping a large amount of their data and having much of their services relying in RDBM's. As such, in the specific context of this project, this particular kind of resource is expectable to constitute a valuable source of knowledge for constructing domain-specific ontologies.

Besides these facts, we also have to consider that in this project we have, on one hand, a repository of communications and meta-data associated to them, and on the other hand, a set of ontologies with the mission to contextualize and provide classification means to the communications. Regarding the communications repository, we are not sure yet in which technologies it will rely upon; however, the XEO platform itself relies in the relational model, and it is a strong possibility that we have to deal with mappings between the two worlds, for these matters.

2.5.1. General Approaches

A W3C Working Group has been created with the specific goal of studying the definition of a standard language to map relational databases to RDF. The first stage of work of this group led to the formation of the RDB2RDF Incubator Group. This group concluded its work in February 2009, having produced two deliverables: a preliminary survey [69] documenting the techniques, tools and applications already existing in this field, and a RDB2RDF XG Final Report[18]. The RDB2RDF XG Final Report recommended that the W3C initiates a Working Group to standardize a language for mapping relational database schemas to RDF and OWL, from which has emerged the RDB2RDF Working Group[19].

To date, most of such mappings have been done on an ETL (Extract Transform Load) basis. The ETL implementations such as the application described by Byrne [16], also called "RDF dump", use a

[18] http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf-20090126/
[19] http://www.w3.org/2009/08/rdb2rdf-charter.html

batch process to create the RDF repository from RDB using mapping rules, and imply physically storing triples produced from relational data in an RDF store.

On the other hand, On-demand mapping is a more dynamic approach; it means translating a SPARQL query into one or more SQL queries at query-time, evaluating these against an automatic mapping generation; with respect to this kind of mapping generation, Tim Berners-Lee discussed [9] a set of mappings between RDB and RDF namely:

- A RDB record is a RDF node
- The column name of a RDB table is a RDF predicate
- A RDB table cell is a value

Many systems leverage these mappings to automatically generate mappings between RDB and RDF with the RDB table as a RDF class node and the RDB column names as RDF predicates. An example of this approach is the Virtuoso RDF View [5] that uses the unique identifier of a record (primary key) as the RDF object, the column of a table as RDF predicate and the column value as the RDF subject. Other examples of similar tools are D2RQ [10] (D2RQ also allows users to define customized mappings) and R2O [5].

2.5.2. D2RQ

D2RQ [10] provides an integrated environment with multiple options to access relational data including "RDF dumps" in RDF/XML or N-Triples, Jena and Sesame API based access, and SPARQL endpoints on D2RQ Server.

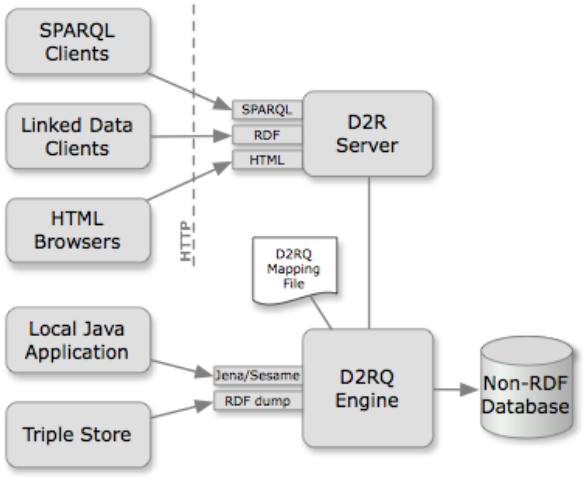

Figure 2.21 D2RQ Architecture [10]

The mappings may be automatically generated by an application helper or defined by the user, thereby allowing the possibility to incorporate domain semantics in the mapping process, although there are some limitations to this as described in [33]. They are expressed in a specific "declarative mapping language"; the language itself is expressed in RDF, and typically written down as an N3 file.

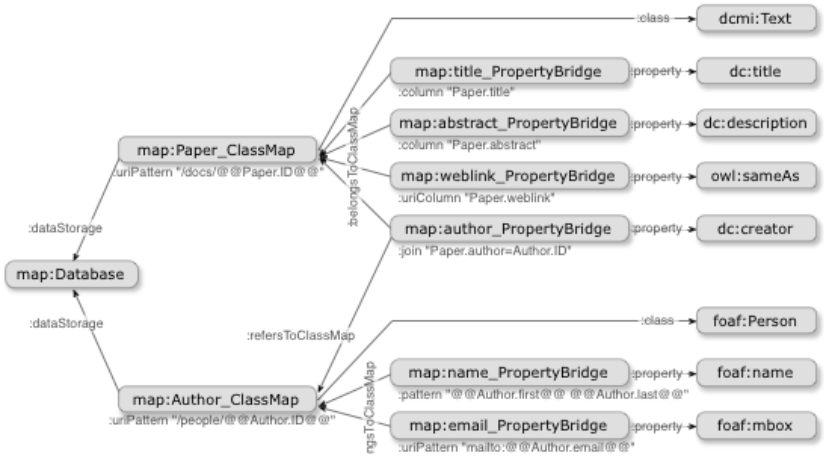

Figure 2.22 D2RQ RDF Graph Mapping Example [10]

An example of a simple D2RQ mapping rendered as an RDF graph is shown in figure 2.22: ClassMaps define a set of resources, and how these resources are identified. PropertyBridges take values from the database and attach them as properties to these resources, or link to other resources.

Basic ClassMaps and PropertyBridges can be further refined. This allows us to express conditional mappings, N:M joins, value translation tables, and to include custom value transformation functions. A full description of the mapping language can be found in the D2RQ manual.

According to the authors, D2RQ has been used with databases that range from hundreds of thousands to a few million records, and the performance varies depending on the access method. It is reported to perform reasonably well for basic triple patterns (similar to equivalent hand-written SQL queries) but suffers when SPARQL features such as FILTER or LIMIT are used, which is due to limitations in the implementation of D2RQ's SPARQL-to-SQL rewriting engine. Those issues are expected to be addressed in the next releases of the engine.

Until now, the compatible (tested and reliable) RDBMS's are Oracle, MySQL, PostgreSQL, MSSQL and Access.

Conclusions

D2RQ is a solid and robust tool that fulfills the purpose of simple mapping, although having some limitations, namely:

- Lack of integration of multiple databases or other data sources.
- D2RQ is read-only, without any kind of CREATE/DELETE/UPDATE operations.
- Lack of inference. For example, it defines that a certain node as a specific rdf:type, but does not infer that that type has rdfs:Class type. It would be desirable to have some inference (datatypes, basic RDFS, owl:sameAs).

2.5.3. Virtuoso RDF Views

Virtuoso[20] [11] is a commercial application that provides a virtual database integration middleware that integrates transparently different data sources, includes a web services development platform, among other things.

One of the Virtuoso components is Virtuoso RDF Views in which, once again, the approach followed is table to class (RDFS class) and column as predicate approach to create virtual RDF Views. It takes into consideration special cases such as whether a column is part of the primary key or foreign key. The foreign key relationship between tables is made explicit between the relevant classes representing the tables.

The RDB data is represented as virtual RDF graphs without physical creation of RDF datasets. Virtuoso RDF views are composed of "quad map patterns" that define the mapping from a set of RDB columns to triples. The quad map pattern is represented in the Virtuoso meta-schema language, which also supports SPARQL-style notation. The mapping is dynamic; consequently changes to the underlying data are reflected immediately in the RDF representation.

The main part of a quad map pattern is the declaration of four quad map values, with each declaration specifying how to calculate the value of the corresponding triple field from the SQL data; the IRI class "*product_iri*" is used to map a product's ID from some table into an IRI.

```
<code>graph <http://www.openlinksw.com/oplweb/>
subject p:product_iri (oplweb2.oplweb.product.product_id)
predicate p:description
object oplweb2.oplweb.product.product_description</code>
```

[20] http://virtuoso.openlinksw.com/

Virtuoso's meta-schema description language also supports a SPARQL-style notation; so, the above example can be written more concisely as:

```
<code>graph <http://www.openlinksw.com/oplweb/>
{
p:product_iri (oplweb2.oplweb.product.product_id) p:description
oplweb2.oplweb.product.product_description .
}</code>
```

Now, given a table of products, a possible example of a mapping can be:

```
<code>prd:Product a rdfs:Class ;
  rdfs:label "Product" ;
  rdfs:comment "An OpenLink product" .

prd:product_id a rdf:Property ;
  rdfs:domain prd:Product ;
  rdfs:range xsd:string ;
  rdfs:label "product id" .

prd:product_description a rdf:Property;
  rdfs:domain prd:Product ;
  rdfs:range xsd:string ;
  rdfs:label "product description" .

prd:product_category a rdf:Property ;
  rdfs:domain prd:Product ;
  rdfs:range prdc:ProductCategory ;
  rdfs:label "product category id" .

prd:product_format a rdf:Property ;
  rdfs:domain prd:Product ;
  rdfs:range prdf:ProductFormat ;
  rdfs:label "product format" .</code>
```

In this example, we can see that the relationships between the tables linked by foreign keys can be made explicit by referencing the relevant classes directly from the Product class, in effect dereferencing the foreign keys. To this end, the attributes *product_category* and *product_format* have ranges *prdc:ProductCategory* and *prdf:ProductFormat* respectively, where *ProductCategory* and *ProductFormat* are RDFS classes corresponding to the entities described by the *product_category* and *product_format* tables.

The end-user application provides a user-oriented interface, in which is possible to choose among several databases and ontologies and decide to generate these kind of mapping specifications automatically or establish mappings between them in a click and drag manner.

Conclusions

Virtuoso is a robust commercial solution that works well with the issue of heterogeneity between different data sources, but still has some limitations on the "semantic gain" on the mapping of relational databases to RDF, as it follows the automatic mapping approach.

2.5.4. R2O

R2O [5] is an XML based declarative language to express mappings between RDB elements and ontologies implemented in RDF(S) or OWL. Due to the language fully declarative nature, R2O mappings can be used to automatically "detect inconsistencies and ambiguities" in mapping definitions. A mapping definition can also be used to verify the integrity of parts of a DB according to an ontology, applying the ontology's axioms to the database elements.

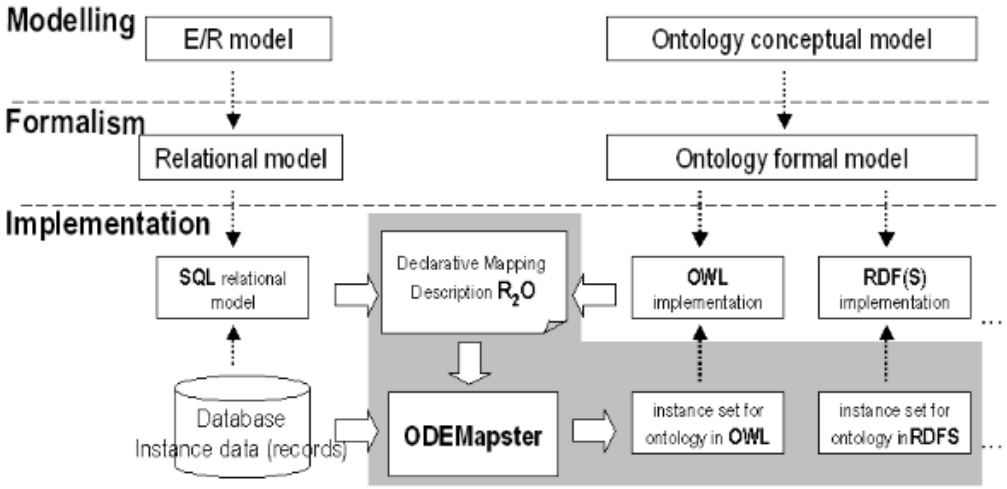

Figure 2.23 R2O Architecture [5]

To improve readability, in the following examples of the meta-language elements, it is used a compact pseudo XML syntax, where opening tags are indicated by bold text, grouping of sub-content is indicated by indentation and closing tags are omitted.

A DB schema description (dbschema-desc) provides a copy of the main structural elements in the DB's SQL schema. It can be extracted automatically from the source DB and the only elements that need to be added manually are the implicit references. The DB schema definition is a "sort of internal" representation of a DB and is needed to restrict the domain and range of the components

of a mapping definition. Some technical information about the DB (url, port, user/pwd, etc.) necessary for implementation is omitted for the sake of clarity.

A dbschema-desc consist of the name of the DB (name), a NL description of the schema (documentation?), and one or more table descriptions (hasTable+), where each DB table is described by means of (table-desc).

A table description (table-desc) provides a description of a DB table. A table-desc consists of a name of the table (name), the type of the table (tableType) that can be either system table, user table or view, its NL description (documentation?), or a set of column descriptions (column-description+).

A column description (column-description) can be either a key column (keycoldesc), a foreign key column (forkeycol-desc) or a non key column (nonkeycol-desc). Any of them consist of a name for the column (name), a type for the data it contains (ColumnType), its natural language (NL) description (documentation?), and the key column referred (refers-to?) if it is a foreign key forkeycol-desc.

Sometimes implicit references exist between columns that are not explicitly declared as such in the DB schema, in this case we also have the referred column (implicitlyrefers-to?). If a DB is correctly defined it should not be necessary. We provide this as a solution for badly designed DB schemas.

As an example of use of a DB schema description, consider:

```
dbschema-desc
 name FISUB
 has-table
  name FundingOpps
  documentation "Stores funding info"
  keycol-desc
   name FundingOpps.FundId
   columnType integer
   documentation "Identifies a f.o."
  nonkeycol-desc
   name FundingOpps.FundTitle
   columnType string
  forkeycol-desc
   name FundingOpps.FundSector
   columnType integer
   refers-to Sector.Id
   documentation "Points at Sector"
 has-table
  name Sector
```

```
documentation "Productive sectors."
keycol-desc
 name Sector. Id
 columnType integer
```

A concept mapping definition associates the name of a class in the ontology to a description of how to obtain it from the DB. A conceptmap-def, consists of the following components:

- The identifier of a concept (URI of the class) in the target ontology (name)
- NL description of the rationale behind the concept mapping (documentation?).
- One or more columns that uniquely identify (identified-by+) the concept in the DB. Each column is described with the column-desc element previously defined.
- A pattern expressed in terms of transformations describing how URIs (uri-as+) for the new instances extracted from the DB are generated. URIs are normally obtained from the key columns after applying some transformations. The absence of this element generates anonymous instances.
- A concept in the ontology is described (described-by*) by a set of attributes and relations.
- A mapping is only applied under certain conditions. The element applies-if? contains a conditional expression describing these conditions. In other words, it specifies the subset of values from the DB that is transformed to populate this concept.
- Sometimes more than one table is implied in the definition of a concept mapping, and join operations are needed. The optional (joins-via?) element describes how these tables are joined in case they use "implicit joins". If this information can be obtained from the DB schema description (only foreign keys are used for joins) the joins-via? element is omitted. The rationale behind this element is that the mapping designer might want to specify a particular join, not valid in all cases but useful in the context of a particular concept mapping (sort of a “specific local join”). The information in the joins-via? element can overwrite that in the DB schema definition or can be added to it. It contains a join-list which consists of a group of one or more join elements, each of them describing a pair of columns (hasJoin+) and a flag (overwrites) indicating if the join list is to be used together with the ones defined in the DB schema description or if we want them to be overwritten. Columns are not necessarily key nor foreign key columns.

