NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — “Semantic-based knowledge and content systems”

D2.2.2 Methods and Tools Supporting Re-engineering

Deliverable Co-ordinator: Boris Villazón-Terrazas

Deliverable Co-ordinating Institution: Universidad Politécnica de Madrid (UPM)

Other Authors: Sofia Angeletou (OU), Andrés García-Silva (UPM), Asunción Gómez-Pérez (UPM), Diana Maynard (USFD), Mari Carmen Suárez-Figueroa (UPM), and Wim Peters (USFD)

This deliverable describes methods and tools for re-engineering a wide range of knowledge-aware resources (e.g. classification schemes, thesauri, folksonomies, text) into ontologies so that they can be integrated in the development of ontologies.

Document Identifier:	NEON/2008/D2.2.2/v2.0	Date due:	December 31, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 13, 2009
Project start date	March 1, 2006	Version:	v2.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dcs.shef.ac.uk
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Food and Agriculture Organization of the United Nations (FAO)** Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com	**Laboratorios KIN, S.A. (KIN)** C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- Universidad Politécnica de Madrid (UPM)
- Open University (OU)
- University of Sheffield (USFD)

Change Log

Version	Date	Amended by	Changes
0.1	21-07-2008	Boris Villazón-Terrazas	TOC
0.2	07-09-2008	Boris Villazón-Terrazas	The Introduction and State of the Art chapters added
0.3	08-09-2008	Sofia Angeletou	The Folksonomy chapter added
0.4	17-09-2008	Boris Villazón-Terrazas and Mari Carmen Suárez-Figueroa	Internal revision
0.5	19-09-2008	Diana Maynard	The Named Entities chapter added
0.6	30-09-2008	Sofia Angeletou	The Folksonomy chapter updated
0.7	07-10-2008	Boris Villazón-Terrazas	The Classification Scheme chapter added
0.8	15-10-2008	Diana Maynard	The Named Entities chapter updated
0.9	15-11-2008	Boris Villazón-Terrazas and Mari Carmen Suárez-Figueroa	Internal revision
1.0	18-11-2008	Diana Maynard	The Named Entities chapter updated
1.1	26-11-2008	Diana Maynard	The Named Entities chapter updated
1.2	01-12-2008	Boris Villazón-Terrazas	The Thesauri chapter added
1.3	16-12-2008	Diana Maynard	The Named Entities chapter updated
1.4	16-12-2008	Diana Maynard	The Conclusions chapter updated
1.5	16-12-2008	Sofia Angeletou	The Conclusions chapter updated
1.7	19-12-2008	Boris Villazón-Terrazas	The Introduction and Conclusions chapters updated
1.8	10-01-2009	Boris Villazón-Terrazas and Asunción Gómez-Pérez	Internal revision
1.9	20-01-2009	Boris Villazón-Terrazas and Asunción Gómez-Pérez	Internal revision
2.0	13-02-2009	Boris Villazón-Terrazas	Updates based on comments from the internal reviewer

Executive Summary

With the goal of speeding up the ontology development process, ontology engineers are starting to reuse as much as possible available ontological resources and non-ontological resources, such as classification schemes, thesauri, and folksonomies that already have some degree of consensus. The reuse of such non-ontological resources necessarily involves their re-engineering into ontological resources. Non-ontological resources are highly heterogeneous in their data models and contents: they encode different types of knowledge, and can be modeled and implemented in different ways. In order to support and promote such reuse and re-engineering based approach, new methods, techniques and tools are needed.

The main goal of this deliverable is to present a set of methods and tools for re-engineering non-ontological resources into ontologies. Thus

- As for classification schemes and thesauri, we present methods based on a re-engineering model that we have adapted from the general model for software re-engineering. The methods rely on the use of Patterns for Re-engineering Non-ontological Resources (PR-NOR). These methods are extensions of the methods presented on D.5.4.1 [SFdCB+08].

- With respect to folksonomies, we present a method with its respective tool that creates an ontological structure for a folksonomy of a specific domain by employing automatically selected knowledge from online available ontologies.

- With respect to corpora, we present a method and a tool that identify patterns for the extraction of entities from unstructured text, and re-engineers these entities into concepts and instances (ontology creation and population).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The development of ontologies in different international and national projects has revealed that there are different alternative ways or possibilities to build ontologies. Just to name a few of them, in the Esperonto[1] project ontologies were built from scratch; in Knowledge Web[2] the issues dealt with were the aligning and versioning of ontologies as well as the use of best practices or patterns related to W3C activities; in the SEEMP[3] project the development of ontologies is based on the reuse of non-ontological resources, e.g., human resources standards, language classifications, etc. Thus, it is not premature to affirm that a new ontology development paradigm is being born, whose emphasis is placed on the reuse and subsequent re-engineering of knowledge-aware resources, as opposed to custom-building new ontologies from scratch. In order to support and promote such re-engineering-based approach, new methods, techniques, and tools are needed. Therefore, the main goal of this deliverable is to contribute to this new paradigm by presenting a set of methods and tools for re-engineering non-ontological resources into ontologies.

1.1 WP2 Objectives and Main Tasks

Workpackage 2, *Collaborative Aspects for Networked Ontologies*, investigates the collaborative aspects of ontology development and reuse. The main objectives of WP2 is, on the one hand, to analyze and describe the activities underlying collaborative design of networked ontologies and, on the other hand, to produce appropriate methods and tools for supporting the related workflow, by focusing on the collaborative aspects underlying the work of a knowledge community. One of the tasks involved in this workpackage is *T2.2 Methods and tools for collaborative engineering ontologies*, which provides methods and tools to support re-engineering, evaluation and selection in the context of building networked ontologies. Within T2.2 we can distinguish the following two subtasks:

- T2.2a Methods and tools for evaluating and selecting ontology components, which includes (1) design pattern based evaluation of ontology modules; (2) evaluation of ontology statements reuse in Watson plugin; and (3) collaborative evaluation of knowledge assets using Open Rating Systems. The associated deliverable of this task is D2.2.3 Methods and tools for ontology evaluation and selection.

- T2.2b Methods and tools for re-engineering non-ontological resources, which includes (1) implementation of methods for re-engineering folksonomies to ontologies; (2) creation of re-engineering patterns for transforming non-ontolgical resources into an ontology; (3) extension of the R_2O and ODEMapster for re-engineering database content and populating ontologies with instances derived in this way; (4) learning support for semi-automatic ontology construction; (5) implementation of term and named entity extraction; (6) implementation of a component for LSA and sense tagging. The results of all

[1] http://www.esperonto.net
[2] http://knowledgeweb.semanticweb.org
[3] http://www.seemp.org

these are reported in this deliverable, except (3), which is reported in *D6.10.2 Updated NeOn Toolkit plugins*, and (6), which is reported in *D2.5.2 Library of ontology design patterns and software support for pattern-based design*.

1.2 Deliverable Main Goals and Contributions

When compared with the previous version of this deliverable, it can be noted that the improvements of this version are mainly centered on evolving existing methods and investigating novel ones for re-engineering non