As an example of concept mapping definition, consider:

```
conceptmap-def
 name Customer
 identified-by Users.userID
 uri-as
  <transformation>
 applies-if
  <cond-expr>
documentation Select all rows from table Users with 'true' in column
isPreferential.
```

Engine

The ODEMapster engine uses a R2O document to either execute the transformation in response to a query or in a batch mode to create a RDF dump. ODEMapster is both the R2O processor engine and a Neon Toolkit[21] plugin, providing a graphical interface to make correspondences between the ontologies opened in a project and a database.

Conclusions

R2O and ODEMapster are useful mechanisms, even more for the reason of being part of the NeON project and the possibility of using it as an Eclipse plugin and integrate it with other development tools that the toolkit supplies, at the knowledge acquisition level (text extraction for example) or the creation, edition and validation level. It also has an expressive language for the intended means of mapping.

2.5.5. General Conclusions

This survey leads to the conclusion that there exist some solid and interesting tools for the issue of converting from RDB to RDF.

These are mainly trivial automatic conversions, but nonetheless, they can be a good starting step to the initial setup of domain ontologies. With the possibility of expert's interference in the automatic mapping definitions, it is expectable that these tools are complemented with methodologies that allow for taking better advantage of the underlying data model and semantic meaning, in order to gain the domain specific description enrichment. It may also be interesting to keep an eye in the W3C working group[22] next developments, as they are trying to be active, with weekly meetings and continuous effort on the achievement of new alternatives and a possible

[21] http://neon-toolkit.org/
[22] http://www.w3.org/2001/sw/rdb2rdf/wiki/Main_Page

standardization, as each implementation has its own specificity and mapping language, although there are some equivalences between them.

Regarding the equivalences between the languages, Virtuoso, for example, starts with the triple and says which tables and columns will produce it. Triplify[23] starts with the SQL statement and says what triples it produces. These are fairly equivalent. For the web developer, the latter is likely more self-evident, while the former may be more compact and have less repetition. Another example is that D2RQ provide properties for conditional mappings based on queries conditions, while Virtuoso can include those in the name aliases.

[23] http://triplify.org/

2.6. Ontology Learning from Unstructured Text

The term ontology learning was originally coined by Alexander Madche and Steffen Staab [52], and can be described as the acquisition of a domain model from data.

Ontology learning from unstructured texts is based on the use and analysis of text corpora. A corpus of texts is a set of texts that should be representative of a specific domain, prepared to be processed by a computer, and accepted by domain experts [58]. In this project, we are dealing with classification and categorization of communications, and one of the sources of the knowledge model we want to contextualize may be the communications by themselves, and that communications (e-mails, faxes, etc.) are, in its essence, text.

In this section the specific details of text extraction methods and algorithms are not deeply approached, but instead a superficial view of the overall approaches is given, and it is explained how they can be used in our project for ontology learning purposes.

2.6.1. Named Entities Recognition

Named entity recognition (NE) from textual sources has been a subject of research for the last two decades, and it is crucial for many Natural Language Processing (NLP) tasks, such as information extraction, which rely on the initial extraction of entities before identifying relations, co-reference, etc [83]. Traditional methods and tools for NE have been widely described and discussed (e.g. in [55]), relying for example in rule-based or statistical methods.

Named Entities Recognition with GATE

GATE, the General Architecture for Text Engineering [22], is a framework and graphical environment providing support for a variety of language engineering tasks. It includes a vanilla information extraction system, ANNIE, and a large number of plug-ins for various tasks and applications, such as ontology support, information retrieval, support for different languages, WordNet [25], machine learning algorithms, and so on.

A brief summary of the components and methods used for rule-based information extraction in GATE is presented below:

- The tokenizer splits text into simple tokens, such as numbers, punctuation, symbols, and words of different types (e.g. with an initial capital, all upper case, etc.), adding a "Token" annotation to each. It does not need to be modified for different applications or text types.

- The sentence splitter is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. Both the splitter and tagger are generally domain and application independent.
- The tagger is a modified version of the Brill tagger [107], which adds a part-of-speech (POS) tag as a feature to each Token annotation. Neither the splitter nor the tagger is a mandatory part of the NE system, but the annotations they produce can be used by the semantic tagger (described below), in order to increase its power and coverage.
- The gazetteer consists of lists such as cities, organizations, days of the week, etc. It contains some entities, but also names of useful key words, such as company designators (e.g. "Ltd."), titles (e.g. "Dr."), etc. The lists are compiled into finite state machines, which can match text tokens.
- The semantic tagger (or JAPE transducer) consists of hand-crafted rules written in the JAPE pattern language [89], which describe patterns to be matched and annotations to be created. Patterns can be specified by describing a specific text string or annotation (e.g. those created by the tokenizer, gazetteer, document format analysis, etc.).
- The orthomatcher performs coreference, or entity tracking, by recognizing relations between entities. It also has a secondary role in improving NE recognition by assigning annotations to previously unclassified names, based on relations with existing entities.

In terms of adapting to new tasks, the processing resources in ANNIE fall into two main categories: those that are domain-independent, and those that are not. For example, in most cases, the tokenizer, sentence splitter, POS tagger and orthographic coreference modules fall into the former category, while resources such as gazetteers and JAPE grammars need to be modified according to the application. Similarly, some resources, such as the tokenizer and sentence splitter, are largely language-independent (exceptions may include some Asian languages, for example), and some resources are more language dependent, such as gazetteers. The feasibility of reusing grammars and other components for named entity recognition tasks is discussed at length in [64]; the conclusions drawn were very positive given 4 factors: use of a flexible and robust architecture (such as GATE), use of an appropriate rule formalism (such as JAPE), the nature of the application(s) in question, and the languages used.

2.6.2. Patterns for entity recognition

Traditional Named Entity recognition, and even ontology-based information extraction applications in GATE, rely on a fairly small set of patterns which aim to identify the relevant entities in text. These rely largely on gazetteer lists, which provide all or part of the entity in question, in combination with linguistic patterns (see for example [59] for a discussion on the importance of gazetteers in pattern-based NE recognition). For example, a typical rule to identify a person's name

consists of matching the first name of the person with an entry in the gazetteer (e.g. "John" is listed as a possible first name), followed by an unknown proper noun (e.g. "Smith", which is recognized as a proper name by the POS tagger). Most patterns include some combination of a proper noun or word with an initial capital letter and either some gazetteer entry or linguistic feature.

However, identifying ontological concepts and/or relations requires a slightly different strategy. While we can still make use of known lists of terms (either via a gazetteer or by accessing the class, instance and property labels in an existing ontology), this is often not sufficient for a variety of reasons:

- The concept may not be in the ontology already
- The concept may exist in the ontology only as a synonym or linguistic variation (singular instead of plural, for example)
- The concept may be ambiguous
- Only a super class of the concept may exist in the ontology

Therefore, the use of linguistic patterns and also contextual clues is needed, rather than relying on gazetteer lists as with traditional recognition.

Text2Onto [20] performs synonym extraction on the basis of patterns. It combines machine learning approaches with basic linguistic processing such as tokenization or lemmatization and shallow parsing. It can be used as a plug-in for the Neon Toolkit, of the NeOn project, and the method it follows is explained in the respective section of this document.

In [83], three sets of patterns are mentioned which can help us identify concepts, instances and properties to extend an ontology: the Hearst patterns the Lexico-Syntactic Patterns developed in NeOn corresponding to Ontology Design Patterns and some new contextual patterns defined by us which take into account contextual information. These set of patterns are described as follows:

Hearst Pattern-based Extraction

The Hearst patterns are a set of lexico-syntactic patterns that indicate hyponymic relations [40], and have been widely used by researchers. Relations at the conceptual level are recognized from sequences of words in the text that follow a given pattern.

For example, in English a pattern can establish that if a sequence of n names is detected, then the n-1 first names are hyponyms of the n^{th}. According to this pattern, the term "Portugal location" could be used to obtain the hypononymy relationship between the term "Portugal location" and the

term "location". This relation at the linguistic level as the //Subclass-Of// relation between the concept associated to the term "Portugal location" and the concept associated to the term "location".

Typically, these patterns achieve a very high level of precision, but quite low recall: in other words, they are very accurate but only cover a small subset of the possible patterns for finding hyponyms and hypernyms [83].

Lexico-Syntactic Patterns

In these set of patterns, for each relation, there are several possible patterns: mostly these can be combined into a single rule in a grammar, but they are separated here for ease of understanding. The grammars can be written, for example, in JAPE [89].

In the following rules, <sub> and <super> are like variable names for the subclasses and superclasses to be generated; CN means class of, group of, etc.; CATV is a classification verb2; PUNCT is punctuation; NPlist is a conjoined list of NPs ("X, Y and Z").

```
  * 1. Subclass rules
<code>NP<sub> be NP<super>
NPlist<sub> be CN NP<super>
NPlist<sub> (group (in|into|as) | (fall into) | (belong to)) [CN]
NP<super>
NP<super> CATV CV? CN? PUNCT? NPlist<sub></code>
```

Example: "Frogs and toads are kinds of amphibian".

```
  * 2. Equivalence rules
<code>NP<class> be (the same as|equivalent to|equal to|like) NP<class>
NP<class> (call | denominate | (designate by|as) | name)
NP<class> (where the verb is passive)
NP<class> have (the same|equal) (characteristic | feature | attribute |
quality | property) as NP<class></code>
```

Example: "Poison dart frogs are also called poison arrow frogs"

```
  * 3. Properties
<code>NP<class> have NP<property>
NP<instance> have NP <property></code>
```

Example: "Birds have feathers".

While these patterns are quite productive (for example X is a Y), most of them are potentially ambiguous and susceptible to over generation. For example, in the following sentence:

Mistakenly, some artists and writers have penguins based at the North Pole.

The patterns produce the inference that writers have penguins, recognizing penguin as a property of writer. Clearly it is ludicrous that every expression of the form X has Y should result in the relation Y is a property of X. The difficulty is deciding where to draw the line between acceptable patterns and those that just overgenerate.

Contextual patterns

These patterns are based on the definition of a set of rules designed to make use of contextual information in the text about known entities already existing in the ontology (unlike the lexico-syntactic patterns which assume no previous ontological information is present). These rules are used in conjunction with the OntoRootGazetteer plug-in in GATE, which enables any morphological variant of any class, instance or label in the ontology to be matched with (any morphological variant of) any word or words in the text. Which elements from the ontology are to be considered (e.g., whether to include properties, and if so which ones) is determined in advance by the user when setting up the application. Initially we use the following rules to find new classes and instances:

- 1. Add a new subclass: (Adj|N) NP<class> ! NP<subclass>. This matches a class name already in the ontology preceded by an adjective or noun, such as adjective preceding a known type of fish, which we assume is a more specific type. For example, when we encounter the phrase ". . . Japanese flounder. . ." in a text and flounder is already in the ontology, we add Japanese flounder as a subclass of flounder.
- 2. Add a new class (a more generic version of the Hearst patterns). Here we postulate that an unknown entity, amidst a list of known entities, is likely to be also an entity of the same type. For example, if we have a list of classes of fish, and there is an unknown noun phrase in the list, we can presume that this is also a class of fish. To decide where to add this new class in the ontology, we can look for the Most Specific Common Abstraction (MSCA) of all the other items in the list (i.e. the lowest common super class of all the classes in the list) and add the new entity as a subclass of this class. However, this has not currently been implemented due to the complexities of implementation in NEBOnE, but is planned for the future. Therefore, we just add it as a new subclass of Thing (top level) and leave it to the user to move it to a more appropriate place. As an example, consider: "Hornsharks, leopard sharks and catsharks can survive in aquarium conditions for up to a year or more", where hornshark and leopard shark are classes in the ontology and catshark is unknown; in this case, we can recognize

catshark as a subclass with the same parent as that of hornshark and leopard shark, in this case shark.

- 3. Add an alternative name as a synonym: a name followed by an alternative name in brackets is a very common pattern in some kinds of text. For example in texts about flora and fauna we often get the common name followed by the Latin name in brackets, as in the following sentence: “Mummichogs (Fundulus heteroclitus) were the most common single prey item”. If we know that one of the two NPs is a class or instance in the ontology, we can predict fairly accurately that the other NP is a synonym.

2.6.3. Maedche and colleagues’ method

Maedche and colleagues’ method [45] assumes that documents from a domain describe most of the domain concepts and relations to be included in an ontology as well as the domain terminology. This method proposes to learn the ontology using as a base a core ontology (SENSUS, WordNet, etc.), which is enriched with the learned concepts.

New concepts are identified using natural language analysis techniques over the resources previously identified by the user. The resulting ontology is pruned and then focused on a specific domain by means of several approaches based on statistics. Finally, relations between concepts are established applying learning methods. Such relations are added to the resulting ontology.

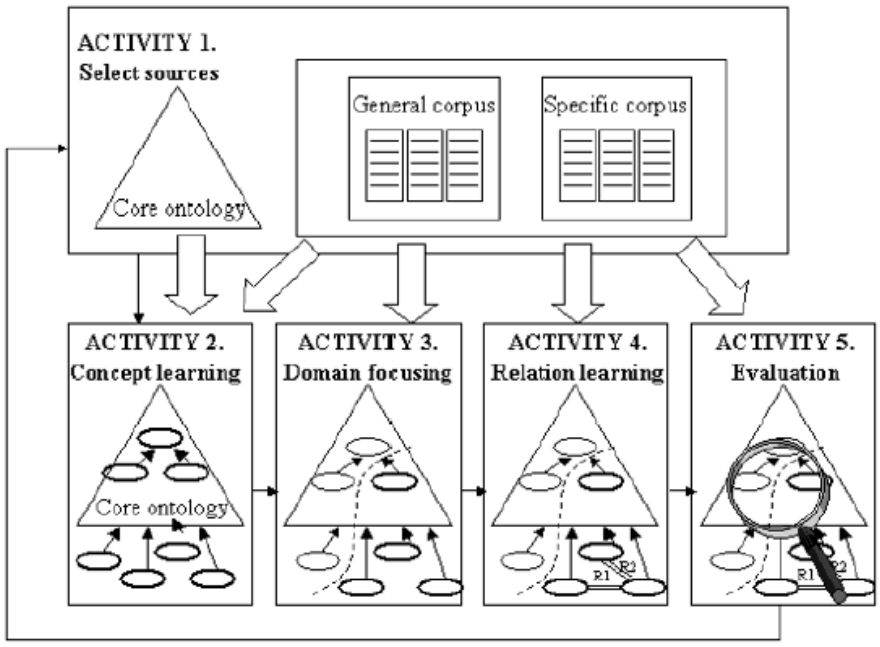

Figure 2.24 Activities following in Maedche and colleagues method [32]

The activities proposed in this method, as shown in figure 2.24, are:

- **Activity 1.** Select sources. In this method, sources are either ontologies or documents. The process starts with the selection of a core ontology, which is used as a base in the learning process. This ontology should contain generic and domain specific concepts. In this first activity the ontologist should specify which documents should be used in the steps to follow to refine and extend the previous ontology. By its own nature, documents are heterogeneous in their formats and contents. They can be free text, semi-structured text, domain text, and generic text documents. Documents will make up two corpora: one with domain specific terms, and another with general terms.
- **Activity 2.** Concept learning. Its goal is to acquire new generic and domain specific concepts. Both types of concepts are extracted from texts by means of mainly natural language processing (NLP) tools that use pattern-based extraction and conceptual clustering. The selection of the tools depends on the languages to be processed (Spanish, English, German, etc.). The method suggests linking the learned concepts to the core ontology using, above all, the Subclass-Of relation.
- **Activity 3.** Domain focusing. Its purpose is to prune the enriched core ontology and remove general concepts. The results of the analysis of the term frequency in the generic and the specific corpora are used to prune the ontology. The terms appearing more frequently in the domain-specific corpus than in the generic corpus should be proposed to the ontologist for deciding whether they should be kept in the whole enriched ontology.
- **Activity 4.** Relation learning. Ad hoc relations between concepts of the domain are learnt by means of pattern-based extraction and association rules.
- **Activity 5.** Evaluation. Its goal is to evaluate the resulting ontology (the core ontology enriched and pruned in the previous activities) and to decide whether it is necessary to repeat the process again.

This method has been applied inside the project On-To-Knowledge, supported by the tool Text2Onto, whose methodology was described in Section 2.3.5.

2.6.4. General Conclusions

Ontology learning from texts is one of the most well-known and studied categories of ontology learning approaches; however, and although in this work evaluation issues on these methods are not discussed, it is well accepted by the ontology learning community that they are far from foolproof and consensus. Another problem is that few of the existing approaches for text extraction are available for the Portuguese language, which is the main focus of the project, at this moment.

The idea of using text corpora for extracting terms in the context of this project cannot be discarded, given the potential knowledge residing on communications, but more research is needed on the techniques to determine its feasibility, and it would require more time effort that we cannot have for this thesis. The method presented by Maedche may provide a good framework to tackle this problem, as well as its implementation as a NeOn toolkit plugin.

Chapter 3
Proposed Solution

This chapter presents the design of the ontology repository, identifying requirements, discussing modeling issues and presenting the methodology proposed to extend it to a specific domain.

This chapter introduces the design of the ontology repository, starting with a list of general requirements for its usage. With those requirements in mind, the high-level architecture of the core set of ontologies for the XEO ECC platform is presented, followed by the steps that define how to extend them with specific domain knowledge.

The RDF examples illustrated are either presented in the RDF/XML serialization, or graphically, in the form of graphs, for the most complex or largest ones.

3.1. Requirements

Briefly reminding the ontology objectives enumerated in section 1, the main goals of the ontologies designed in this work are to provide effective means of representing the communications of an organization and the relevant data embedded in their content. This representation constitutes the basis of the automatic classification of communications and the search and navigation processes. The developers of these processes are the main contributors of the list of requirements the ontology must fulfill.

The list of requirements that guided the choice and design of the core set of ontologies is organized and presented in the Ontology Requirements Specification Document, as shown in table 3.1 and adapted from the template presented in section 2.3.5 of the state of the art.

	Ontology Requirements Specification Document Template
	Purpose
	The purpose of the ontology is to describe communications in the context of organizations.
	Scope
	The ontology should be able to properly describe all the metadata associated to the communications (subject, format, sender) as well as the organizations structure and specific domain “lingo”. This lingo must be expressed in terms of its semantic meaning, but also of its structure of classes and properties that may serve to populate instances.
	Implementation Language
	The ontologies should be implemented in RDF and OWL. This is not an explicit requirement, but since the first meetings with the company responsible for this project, it was clear that they expected that the ontology follow the current W3C recommendations for the Semantic Web.
	Intended End-Users
	User 1: Developer of automatic classification processes. User 2: Developer of search and navigation processes.
	Intended Uses
	Use 1a: Retrieve the list of the possible communications purposes from a certain domain, and their characterization Use 1b: Retrieve the list of all terms used in the lingo of a certain domain Use 1c: Retrieve the URI of a element, given a certain attribute specified (for example, retrieve the URI of a person, given a contact number, a person identification number, etc.) Use 2a: Retrieve the list of all persons from the organization, with all their respective attributes and

	roles within the organization Use 2b: Same as use 1a
	Ontology Requirements
	a. Non Functional Requirements
	The ontology must have the capacity of, if demanded, being expressed in more than one language. The XEO framework and the case study of this work are Portuguese-based, but the platform may be used for non-Portuguese customers in the future. The ontology must be maintained in a repository which allows it to be easily queried from an interface. The ontology must be easily extended to any domain.
	b. Functional Requirements: Groups of Competency Questions
	Besides the metadata of the communications, the organizations structure and the description of purposes, which are requisites common to all domains, the competency questions addressed in order to obtain a pre-glossary of ontology terms depends on the domain we are dealing with; in chapter 4, a pre-glossary of terms is obtained for the specific case study.

Table 3.1 ORSD Template –adapted from [87]

3.2. Core Set of Ontologies and Vocabularies

Regarding the core set of ontologies, we may divide the problem into three sub-tasks:

- Describing the organization and its people (the possible interlocutors of the communications)
- Describing the basic metadata associated to the communications
- Describing the context associated to the domain of the communications

The general model proposed is illustrated in figure 3.1, and each component and the way the vocabularies mentioned on the section 2.1 of the state of art were used to fulfill the requirements enumerated are explained in the following sections.

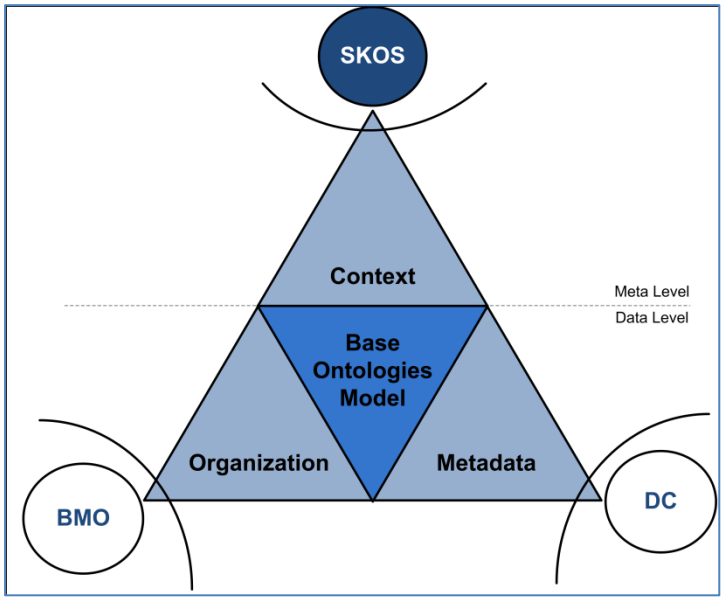

Figure 3.1 Core Set of Ontologies General Model

3.3. Ontology Levels

The Ontologies presented here can be distinguished in two different levels: the Meta level and the Data Level, represented by two distinct vocabularies.

The main difference between the two levels is that in the Meta level we are only defining the semantic meaning and relationship between its concepts. On the other hand, in the data level we are defining the structure and properties of classes that serve as types for individual instances. This difference is perceived more clearly in the examples provided further in this document.

The ontology elements belonging to the Meta level are defined using the “ecc-meta” prefix. In this level we have a glossary of the terms that belong to the specific business domain and we have the communications possible purposes.

The data level contains the modeling of products, services and the organization's structure. The elements belonging to the Data level are defined using the “ecc-data” prefix.

3.3.1. Motivation and Alternatives

Briefly reminding the goal of this work, we have to achieve a conceptualization of classes describing the organizations structures, products and services, and their relevant properties. This part of the problem can easily be modeled with a set of OWL classes.

But what about the whole project goals? We have to express the entire specific domain lingo in our ontologies, in order to obtain effective classification and searching results. This lingo is implicitly expressed in our modeling, but we need to make its whole semantically accessible; ontologies users must know the whole semantic meaning of every concept of the vocabulary, their definition, possible lexical labels and the way they are related.

One alternative to tackle this problem would be to keep it all in OWL. Instead of using a SKOS concept to define possible concept labels, we could use the SKOS documentation properties directly to document OWL Classes. However, there are a number of issues here. The SKOS labeling and documentation properties are currently defined as OWL Object and Data properties. This is partly due to a requirement that there must be sub-property relationships between some of these properties. OWL-DL only allows annotation assertions on classes, which means that the use of label

or documentation properties on classes results in an OWL-Full ontology. Representing the properties as OWL annotation properties would preclude the possibility of extending the properties through the definition of sub properties.

Another alternative would be to overlay SKOS with OWL, that is, defining the type of some node both as owl:Class and skos:Concept, or asserting that some OWL class and some SKOS concept are equivalent using the property owl:sameAs. Again, this would necessarily lead to an OWL Full representation, because an instance of skos:Concept might also be an instance of owl:Class.

The approach followed keeps both the worlds in a separate stream, avoiding an OWL Full representation and also allowing us to organize our ontology in such a way that keeps both our requirements on separate levels, which eases the user's queries "mental process".

3.3.2. The Meta Level

The specific domain terms gathered during the elicitation process are defined at this level as skos:Concepts. Every term has a formal meaning described through the skos:definition property, and their different syntactical forms are included with the skos:altLabel property, as shown in figure 3.2.

```
<skos:Concept rdf:about="&ecc-meta;some_term">
    <skos:prefLabel rdf:datatype="&xsd;string">Some Term</skos:prefLabel>
    <skos:definition rdf:datatype="&xsd;string">
        This is an example of a definition of a term.
    </skos:definition>
    <skos:inScheme rdf:resource="&ecc-meta;general-terms"/>
</skos:Concept>
```

Figure 3.2 skos:Concept - defining a term

As we can see in the example, we can define that some concept belongs to a conceptScheme, through the skos:inScheme property; this is a good practice, not only to allow a better organization of the different terms of our glossary, but also to provide a good mean to filtering future user queries.

Regarding the purposes of communications, at first, a purpose was simply modeled as a skos:Concept, that is, there existed a certain class ecc:meta-Purpose that was a subclass of skos:Concept, and it could establish a hierarchy starting with more general purposes, and descending into more specific ones, as depicted in figure 3.3.

```
<ecc-meta:Purpose rdf:about="&ecc-meta;drink_purpose">
    <skos:prefLabel rdf:datatype="&xsd;string">Drink something</skos:prefLabel>
    <skos:definition rdf:datatype="&xsd;string">
        This is an example purpose for drinkers
    </skos:definition>
</ecc-meta:Purpose>

<ecc-meta:Purpose rdf:about="&ecc-meta;drink_milk_purpose">
    <skos:prefLabel rdf:datatype="&xsd;string">Drink milk</skos:prefLabel>
    <skos:definition rdf:datatype="&xsd;string">
        This is an example purpose for milk drinkers
    </skos:definition>
    <skos:related rdf:resource="#milk"/>
    <skos:broader rdf:resource="&ecc-meta;drink_purpose" />
</ecc-meta:Purpose>

<ecc-meta:Purpose rdf:about="&ecc-meta;drink_chocolate_milk_purpose">
    <skos:prefLabel rdf:datatype="&xsd;string">Drink chocolate milk</skos:prefLabel>
    <skos:definition rdf:datatype="&xsd;string">
        This is an example purpose for chocolate milk drinkers
    </skos:definition>
    <skos:related rdf:resource="#chocolate"/>
    <skos:related rdf:resource="#milk"/>
    <skos:broader rdf:resource="&ecc-meta;drink_milk_purpose" />
</ecc-meta:Purpose>
```

Figure 3.3 - Purpose Hierarchy

From the result of a series of discussions with the people involved on the project, from both the ITDS and the FCT-UNL side, a purpose is now defined as a collection of concepts; as such, there exists a class ecc-meta:Purpose, which is a subclass of skos:Collection. When we define a purpose, we give it a formal semantic meaning with the label and definition properties, and specify the concepts that belong to the purpose definition (with the skos:member property).

We can also define a purpose argument, specified by the ecc-meta:argument property, which is a sub-property of skos:related. An argument is an element the classification process may find in a communication, and may need to identify and to instantiate it.

```
<ecc-meta:Purpose rdf:about="&ecc-meta;choco_milk_purpose">
    <skos:prefLabel rdf:datatype="&xsd;string">Drink a chocolate milk</skos:prefLabel>
    <skos:definition rdf:datatype="&xsd;string">
        This is an example purpose
    </skos:definition>
    <skos:member rdf:resource="&ecc-meta;chocolate"/>
    <skos:member rdf:resource="&ecc-meta;milk"/>
    <skos:member rdf:resource="&ecc-meta;drink"/>
    <ecc-meta:argument rdf:resource="&ecc-meta;mug"/>
</ecc-meta:Purpose>
```

Figure 3.4 - Purpose as a Collection

This approach, being more loosely modeled, allows both for the classification and the searching processes to gather the purposes information in a more direct and flexible fashion. For example, they can even still serialize a purpose as a hierarchy, as there exists an implicit hierarchy embedded in this representation, which is the one represented by the concepts that form the collection.

Moreover, it complies with an approach followed by both the classification and search and navigation processes, which is the Enhanced Topic-based Vector Space Model (eTVSM) [108]. Briefly, the eTVSM is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings.

Briefly, in this approach and on this project, a communication is represented by a vector:

$c_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$

Each dimension corresponds to a separate topic. If a topic occurs in the communication, its value in the vector is non-zero. A topic occurs in the communication if a communication involves a term that constitutes the topic.

The topics can also be related among themselves. Besides the intrinsic relation they have by being part of a same dimension, there can established hierarchies within the dimension. This may allow the establishment of some general purposes, or to derive more specific ones, depending on the hierarchy inherited from the domain.

The topic relations are expressed in a topic map. A topic map is a directed graph with topics as nodes; the graph edges assign super-, sub-topic relations between the topics.

Given these definitions, we can consider that the concepts of the domain in our ontology are equivalent to topics in a topic map. The terms assigned to topics are the labels assigned to the concepts that correspond to those topics. The edges of the topic map nodes are the relations of *skos:narrower* and *skos:broader* that exist between the concepts, and the dimensions they belong to are expressed as follows:

- A class ecc-meta:Dimension was created, being rdf:subClassOf skos:ConceptScheme
- A concept can belong to a dimension having the property inScheme associated to a certain dimension.
- If a concept belongs to a dimension, all the concepts that are below it in the SKOS hierarchy (that is, concepts which are narrower), also belong to the dimension.

This representation can be achieved through a SPARQL or SerQL CONSTRUCT query, and is transparent to the end-user (the classification developer).

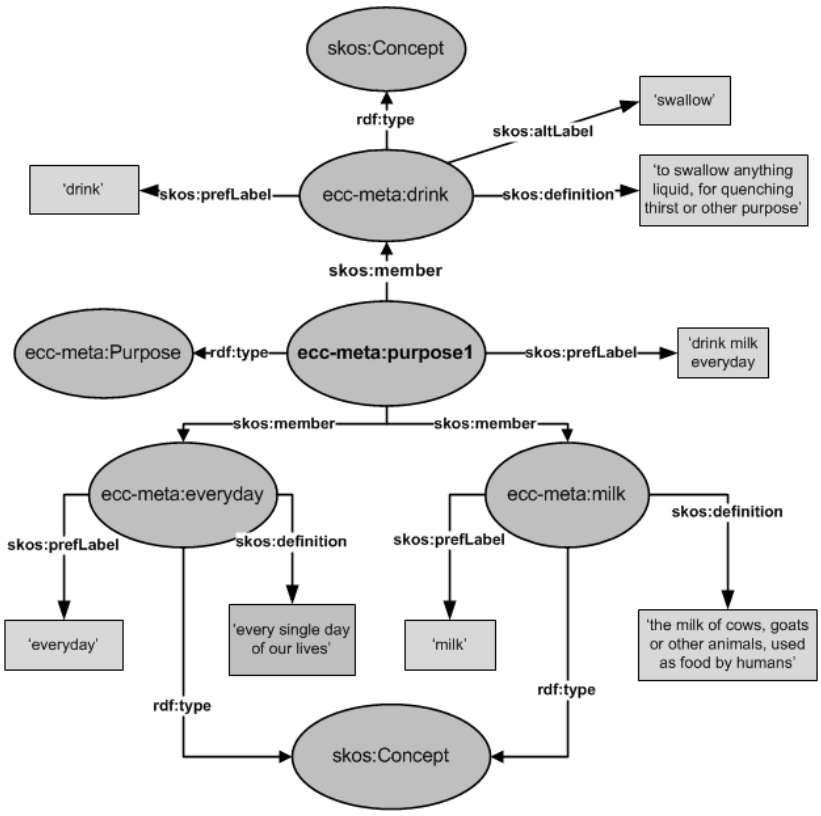

Figure 3.5 A Purpose and its Concepts

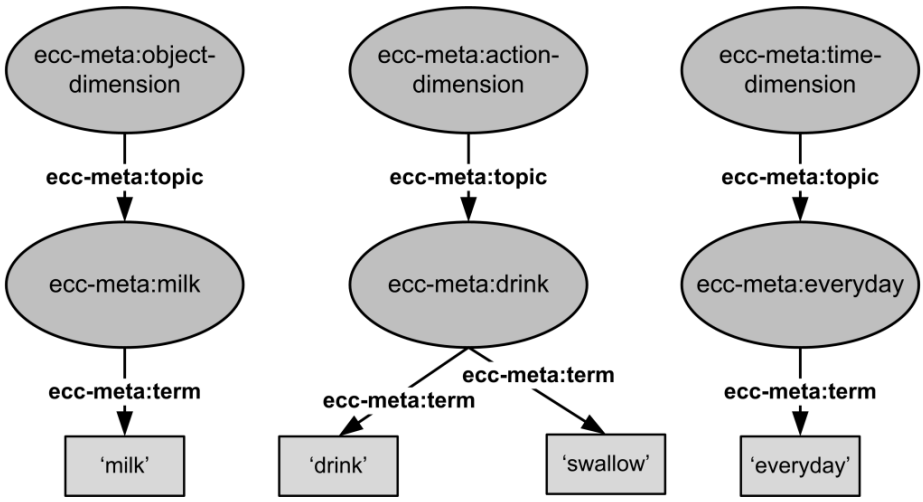

Figure 3.6 Topic Map

For example, given the purpose exemplified in picture 3.5, and the concepts that define that purpose, the corresponding topic map would be the one depicted in figure 3.6, and the purpose, as seen by the classification developer, would be as depicted in figure 3.7.

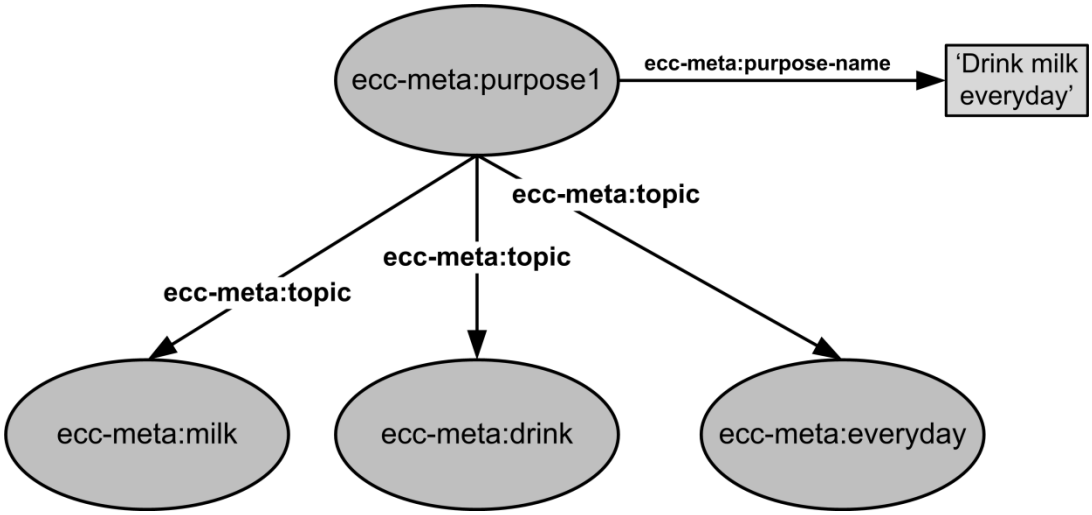

Figure 3.7 Purpose Serialization

Note that the dimensions used here are only examples; the dimensions specified in the ontology strongly depend on the specific domain we are working on, as we will see in chapter four, when we apply this methodology to the case study.

3.3.3. The Data Level

In the data level we define and instantiate the classes, properties and individuals that define the organization's structure, services and products and all the elements related to them.

These classes are typically extensions of the classes available in the BPMO vocabularies, like the ones depicted in figure 3.8.

```
<owl:Class rdf:about="&docs;Document"/>
<owl:Class rdf:about="&products:Product"/>
<owl:Class rdf:about="&org:OrganizationalUnit"/>
<owl:Class rdf:about="&org:Employee"/>
```

Figure 3.8 BPMO Classes

The characterization of communications regarding their metadata is also present in the data level; a communication is described using the owl:Class *ecc-data:Communication*, created for this matter, and it can assume any of the properties of the Dublin Core Metadata Element Set.

3.3.4. Mapping between the two levels

Regarding the terms defined in the Meta Level, we can have a property stating that a given term is mapped by some class in the Data level. The property that allows us to do that is the ecc-meta:maps property.

Let us say, for example, that we have a specific type of document very commonly used in the activities of some particular business domain. It is natural that we need to model this type of document, in order to maintain information about the instances of that type of document that occur during the organizational activity. It is also natural that we want to semantically understand what that type of document is about, and eventually what is its relation to other concepts in our business domain. Figure 3.9 exemplifies how this kind of situation looks like.

```
<owl:Class rdf:about="&ecc-data;ParticularDocument">
    <rdfs:subClassOf rdf:resource="&docs;Document"/>
</owl:Class>

<owl:DatatypeProperty rdf:about="&ecc-data;VeryParticularProperty">
    <rdfs:domain rdf:resource="&ecc-data;ParticularDocument"/>
</owl:DatatypeProperty>

<skos:Concept rdf:about="&ecc-meta;VeryParticularDocument">
        <skos:prefLabel rdf:datatype="&xsd;string">
            Very Particular Document
        </skos:prefLabel>
        <skos:altLabel rdf:datatype="&xsd;string">
            Very, Very Particular Document
        </skos:prefLabel>
        <skos:definition rdf:datatype="&xsd;string">
            Very particular document with very peculiar properties which deals with very specific stuff
        </skos:definition>
        <skos:related rdf:about="&ecc-meta;VeryPeculiarStuff"/>
        <ecc-meta:maps rdf:about="&ecc-data;ParticularDocument"/>
</skos:Concept>
```

Figure 3.9 Mapping between levels example

On the first part of the example, we define the type of document in the data level. In that level we can assert instances of that type of document.

On the second part of the example, we define the meaning of that type of document in the meta-level. In that level we define that the class that allows us to have instances of that type is in the data level, through the ecc-meta:maps property.

3.4. Methodology

The methodology established is described through a series of five steps, which are:

- Knowledge Elicitation
- Domain Lingo
- Communications Purposes
- Organization's Structure
- Mapping between the two levels

Each of the following sub-sections represents and describes a step of the methodology, illustrated with concrete examples whenever possible.

3.4.1. Knowledge Elicitation

Depending on the knowledge sources available in each organization, we must define what methodologies we may apply in order to use those sources to extract and model elements of our ontology. This decision is represented in figure 3.10.

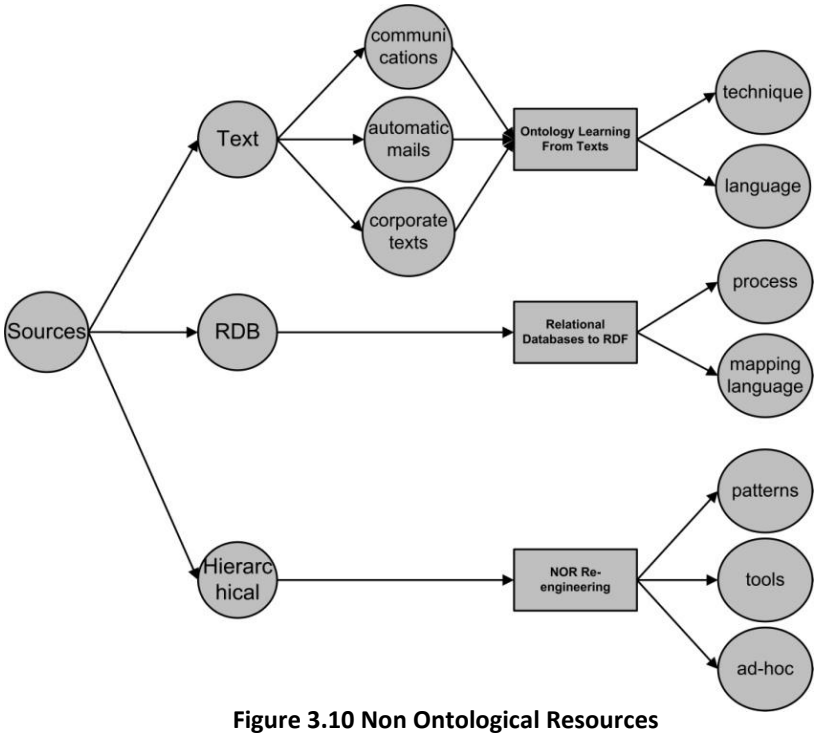

Figure 3.10 Non Ontological Resources

Regarding textual resources, a distinction is made between hand-made communications (from any source), automatically generated e-mails (from templates, forms, etc) and corporate texts. These resources have different kind of structure, and that structure may influence the choice of a specific technique to extract domain terms from corpus of texts. The language of the textual resources is also an important issue, as there are few available methods for the Portuguese language, for example, when compared to the English language.

Regarding the relational models that may exist in the organization, the choice is based upon the kind of process (automatic, semi-automatic) and the mapping language available.

Last, but not least, the organizations may have hierarchical resources of any kind to represent their own internal structure, for example, taxonomies with the hierarchical responsibilities of each department or detailing their offer of products and services. In this case, we may verify if there are existing available patterns or tools to re-engineer the resource, or perform an ad-hoc procedure to model them into our ontology. This ad-hoc procedure may be performed manually, or with some auxiliary technology, such as a XSLT transformation, for example.

3.4.2. Domain Lingo

The role of obtaining and modeling the organization domain lingo plays a crucial part in achieving the desired specificity of the application.

From any kind of source listed in the previous section we may extract valuable pieces of domain lingo; our first goal is to reunite a list of every term extracted from those sources, which results in a first pre-glossary of terms. These terms are then modeled as skos:Concepts on the Meta Level, as explained in the section 3.3.2.

Every skos:Concept must have a skos:prefLabel property specifying its most common label and a skos:definition property specifying its meaning; the value associated to this property can be equal to the value corresponding to the skos:prefLabel property, whenever the name speaks for itself, but the property must have a value associated, for searching and navigation purposes.

It can also have one or more skos:altLabel and skos:hiddenLabel properties, in order to specify alternative names for the concept, in the first case, and deprecated labels, in the second. As with misspelling, it is expected that the classification process includes its own mechanisms to process, detect or avoid them.

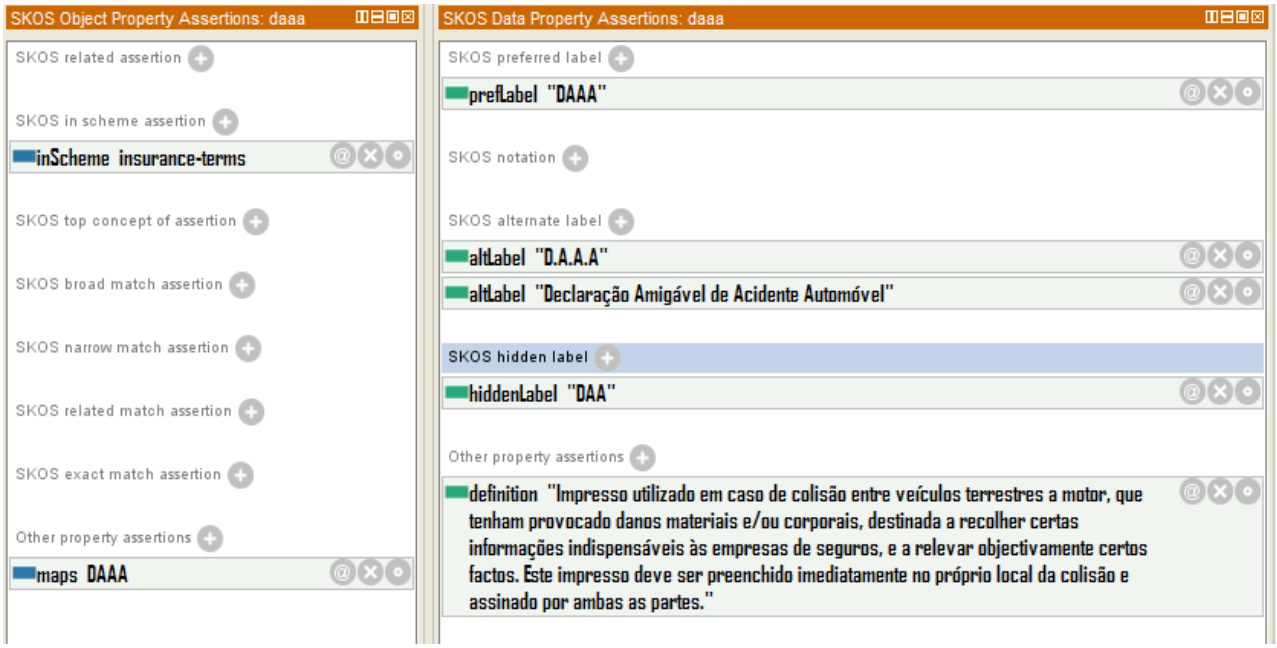

Figure 3.11 Definition of SKOS properties in Protégé SKOSed Plugin

Also, every concept is associated to a conceptScheme specifying its domain of terms. For example, for our case study from the insurance domain, every concept is assigned to a conceptScheme labeled “insurance scheme”. This promotes eventual reutilization of terms between different domains, with the possibility of assigning more than one conceptScheme to a concept.

The definition of these elements can intuitively be made through the Protégé SKOSed plugin interface, as shown in figure 3.11.

3.4.3. Communication Purposes

More than the others, the application of this step will strongly depend on the organization and the domain we are dealing with, and it is strongly encouraged that some domain expert participates actively in its elaboration since the beginning.

We may take a look at the big picture by analyzing the communications, crossing their content with the organization existing departments and activities and trying to model the possible communications purpose by ourselves. But we may not be able to fully correspond to the organization internal processes, and the intents they wish to achieve by assigning purposes to communications.

The first step is to obtain a flat list of possible purposes to assign to communications. Establishing a hierarchy of purposes would eventually be more intuitive to the domain expert, but it may lead to an unsatisfactory result: organization users tend to think only in terms of relay and forwarding between departments; although this may fit the classification process requirements, it does not comply with the search and navigation ones.

This list is then analyzed by the ontologist, who needs to identify and break it down in several dimensions. The number and nature of dimensions may vary with the organization we are dealing with. Consider, for example, that we have the following purposes:

- Request for Destruction of Document
- Request for Sending a Document
- Supply of Requirements

We could define that these purposes are dealing with three dimensions: “nature”, “activity” and “object”. The nature dimension would involve the “Request” and “Supply” topics, the “activity” dimension, the “destruction” and “sending”, and the “object” dimension the “document” and “requirements”. As we can see, a purpose does not necessarily involve topics from every dimension defined.

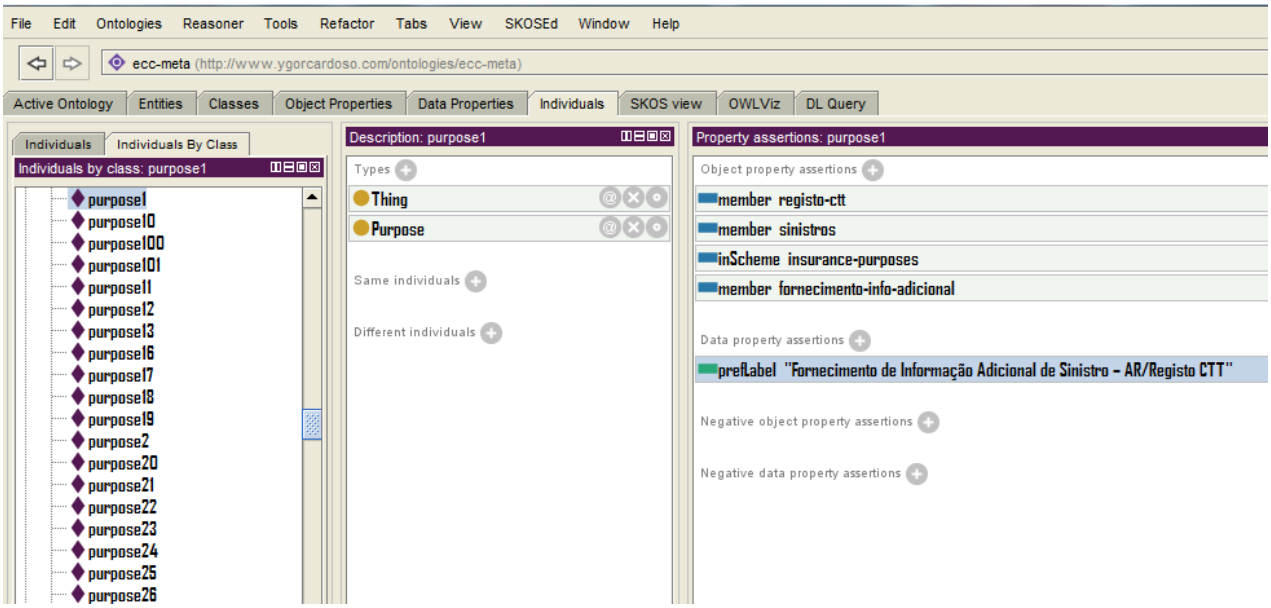

Figure 3.12 Definition of purposes in Protégé SKOSed Plugin

Once again, as these definitions are made through extensions of the SKOS vocabulary, the definition of these elements can intuitively be made through the Protégé SKOSed plugin interface, as illustrated in figure 3.12.

3.4.4. Organization’s Structure / Products and Services

For extending the base ontology to include the organizations structure and the products and services they offer, in most cases, we simply use Protégé forms to fulfill the details about their employees, the departments they have and their products. This is depicted in figure 3.13, where we are asserting an instance of an employee, which manages a certain organizational unit within the organization.

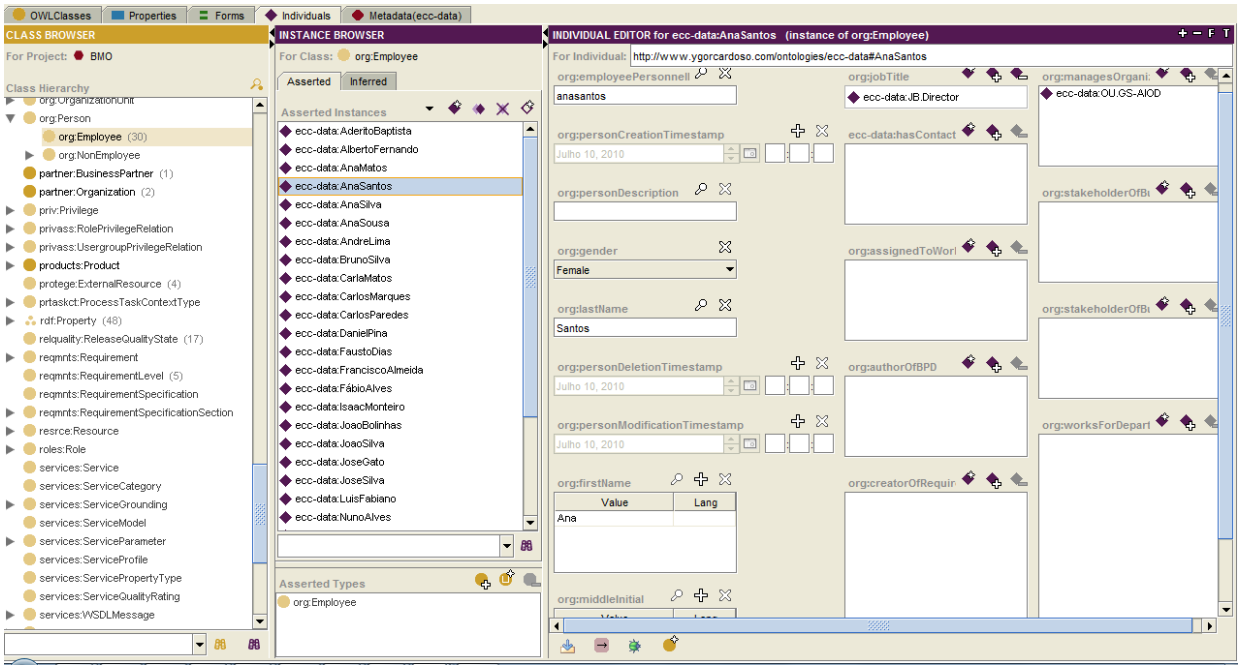

Figure 3.13 Employee Details

This is the step that may require the least intervention of the domain expert, as it is expected to be more linear and to have almost all of its relevant content on the knowledge sources provided. However, as always, it should be concluded with the domain expert validation and approval.

3.4.5. Mapping between the two levels

Whenever deemed necessary, the mapping between the meta level and the data level can be established with the property “ecc-meta:maps”. This necessity may be expressed by the domain expert, or simply by the detection of the existence of a concept in the Meta Level and a class in the Data Level that refer to the same thing.

The property ecc-meta:maps can also be used and established in the protégé interface. The domain of this property is the class skos:Concept; although we cannot state a rule forcing the range of a property to correspond only to classes of a certain vocabulary, we must only establish these property with classes of the Data Level.

Chapter 4
Application

In this chapter the methodology presented in chapter 3 is exemplified through its application on a specific case study.

The real case study supplied by ITDS is about an Insurance Company, which works with insurances from different sectors (labor insurances, health insurances, etc), but concentrates almost all its activity in the car insurance area.

As this is a Portuguese company, every source gathered is in its mother language and the final product is expected to accommodate this language, the examples provided will also remain in Portuguese, with their respective explanation remaining in English. The company name remains anonymous, and the employees presented in the examples are fictional.

4.1. Knowledge Elicitation

ITDS supplied a set of about one thousand communications from this company, classified with different purposes, an organogram with the company complete organization and some document templates used on their business activities.

During the course of this project, the students had several meetings with the company's organizational development director, who served as the domain expert and was able to impose some requirements about the case study, and to verify and validate the final result of this application.

Besides this material, we also gathered some material independently, from the company's website, website of other companies from the same domain, and documents from our own car insurances.

In summary, we had at our disposal:

- Unstructured text, in the form of one thousand e-mail communications from the insurance company, provided by ITDS, the company website, a company glossary of terms
- Structured text, on the form of insurance documents of our own and provided by the company, as well as the company's own corporate website
- A domain expert, the company's organizational development director

The communications used are inbound communications, that is, communications entered in the company, sent by customers. These are the ones that contain the most challenging issues, as their content is expected to be more unstructured and unexpected than the content of outbound communications. Outbound communications may have some kind of format, template or standard associated, which eases the description of the content and classification process.

In the following sections, the methodology is applied to this specific case study, with the distinction of its steps in the Meta Level and the Data Level.

4.2. Meta Level

4.2.1. Modeling the Domain Lingo

Based on an illustrative glossary document available on the company's website and on a list of document templates kindly given by the company, specific domain terms were captured and defined as skos:Concepts. This definition includes their formal definition (included in the documents), preferential labels and some alternative labels (acronyms explanation, synonyms), as shown in the examples depicted by figures 4.1 to 4.3.

```
<skos:Concept rdf:about="&ecc:meta;beneficiario">
        <skos:definition rdf:datatype="&xsd;string">
            A pessoa, singular ou colectiva, a favor de quem reverte o
            capital decorrente do contrato de seguro ou de uma
            operação de capitalização.
        </skos:definition>
        <skos:altLabel rdf:datatype="&xsd;string">Beneficiario</skos:altLabel>
        <skos:prefLabel rdf:datatype="&xsd;string">Beneficiário</skos:prefLabel>
        <skos:inScheme rdf:resource="&ecc:meta;insurance-terms"/>
</skos:Concept>
```

Figure 4.1 - Domain Lingo Example 1

```
<skos:Concept rdf:about="&ecc:meta;fga">
        <skos:prefLabel rdf:datatype="&xsd;string">F.G.A</skos:prefLabel>
        <skos:altLabel rdf:datatype="&xsd;string">FGA</skos:altLabel>
        <skos:altLabel rdf:datatype="&xsd;string">Fundo de Garantia Automóvel</skos:altLabel>
        <skos:definition rdf:datatype="&xsd;string">
            Organismo que funciona sob a tutela do Instituto de Seguros de Portugal,
            ao qual compete a regularização de sinistros automóvel causados por veículos
            não identificados ou por veículos que não possuam seguro de
            Responsabilidade Civil válido e eficaz.
        </skos:definition>
        <skos:inScheme rdf:resource="&ecc:meta;insurance-terms"/>
</skos:Concept>
```

Figure 4.2 Domain Lingo Example 2

```
<skos:Concept rdf:about="&ecc:meta;peritagem">
        <skos:definition rdf:datatype="&xsd;string">
            Documento técnico que traduz detalhadamente o apuramento
            e a quantificação dos danos, resultantes de um sinistro,
            sempre que tal for tecnicamente viável. O Relatório de Peritagem será
            disponibilizado sempre que solicitado. Porém, a Lusitania deixa sempre uma cópia
            do mesmo dentro do veículo sinistrado, imediatamente após a conclusão da peritagem.
            </skos:definition>
        <skos:prefLabel rdf:datatype="&xsd;string">Peritagem</skos:prefLabel>
        <skos:altLabel>Relatório de Peritagem</skos:altLabel>
        <skos:related rdf:resource="&ecc:meta;SinistroAutomovel"/>
        <skos:inScheme rdf:resource="&ecc:meta;insurance-terms"/>
</skos:Concept>
```

Figure 4.3 Domain Lingo Example 3

At the time of the writing of this thesis, the meta level had 315 specific terms of this domain (instances of skos:Concept).

4.2.2. Modeling the Communication's Purposes

In this organization, the domain expert strongly participated in the process of defining the communications purposes, as well as the classification process developer. In fact, the classification process developer even collected and proposed new purposes, based on the tests made on his work. That made a clear alert that we always have to keep in mind: the purposes are very susceptible to changes: if the organization structure changes, the purposes may change; if the company start selling new products or dealing with new business areas, the purposes may change; and even if their customers behavior changes, the purposes may suffer changes.

At first, as expected, the domain expert provided the flat list with all the possible purposes of the communications. Since the beginning, the expert has stated that these purposes were divided in three different areas: production, accidents management and client's management; this distinction made clear that a dimension should emerge to accommodate it, and we called it the "organizacional" (in Portuguese) dimension.

Some examples of purposes from that flat list are:

- Supply of Additional Information about Accident: Vehicle Number
- Supply of Additional Information about Accident: Lawyer
- Request for Car Insurance Quotation – Production
- Request for Contract Termination - Production
- Request for Accident Compensation Payment

Most of the purposes also showed that they were related to some customer intention in respect to the organization activities. In fact, there are only few distinct intention covered by this purposes, which are additional information supply, complementary information supply, payment request, product quote or (accident) process opening and complaint. The difference between complementary information and additional information is that the complementary information is required and crucial to some process, while the additional information is not.

Finally, the intentions are common to a lot of purposes, and the third dimension that makes it possible to distinguish them is the object dimension: an object can be a document, an actor or an action. A complaint about what? Additional information covering what? The answer to that "what" is an object, from the object dimension.

The full definition of the purposes is illustrated in the annex of this thesis; by the time this writing, there were 95 possible purposes identified in this organization.

4.3. Data Level

4.3.1. Modeling the Organization's structure

The organogram with the company's internal structure is modeled as follows:

The BPMO base class org:OrganizationalUnit is extended with the subclass Division.

The BPMO base class org:Department is extended with the classes ecc-data:StaffDepartment (to allow a distinction between staff department and production line department, for example, even if it is not quite relevant for this case). The different company divisions and departments are instantiated, and the departments are linked to their division through the org:ParentOrganizationalUnitProperty.

```
<ecc-data:Division rdf:ID="OU.CGS">
    <org:organizationUnitManager rdf:resource="#anacarvalho">
    <org:organizationUnitName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Centro de Gestão de Sinistros
    </org:organizationUnitName>
</ecc-data:Division>

<ecc-data:StaffDepartment rdf:ID="OU.CGS.RiscosEspeciais">
    <org:parentOrganizationUnit rdf:resource="#OU.CGS"/>
    <org:organizationUnitName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Centro de Gestão de Sinistros - Departamento de Riscos Especiais
    </org:organizationUnitName>
</ecc-data:StaffDepartment>

<org:Employee rdf:ID="AnaCarvalhal">
    <org:firstName xml:lang="pt">Ana</org:firstName>
    <org:managesOrganizationUnit rdf:resource="#OU.CGS"/>
    <org:jobTitle rdf:resource="#JB.Director"/>
    <org:lastName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Carvalhal</org:lastName>
    <org:gender rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Female</org:gender>
</org:Employee>
```

Figure 4.4 Organization Structure Example

Several persons with their respective personal information are instantiated through the org:Employee class. These persons are then arranged regarding their roles and titles, through the BPMO properties org:OrganizationUnitManager, org:JobTitle, etc. In figure 4.4, an example of a person and her role within the organization is exemplified.

4.3.2. Modeling the Organization's products and services

The main source for modeling the organization products and services was a table of product types provided by the domain expert, with a subset of it being represented in table 4.1. The products are organized in areas, and each product may have specific sub-products deriving from it, as represented in the second column.

Ramo/Área	Sub-Ramos
ACIDENTES E DOENÇA	
Acidentes Pessoais	Acidentes Pessoais – Escolar Acidentes Pessoais – Criança Acidentes Pessoais - Congressista
Acidentes em Trabalho	Acidentes em trabalho – Conta Outrém Acidentes em trabalho – Trabalhadores Independentes
Saúde	Saúde – Salutare Saúde - Individual
AUTOMÓVEL	
Assistência em viagem	
Automóvel	
Responsabilidade Civil	Responsabilidade Civil –Portadores de Armas Responsabilidade Civil – Pescadores Responsabilidade Civil - Profissional

Table 4.1 Insurance Company Products

So, a subclass of the BPMO class products:Product is created in order to specify the specific class of products of this company's domain, which is the ecc-data:InsuranceProduct class. This class contains other subclasses, like ecc-data:CarInsuranceProduct (Automóvel in the table) or ecc-data:HealthInsuranceProduct, (Saúde) and those classes are instantiated by some individuals representing specific commercial products, as depicted in figure 4.5.

```
<rdfs:Class rdf:about="#InsuranceProduct">
    <rdfs:subClassOf>
      <rdfs:Class rdf:about="&products;Product"/>
    </rdfs:subClassOf>
</rdfs:Class>

<rdfs:Class rdf:ID="CarInsuranceProduct">
    <rdfs:subClassOf rdf:resource="#InsuranceProduct"/>
</rdfs:Class>

<rdfs:Class rdf:ID="HealthInsuranceProduct">
    <rdfs:subClassOf rdf:resource="#InsuranceProduct"/>
</rdfs:Class>

<ecc-data:CarInsuranceProduct rdf:ID="SeguroAutoXS"/>
<ecc-data:HealthInsuranceProduct rdf:ID="SaudeSenior"/>
```

Figure 4.5 Insurance Products

Although presented here as an ad-hoc transformation of the table of product types, the table could easily be coded in a XML format and transformed to the ontology through an XSLT transformation or an NeOn plugin. But, for the sake of the simplicity of the example, it remains as an ad-hoc transformation.

```
<owl:Class rdf:ID="Apolice">
    <rdfs:subClassOf rdf:resource="&docs:Document;Document"/>
</owl:Class>
<owl:Class rdf:ID="ProcessoSinistro">
    <rdfs:subClassOf rdf:resource="&docs:Document;Document"/>
</owl:Class>
<owl:Class rdf:ID="DAAA">
    <rdfs:subClassOf rdf:resource="&docs:Document;Document"/>
</owl:Class>
```

Figure 4.6 Insurance Documents

After identifying all the products covered by the company, specific documents concerning these products were also identified, and they were modeled as subclasses of the BPMO class docs:Document, as shown in figure 4.6, and some properties for the “Apolice” document subclass are defined as shown in figure 4.7.

```
<owl:DatatypeProperty rdf:ID="insurance-number">
    <rdfs:domain rdf:resource="#Apolice"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="insurance-owner">
    <rdfs:domain rdf:resource="#Apolice"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="insurance-prize">
    <rdfs:domain rdf:resource="#Apolice"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="insurance-coverage">
    <rdfs:domain rdf:resource="#Apolice"/>
</owl:DatatypeProperty>
```

Figure 4.7 "Apolice" Document Properties

An instantiation of the "Apolice" class is exemplified and shown in figure 4.8, including the establishment of a relation between it and some "Processo de Sinistro" document, as an example of use of the BPMO properties, and also of mappings between the Meta Level and the Data Level (for the “Apolice” and “DAAA” concepts.

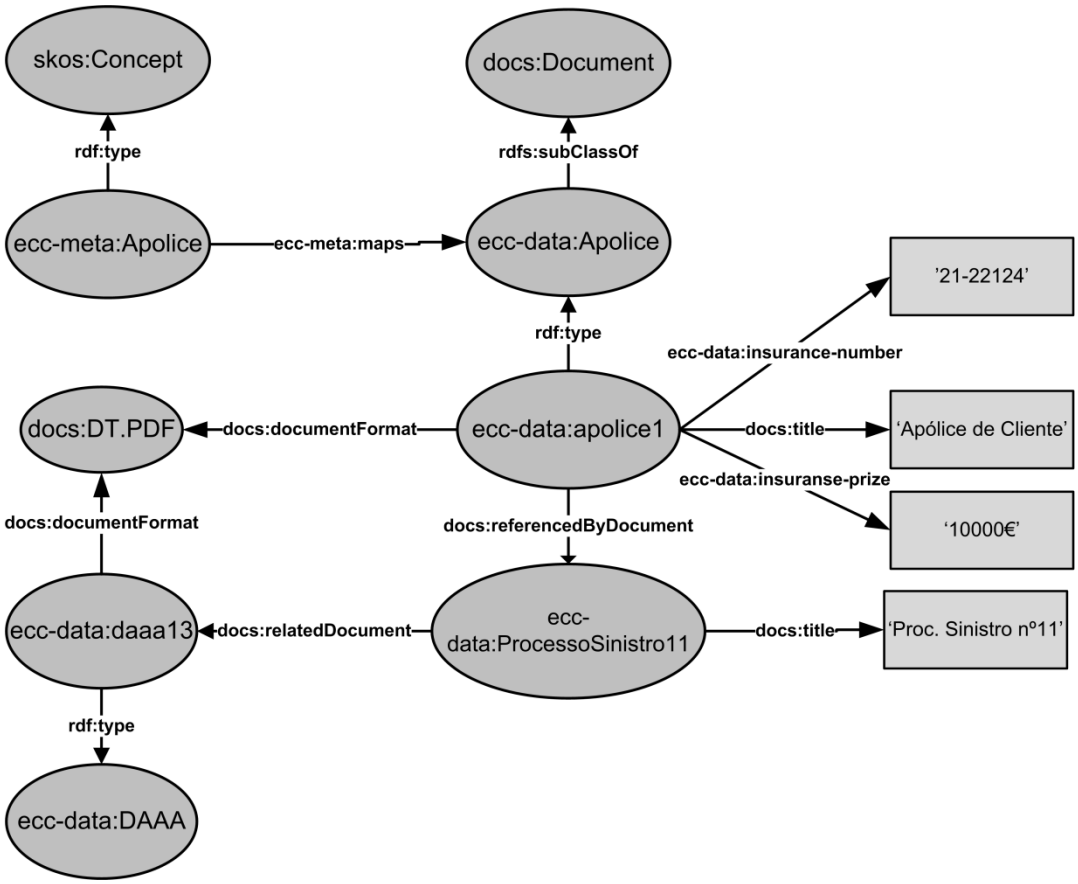

Figure 4.8 "Apolice" and Other Instances of BMO Classes

4.4. Mapping between the two levels

In this case study, there were some terms identified in the Meta Level that corresponded to specific classes on the Data Level, like the “Apolice” and “Processo de Sinistro” documents, which are terms refered in more than 90% of the communications, or the roles of the organization staff, as the role “Mediador”, which is also a very common term.

The mapping between these terms and the classes they corresponded to was established by the ecc-meta:maps property, as specified before. But an interesting situation also emerged: some communications had references to terms that were not expressed in the Meta Level, but were present in the Data Level.

These terms corresponded to the name and type of the company products: in every communication involving a request for a product quote, the name of the product or its type is cited the communication. This clearly showed the need to include these products in the Meta Level, following the same hierarchical structure (in this case, represented by the SKOS properties) of their equivalent Data Level classes.

This made clear that it does makes sense, anyway, that we represent the products in the Meta Level, though at first we didn’t figure out the importance of doing it, because most of the communications of this case study were not about product quotations, but about accidents.

4.5. Implementation Details

All the examples presented in this section were extracted from the data that is integrated in a Sesame repository, on a server provided by ITDS for that mean. The repository, as mentioned in section 2.2.2, is a native-rdfs Java Store, which allows RDF Schema inferencing (an implicit requirement from the company.

The data was created and edited with Protegé, with the help of the SKOSed plugin for managing the meta-level concepts.

This work does not have the outcome of providing a specific prototype or software, as it is beyond its scope, since the methodologies presented will be integrated on the XEO platform during the integration phase, which is due to start after the conclusion of this document.

Chapter 5
Evaluation

This chapter presents some considerations about the compliance with requirements defined as objectives, the consistency of the ontologies and their behavior when confronted with changes.

Probably the most obvious approach for evaluating the quality of the ontologies designed in this project is to measure the satisfaction of the developers of the classification and search processes, and also the quality of the results they obtain. However, their results are being obtained and written concurrently with this thesis, and it is not feasible to include them already in this report.

It is also not feasible to achieve a "human-made" evaluation from the domain experts (from the case study), as their collaboration in this phase was restricted to provide the material and to validate our immediate achievements concerning its application, namely the list of purposes and our doubts about the domain lingo; their real usage of the ECC platform is not yet available.

The ontologies could be evaluated from the perspective of the "golden standard" technique [97], but as this is a very specific application-dependant ontology, there is no "golden standard" covering its requisites and usage.

As such, this work is evaluated in this chapter by three criterions: the satisfaction of the requirements expressed as objectives, the way the ontology is expected to deal with changes and the ontology consistency. Each one of these criterions is detailed in its respective sub-section as follows.

5.1. Satisfaction of Requirements

Regarding the requirements expressed and detailed in section 3.1:

The ontology must have the capacity of, if demanded, being expressed in more than one language.

This is a requirement directly and trivially fulfilled by the intrinsic capacities of the semantic web standard ontology languages of expressing the data they represent in a Multilanguage form.

The case study presented in chapter 4 is based on a Portuguese company, and the language defined for its application on this ontology is also this one. Every time some property is associated with some string literal value, that property is associated with the "xml:lang" property specifying the "pt-PT" value; in the future, if some additional language is required, it is gracefully applied with the addition of a new property with another "xml:lang" value.

The ontology must be maintained in a repository which allows it to be easily queried from an interface.

The ontologies are present in a Sesame repository; this repository includes two contexts, one with the data from the Meta Level, and the other with the data from the Data level. At present, the classification and search and navigation developers are only querying it through the openRDF web

interface, but a specific gateway to this project will be developed after the conclusion of our thesis with the collaboration of ITDS.

The ontology must be easily extended to any domain.

This is the main reason why the methodology described in section 3.4 was developed, in the first place.

The application to a case study illustrated in chapter 4 showed how it was well suited for the insurance domain; to effectively demonstrate its usage to different domains, we would have to contemplate different case studies, which are not available, at the time being, and would require finding an expert with available time (and motivation) for discussing the ontology; this is something difficult to achieve given the time constraints of a MSc thesis. However, as the insurance domain was a completely random domain and was applied after the main principles of the core set of ontologies were already established, we are lead to believe that it will behave well on the presence of other domains; besides that, no specific parts of the core set of ontologies were created or tuned specifically for this case study: every part of this application consisted on extensions and instantiations of the existing ontologies.

5.2. Response to Changes

Changes are distinguished in two categories: the predictable changes that can occur frequently within an organization and the requirements changes that can occur anywhere on the own project and its components. On the following sub-sections these two categories are analyzed.

5.2.1. Predictable Changes

On the predictable changes field, there are a few ones that are supported directly and gracefully by the model. It is expected that the domain lingo is continuously being improved, and that new terms are frequently being added to domain; terms already existing in the domain can also be changed, or updated with new synonyms. For example, in the case study presented, there is a Portuguese government entity called “CIMPAS”, that was formerly called “CIMASA”; the label “CIMASA” remains as a hidden label to the skos:Concept that represents CIMPAS. We could create a new concept representing the entities separately, but for classification purposes it is expected that they refer to the same thing.

Other predictable changes may require further adjustments; for example, changes in the organization structure: in this case study, a straight department’s hierarchical structure is presented, and if these departments change their names, their roles or their structure, the update of the ontology is also straightforward; however, an organization may assume, for example, a matrix

structure with an arrangement between its departments and services/products. The BPMO vocabularies do not provide guidelines for native support to matrix management organizations, but the properties it contains can be adapted to represent that kind of structure. For example, the "org:worksForDepartment" and the "org:managesOrganizationalUnit" properties used to describe person roles do not have cardinality constraints imposed to them; this means that a single employee can work or manage departments of different areas, which covers the most basic aspect of matrix organizations: multidisciplinarity. This multidisciplinarity can be further refined to explicit represent the disciplines that are being crossed; for example, with the "org:Workgroup" class (which is also a subclass of "org:OrganizationalUnit", as are "org:Department" and "org:Division"), we can define that one or more employees belong to a workgroup, and that employees can belong to distinct departments and divisions between them. That workgroups can be assigned to products, services or whatever element the matrix organization crosses their departments with.

As with purposes, besides the discovery or need for new purposes, there are two variants for their evolution: the need of simply including new purposes on the same dimensions, or the need of adding new dimensions on which new purposes and old purposes are evaluated. Once again, adding new purposes based on the same dimensions is done gracefully. As with new dimensions, besides adding those dimensions, we got to define how to deal with the old ones: they are simple discarded and substituted by the new ones, or do we have a need to maintain historical information about them? This will depend on the way the classification process annotates the classified communications with a purpose: will all the purpose information the search and navigation process requires, be embedded on the annotation, or will the ontology have to provide support for obtaining that information? In either case, the ontology is prepared to eventually accommodate those changes: we can always add new properties to indicate whether a purpose or a dimension is active or not, or to provide historical information about it.

5.2.2. Changes in the Requirements

Regarding the requirements expressed in section 3.1 and re-enumerated in section 5.1, changes that may occur among them are trivial; the ontology having or not the capacity of being expressed in more than one language is intrinsically related to the ontology languages capacities, and if this requirement is discarded, these properties are simply ignored or not used at all. As with the other requirements enumerated, the ontology being or not present in a repository is not a threatening change, as it is easily physically exported to any other means, and the extensibility requirement is certainly not being changed, as it is one of the main aspects that characterize the platform.

About new requirements that may arrive, it depends on the nature of those requirements: they may come in the form of requirements about the data needed or the interaction requests by

classification or search and navigation processes, on the way the data is represented or on the methodology steps.

The interaction requests will not require further readjustments in the model itself, but in the queries executed to provide the users data; requests based on the way data is represented and on the methodology steps cannot be easily predicted until the platform is in full utilization.

5.3. Consistency

As we are dealing with two different levels (the Meta and the Data level) that may refer to the same concepts, the need of having consistency between the two levels emerges, in order to guarantee that not only we do not get into situations of undesirable logical inconsistency in our ontologies, but also to ensure that the transition from the knowledge sources to our desired model is always made in the same way.

For this matter, in this section some rules are defined to ensure that consistency and to serve as a guide of good design and modeling practice. Some queries regarding these rules are also suggested, in order to further test that rules are being correctly asserted, and even to eventually include them in the application (as user warnings, for example). We also showed some examples on how the application of these rules can be tested.

5.3.1. Rule 1: Subclass inheritance

Every time there exists a skos:narrower property asserted between two concepts in the meta level, and they are mapped to classes in the data level, there must exist a rdfs:subClassOf property between the corresponding classes.

We can verify this rule with the following SPARQL query, for example:

```
SELECT ?concept WHERE {
    ?concept rdf:type skos:Concept.
    ?concept skos:narrower ?otherconcept.
    OPTIONAL {
        ?concept ecc-meta:maps ?class.
        ?otherconcept ecc-meta:maps ?otherclass.
        ?otherclass ?property ?class.
        ?property rdfs:subClassOf rdfs:subClassOf.
    }
    FILTER (!bound(?property))
}
```

This query obtains the concepts that have a skos:narrower property asserted between it and other concepts, and verify if they are mapped to classes in the data level, and whether do a rdfs:subClassOf property do not exist between them; this give us concepts that violate the rule.

5.3.2. Rule 2: Mapping down to Data Level

The property ecc-meta:maps shall only be asserted to a class defined in the data-level. This rule exists because we can use the property rdfs:range to state that the values of a property are instances of one or more classes, but we cannot use it to define the values a property cannot assume; of course the values it cannot assume are the ones that are not defined in its range, but to define all the possible ranges of the property would be quite limitative of its potential. We also cannot define a rule that states that the range is made by the classes that belong to a specific vocabulary.

This rule can be tested by the following query:

```
//Query for Rule 2
SELECT ?concept ?class WHERE {
    ?concept ecc:maps ?class
    ?concept rdf:type skos:Concept.
}
```

This query allows us to verify if there is any skos:Concept being asserted as a mapping of another skos:Concept (the verification of the ?class variable being of type skos:Concept is implied, as the range of the property is the skos:Concept class). We cannot directly filter a query to state that we only want classes defined in a specific namespace (it would be desirable to check if every maps property only has a data-level class as value).

In order to keep this test effective, we need to follow the rule 3, as this would be useless if we had skos:Concepts in the data level.

5.3.3. Rule 3: Concepts Only in Meta Level

There can be no skos:Concepts in the data-level. This rule is just needed in order to confirm the second rule.

Chapter 6

Conclusions and Future Work

This chapter draws final conclusions on the design and implementation of this thesis and presents future work activities.

This chapter draws final conclusions on the design and implementation of this thesis and presents some future work expected to be developed.

6.1. Conclusions

The core set of ontologies and vocabularies, whose conceptual model and implementation is presented in this thesis, was designed to comply with several requirements for effectively describing communications (as listed in chapter 3.1), with a strong emphasis on extensibility. By complying with those requirements, the core set of ontologies becomes an effective solution for knowledge representation in any specific business domain. This capacity of properly describing the specificity of the communications is a key factor for achieving good results in the classification and search and navigation processes, and thus allowing for the ECC platform to really make a difference.

At the first phase of this thesis work, a strong effort was put on the conceptualization of the problem and of the solutions it required, in order to try to cover all the aspects regarding the requirements the model needed to comply with, to gather all the alternative choices available to comply with those requirements, and also to maintain some independence from technology, platform and domain issues before effectively implementing it.

The use of ontologies was not a choice decided and defended in this work, but taken for granted by ITDS since the beginning of this project; as expected, it proved to be a well suited suggestion, fulfilling all the platform needs for semantic expressivity, flexibility and characterization of linguistic concerns.

The technologies used comply with most of the current World Wide Web's standards for the Semantic Web, and, thus inheriting all the benefits from belonging and evolving along with the web of linked data, which includes interoperability, stability, consistency, support and even visibility of the product.

The methodology presented describes a step-by-step guideline of an iterative process for identifying the knowledge sources that may be available in an organization, for applying effectively the extensibility potential of the model, and to minimize the effort required on the setup process of the ECC platform on future customers. In fact, this setup process will be very similar to the one described on the application of the methodology to the specific case study, and will serve as a reference for future consultants trained by ITDS for this matter.

The case study that was developed allowed us to deal hands-on with real and specific knowledge sources of different kinds, and to interact and discuss with qualified domain experts. This gave us a

good perspective of the expected behavior of the model and the methodology, and of the problems that we may expect and deal in the future. It also gave us the notion that in order to gain the desired specificity, we must involve the domain experts actively in that process, as they are not only the ones who know where specificity is embedded in the knowledge sources, but also the ones that will effectively benefit from capturing it; only their validation can guarantee that the knowledge gathered in our ontologies will be useful, and that it covers the domain at a correct extent.

The evaluation made on the satisfaction of requirements and response to changes allowed us to verify if the objectives were being achieved, to draw some scenarios about future problems, and to specify some possible solutions to respond to them. It also showed that the consistency of the model is not only achievable by following the steps of the methodology, but also easily verifiable with queryable means.

Although the results achieved by the classification and search and navigation processes are not completely available by the time of the conclusion of this document, the preliminary tests made on the case study showed some good indicators.

6.2. Future Work

This section presents future activities for the work described in this thesis, for the design and implementation of the metadata repository. Regarding these activities the following points are suggested.

Integration in XEO-ECC platform

The full integration of the ontologies and the methodology developed on this thesis with the XEO Studio ECC Edition is expected to be started immediately after the delivery of this document and occur at least until December 2010; this may include establishing the interaction between the ontologies designed on this work and the XEO native business objects, and eventually developing pluggable elements to the ECC platform in order to assist the application of the methods described in this document.

As such, we can divide this integration into two different goals: to provide interoperability between the ITDS framework and the methods processes developed in the FCT side, and to provide mechanisms and tools to integrate editing and maintenance capabilities to the ITDS framework to the final product.

During the integration phase, the FCT-UNL students will provide specific formation to XEO developers and users about RDF, OWL and SPARQL technologies and tools.

Application of the methodology to other domains

This activity will emerge naturally during the life cycle of the platform, and will be an important test of the methodology correctness (and also a strong contributor to its evolution). As already mentioned, the methodology will also serve as a guide to future consultants assigned by ITDS for this matter.

Evaluation of the ontologies

After the integration of the ontologies on the ECC platform and its complete development, a full, intensive and thorough formal evaluation of the ontology performance and structure during the utilization by its end-users can be made, and eventually serve as an own MSc thesis, in collaboration with FCT-UNL.

Specific Knowledge Acquisition Methods Development

Once again, the development of specific knowledge acquisition methods mentioned on this work can result in specific MSc theses; this may include specific works about ontology extraction from text corpus, relational databases or any other knowledge source identified; those works can also include the development of plugable methods for benefiting of those sources.

If I had to choose, this would the area that I would be more interested in continuing working on, as I had the opportunity to do some superficial research around these methods in the state of the art, but lacked the time to effectively delve into them.

Consistency checking plug-ins

The queries exemplified in the consistency section of the evaluation chapter can be included in the ECC platform in order to alert the user about the possible violation of consistency constraints.

References

[1] Abbasi, R., Staab, S. & Cimiano, P. (2007), "*Organizing Resources on Tagging Systems using T-ORG*", In proceedings of Workshop on Bridging the Gap between Semantic Web and Web 2.0 at ESWC 2007., pp. 97-110.

[2] Aguado De Cea, G., Gómez-Pérez, A., Montiel-Ponsoda, E. & Suárez-Figueroa, M.C. (2008), "*Natural Language-Based Approach for Helping in the Reuse of Ontology Design Patterns*", In EKAW '08: Proceedings of the 16th international conference on Knowledge Engineering. Berlin, Heidelberg., pp. 32-47. Springer-Verlag.

[3] An, Y. & Mylopoulos, J. (2005), "*Translating XML Web Data into Ontologies*", In OTM Workshops., pp. 967-976.

[4] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S. & Aumueller, D. (2009), "*Triplify: light-weight linked data publication from relational databases*", In WWW '09: Proceedings of the 18th international conference on World Wide Web. New York, NY, USA., pp. 621-630. ACM.

[5] Barrasa, J., Corcho, Ó. & pérez, A.G. (2004), "*R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language*", In Proceedings of the 2nd Workshop on Semantic Web and Databases (SWDB2004., pp. 1069-1070. Springer.

[6] Bender, O., Och, F.J. & Ney, H. (2003), "*Maximum entropy models for named entity recognition*", In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003. Morristown, NJ, USA., pp. 148-151. Association for Computational Linguistics.

[7] Bergman, M.K. (2001), "*The Deep Web: Surfacing Hidden Value*", Journal of Electronic Publishing., In TAKING LICENSE: Recognizing a Need to Change., WHITE PAPER. Ann Arbor, Michigan, August, 2001. Vol. 7(1) Scholarly Publishing Office, University of Michigan University Library.

[8] Bernaras, A., Laresgoiti, Iñ. & Corera, J.M. (1996), "*Building and Reusing Ontologies for Electrical Network Applications*", In ECAI., pp. 298-302.

[9] Berners-Lee, T. (1998), "*What the Semantic Web Can Represent*", Online.

[10] Bizer, C. & Cyganiak, R. (2007), "*D2RQ - Lessons Learned*", W3C Workshop on RDF Access to Relational Databases., October, 2007.

[11] Blakeley, C. (2007), "*RDF views of SQL data (declarative SQL schema to RDF mapping)*".

[12] Bob, G.S., Wielinga, B. & Jansweijer, W. (1995), "*The KACTUS View on the 'O' Word*", In IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing., pp. 159-168.

[13] Brandon, D. (2005), "*Recursive database structures*", J. Comput. Small Coll.. , USA Vol. 21(2), pp. 295-304. Consortium for Computing Sciences in Colleges.

[14] Brank, J., Grobelnik, M. & Mladenic, D. (2005), "*A survey of ontology evaluation techniques*", In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005.

[15] Buitelaar, P. & Cimiano, P. (2008), "*Ontology Learning and Population: Bridging the Gap between Text and Knowledge*" Amsterdam Vol. 167 IOS Press.

[16] Byrne, K. (2008), "*Having Triplets – Holding Cultural Data as RDF*", In Proceedings of the ECDL 2008 Workshop on Information Access to Cultural Heritage, Aarhus, Denmark, September 18, 2008, M. Larson, K. Fernie, J. Oomen, and J. Cigarran, Eds., September 2008..

[17] Chang, K., He, B., Li, C. & Zhang, Z. (2003), "*Structured databases on the web: Observations and implications*".

[18] chuan Chang, K.C., He, B., Li, C., Patel, M. & Zhang, Z. (2004), "*Structured Databases on the Web: Observations and Implications*".

[19] Chikofsky, E.J. & Cross II, J.H. (1990), "*Reverse Engineering and Design Recovery: A Taxonomy*", IEEE Softw.. Los Alamitos, CA, USA Vol. 7(1), pp. 13-17. IEEE Computer Society Press.

[20] Cimiano, P. & Völker, J. (2005), "*Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery*".

[21] Cruz, I.F., Xiao, H. & Hsu, F. (2004), "*An Ontology-Based Framework for XML Semantic Integration*", In IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium. Washington, DC, USA., pp. 217-226. IEEE Computer Society.

[22] Cunningham, H., Maynard, D., Bontcheva, K. & Tablan, V. (2002), "*GATE: A framework and graphical development environment for robust NLP tools and applications*", In Proceedings of the 40th Annual Meeting of the ACL.

[23] Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D.S. & Yates, A. (2004), "*Web-scale information extraction in knowitall: (preliminary results)*", In WWW '04: Proceedings of the 13th international conference on World Wide Web. New York, NY, USA., pp. 100-110. ACM.

[24] Farquhar, A., Fikes, R. & Rice, J. (1996), "*The Ontolingua Server: a Tool for Collaborative Ontology Construction*", In International Journal of Human-Computer Studies.

[25] Fellbaum (1998), "*WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*", Hardcover., May, 1998. The MIT Press.

[26] Fleischman, M. & Hovy, E. (2002), "*Fine grained classification of named entities*", In Proceedings of the 19th international conference on Computational linguistics. Morristown, NJ, USA., pp. 1-7. Association for Computational Linguistics.

[27] Foxvog, D. & Bussler, C. (2006), "*Ontologizing EDI Semantics*", In ER (Workshops)., pp. 301-311.

[28] Gangemi, A., Pisanelli, D.M. & Steve, G. (1999), "*An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies*", Data Knowl. Eng.. Vol. 31(2), pp. 183-220.

[29] Garca-Silva, A., Gómez-Pérez, A., Suárez-Figueroa, M.C. & Villazón-Terrazas, B. (2008), "*A Pattern Based Approach for Re-engineering Non-Ontological Resources into Ontologies*", In ASWC '08: Proceedings of the 3rd Asian Semantic Web Conference on The Semantic Web. Berlin, Heidelberg., pp. 167-181. Springer-Verlag.

[30] García, R. & Celma, O. (2005), "*Semantic Integration and Retrieval of Multimedia Metadata*", In Proceedings of the ISWC 2005 Workshop on Knowledge Markup and Semantic Annotation (Semannot'2005). Volume 185, pp. 69-80. CEUR Workshop Proceedings.

[31] Gomez-Perez, A. (1994), "*Some Ideas and Examples to Evaluate Ontologies*".

[32] Gomez-Perez, A., Corcho-Garcia, O. & Fernandez-Lopez, M. (2003), "*Ontological Engineering*" Secaucus, NJ, USA Springer-Verlag New York, Inc..

[33] Green, J., Hart, G., Dolbear, C., Engelbrecht, P. & Goodwin, J. (2008), "*Creating a Semantic Integration System using Spatial Data*", In 7th International Semantic Web Conference (ISWC2008). October 2008.

[34] Gruber, T.R. (1993), "*A translation approach to portable ontology specifications*", Knowledge Acquisition. Vol. 5, pp. 199-220.
[35] Grüninger, M. & Fox, M.S. (1995), "*Methodology for the Design and Evaluation of Ontologies*".
[36] Guarino, N. (1994), "*The Ontological Level*", In Philosophy and the Cognitive Sciences. Vienna Hölder-Pichler-Tempsky.
[37] Guarino, N., Welty, C. & Common, E. (2002), "*Evaluating Ontological Decisions With Ontoclean*".
[38] Hakkarainen, S., Hella, L., Strasunskas, D. & Tuxen, S. (2006), "*A Semantic Transformation Approach for ISO 15926*", In ER (Workshops)., pp. 281-290.
[39] Han, L., Parr, C., Sachs, J. & Joshi, A. (2007), "*RDF123: a mechanism to transform spreadsheets to RDF*". University of Maryland, Baltimore County, August, 2007.
[40] Hearst, M.A. (1992), "*Automatic acquisition of hyponyms from large text corpora*", In Proceedings of the 14th conference on Computational linguistics. Morristown, NJ, USA., pp. 539-545. Association for Computational Linguistics.
[41] Hepp, M. & de Bruijn, J. (2007), "*GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies*", In ESWC., pp. 129-144.
[42] Hodge, G. (2000), "*Systems of Knowledge Organization for Digital Libraries:Beyond Traditional Authority*".
[43] IEEE (1997), "*IEEE Std 1074-1997 IEEE Standard for Developing Software Life Cycle Processes*".
[44] Isozaki, H. & Kazawa, H. (2002), "*Efficient support vector classifiers for named entity recognition*", In Proceedings of the 19th international conference on Computational linguistics. Morristown, NJ, USA., pp. 1-7. Association for Computational Linguistics.
[45] Kietz, J.-U., Mädche, A., Maedche, E. & Volz, R. (2000), "*A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet*", In EKAW-2000 Workshop "Ontologies and Text", Juan-Les-Pins., pp. 2-6.
[46] López, M.F., Gómez-Pérez, A., Sierra, J.P. & Sierra, A.P. (1999), "*Building a Chemical Ontology Using Methontology and the Ontology Design Environment*", IEEE Intelligent Systems. Piscataway, NJ, USA Vol. 14(1), pp. 37-46. IEEE Educational Activities Department.
[47] Laboratorio, F.L. (1999), "*Overview Of Methodologies For Building Ontologies*".
[48] Lopez, M.F., Perez, A.G. & Juristo, N. (1997), "*METHONTOLOGY: from Ontological Art towards Ontological Engineering*", In Proceedings of the AAAI97 Spring Symposium. Stanford, USA, March, 1997. , pp. 33-40.
[49] Maala, M.Z., Delteil, A. & Azough, A. (2007), "*A Conversion Process From Flickr Tags to RDF Descriptions*", In SAW.
[50] Maedche, A., Maedche, E. & Staab, S. (2004), "*Ontology Learning*", In Handbook on Ontologies., pp. 173-189. Springer.
[51] Maedche, A. & Staab, S. (2000), "*Discovering Conceptual Relations from Text*", In ECAI., pp. 321-325.
[52] Maedche, A. & Staab, S. (2000), "*Ontology Learning from Text*", In NLDB., pp. 364.
[53] Malinowski, E. & Zimányi, E. (2006), "*Hierarchies in a multidimensional model: from conceptual modeling to logical representation*", Data Knowl. Eng.. Amsterdam, The Netherlands, The Netherlands Vol. 59(2), pp. 348-377. Elsevier Science Publishers B. V..

[54] Maynard, D., Bontcheva, K. & Cunningham, H. (2003), "*Towards a semantic extraction of Named Entities*", In Recent Advances in Natural Language Processing.

[55] Maynard, D., Li, Y. & Peters, W. (2008), "*NLP Techniques for Term Extraction and Ontology Population*", In Proceeding of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge. Amsterdam, The Netherlands, The Netherlands., pp. 107-127. IOS Press.

[56] Maynard, D., Saggion, H., Yankova, M., Bontcheva, K. & Peters, W. (2007), "*Natural Language Technology for Information Integration in Business Intelligence*", In BIS., pp. 366-380.

[57] Maynard, D., Tablan, V., Cunningham, H., Ursu, C., Saggion, H., Bontcheva, K. & Wilks, Y. (2002), "*Architectural elements of language engineering robustness*", Nat. Lang. Eng.. New York, NY, USA Vol. 8(3), pp. 257-274. Cambridge University Press.

[58] Mcenery, T. & Wilson, A. (2001), "*Corpus Linguistics*", Paperback., February, 2001. Edinburgh University Press.

[59] Mikheev, A., Moens, M. & Grover, C. (1999), "*Named Entity recognition without gazetteers*", In Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics. Morristown, NJ, USA., pp. 1-8. Association for Computational Linguistics.

[60] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. & Swartout, W. (1991), "*Enabling Technology for Knowledge Sharing*", AI Magazine., August, 1991. Vol. 12(3), pp. 36-56. AAAI Press.

[61] Newell, A. (1982), "*The Knowledge Level*", Artif. Intell.. Vol. 18(1), pp. 87-127.

[62] Noy, N.F. & Musen, M.A. (2003), "*The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping*", International Journal of Human-Computer Studies. Vol. 59, pp. 2003.

[63] Noy, N.F. & Musen, M.A. (1999), "*SMART: Automated Support for Ontology Merging and Alignment*".

[64] Pastra, K., Maynard, D., Hamza, O., Cunningham, H. & Wilks, Y. (2002), "*How feasible is the reuse of grammars for Named Entity Recognition?*", In Proceedings of the 3rd Conference on Language Resources and Evaluation (LREC), Canary Islands.

[65] Pinto, H.S., Staab, S. & Tempich, C. (2004), "*DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies*", In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI., pp. 393-397. IOS Press.

[66] Pooley, R. & Stevens, P. (1998), "*Software Reengineering Patterns*". , 1998.

[67] Popescu, A.-M., Etzioni, O. & Kautz, H. (2003), "*Towards a theory of natural language interfaces to databases*", In IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces. New York, NY, USA., pp. 149-157. ACM.

[68] Presutti, V., Gangemi, A., David, S., de Cea, G.A., Suárez-Figueroa, M.C., Montiel-Ponsoda, E. & Poveda, M. (2008), "*D2.5.1: A Library of Ontology Design Patterns*", In NeOn Deliverable., NeOn Deliverable.

[69] Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Jr, T.T., Auer, S., Sequeda, J. & Ezzat, A. (2009), "*A Survey of Current Approaches for Mapping of Relational Databases to RDF*". 01, 2009.

[70] Schreiber, G. & de Hoog, R. (1999), "*Knowledge Engineering and Management: The CommonKADS Methodology*" MIT Press.

[71] Skuce, D. (1995), "*Conventions for reaching agreement on shared ontologies*", In Proc. 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff Conference Centre, Banff, Alberta, Canada,.

[72] Studer, R., Benjamins, V.R. & Fensel, D. (1998), "*Knowledge Engineering: Principles and Methods*".

[73] Stumme, G. & Maedche, A. (2001), "*FCA-MERGE: Bottom-Up Merging of Ontologies*", In IJCAI., pp. 225-234.

[74] Suarez-Figueroa, M.C. & Gomez-Perez, A. (2008-05), "*Towards a Glossary of Activities in the Ontology Engineering Field*", In Proceedings of 6th International Conference on Language Resources and Evaluation (LREC'08), Marrakech. 2008.

[75] Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R. & Wenke, D. (2002), "*Ontoedit: Collaborative ontology development for the semantic web*"., pp. 221-235. Springer.

[76] Sure, Y. & Studer, R. (2002), "*On-To-Knowledge Methodology*", In On-To-Knowledge: Semantic Web enabled Knowledge Management. , pp. 33-46. J. Wiley and Sons.

[77] Suárez-Figueroa, M.C., Brockmans, S., Gangemi, A., Gómez-Pérez, A., Lehmann, J., Lewen, H., Presutti, V. & Sabou, M. (2007), "*D5.1.1: NeOn modelling components*", In NeOn Deliverable., NeOn Deliverable.

[78] Suárez-Figueroa, M.C., de Cea, G.A., Buil, C., Dellschaft, K., Fernández-López, M., García, A., Gómez-Pérez, A., Herrero, G., Montiel-Ponsoda, E., Sabou, M., Villazon-Terrazas, B. & Yufei, Z. (2008), "*D5.4.1: NeOn Methodology for Building Contextualized Ontology Networks*", In NeOn Deliverable., NeOn Deliverable.

[79] Swiss, R.V., uwe Kietz, J., Volz, R. & Maedche, A. (2000), " *AI for the Web - Ontology-Based Community Web Portals* ", In Proc of the 2nd Learning Language in Logic (LLL) Workshop, Lisbon., pp. 167-175.

[80] Uschold, M. & King, M. (1995), "*Towards a Methodology for Building Ontologies*", In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95.

[81] Uschold, M. & Gruninger, M. (1996), "*Ontologies: Principles, Methods and Applications*", Knowledge Engineering Review. Vol. 11, pp. 93-136.

[82] Vega, J. Cé.A., Gómez-Pérez, A., Tello, A.L. & Pinto, H.S.A.N.P. (1999), "*How to Find Suitable Ontologies Using an Ontology-Based WWW Broker*", In IWANN (2)., pp. 725-739.

[83] Villazón-Terrazas, B., Angeletou, S., García-Silva, A., Gómez-Pérez, A., Maynard, D., Suárez-Figueroa, M.C. & Peters, W. (2009), "*D2.2.2 Methods and Tools Supporting Re-engineering*", In NeOn Deliverable., NeOn Deliverable.

[84] Vrandecic, D., Pinto, H.S., Sure, Y. & Tempich, C. (2005), "*The DILIGENT Knowledge Processes*", Journal of Knowledge Management., October, 2005. Vol. 9(5), pp. 85-96.

[85] Antoniou, G. & vanHarmelen, F. (2004), "*A Semantic Web Primer*" Cambridge, MA, USA MIT Press.

[86] Gruber, T.R. (1995), "*Toward principles for the design of ontologies used for knowledge sharing*", Int. J. Hum.-Comput. Stud.. Vol. 43(5-6), pp. 907-928.

[87] Suárez-Figueroa, M.C., Gómez-Pérez, A. & Villazón-Terrazas, B. (2009), "*How to Write and Use the Ontology Requirements Specification Document*", In OTM '09: Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems. Berlin, Heidelberg., pp. 966-982. Springer-Verlag.

[88] Sabou, M., Angeletou, S., dAquin, M., Barrasa, J., Dellschaft, K., Gangemi, A., Lehman, J., Lewen, H., Maynard, D., Mladenic, D., Nissim, M., Peters, W., Presutti & V., Villazón, B.

(2007), ”*Selection and integration of reusable components from formal or informal specifications*”, Technical report, NeOn project deliverable D2.2.1
[89] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, & C. Ursu (2002) , “*The GATE User Guide*”, http://gate.ac.uk/, 2002.
[90] Fox, M.S. (1992),”*The TOVE Project: A Common-sense Model of the Enterprise*”, in Belli F, Radermacher FJ (eds) Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. (Lecture Notes in Artificial Intelligence LNAI 604) Springer-Verlag, Berlin, Germany
[91] Brank, J., Grobelnik, M. & Mladenic, D. (2005), "A survey of ontology evaluation techniques", In In In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005).
[92] Resource Description Framework (RDF). 2004; Available from: http://www.w3.org/RDF/.
[93] Beckett, D. New Syntaxes for RDF. 2003; Available from: http://www.dajobe.org/2003/11/new-syntaxes-rdf/paper.html.
[94] An XML Syntax for RDF: RDF/XML. Available from: http://www.w3.org/TR/REC-rdf-syntax/#rdfxml.
[95] RDF Schemas and Namespaces. Available from: http://www.w3.org/TR/PR-rdf-syntax/#schemas.
[96] RDF Validation Service. 2007; Available from: http://www.w3.org/RDF/Validator/.
[97] RDF Vocabulary Description Language 1.0: RDF Schema. 2004; Available from: http://www.w3.org/TR/rdf-schema/.
[98] SPARQL Query Language for RDF. 2006; Available from: http://www.w3.org/TR/rdf-sparql-query/.
[99] OWL Web Ontology Language. 2004; Available from: http://www.w3.org/TR/owl-features/.
[100] Broekstra, J. & Kampman, A. (2003), "The SeRQL Query Language". Aduna, 2003.
[101] Broekstra, J., Kampman, A. & van Harmelen, F. (2002), "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema", In ISWC 2002: Proceedings of the First International Semantic Web Conference, Sardinia, Italy., pp. 54-68.
[102] The Protégé Ontology Editor and Knowledge Acquisition System. 2006; Available from: http://protege.stanford.edu/.
[103] What is Protégé? A Protégé Overview. 2008; Available from: http://protege.stanford.edu/.
[104] Open Knowledge Base Connectivity. 1995; Available from: http://www.ai.sri.com/~okbc/.
[105] What is protégé-frames? A Protégé Overview. 2008; Available from: http://protege.stanford.edu/overview/protege-frames.html.
[106] What is protégé-owl? A Protégé Overview. 2008; Available from: http://protege.stanford.edu/overview/protege-owl.html.
[107] Brill, E. (1992), "A simple rule-based part of speech tagger", In Proceedings of the Third Conference on Applied Natural Language Processing.
[108] A. Polyvyanyy & D. Kuropka (2007). A quantitative evaluation of the enhanced topic-based vector space model. Universitätsverlag Potsdam.
[109] RDF/XML Syntax Specification. 2004; Available from: http://www.w3.org/TR/REC-rdf-syntax/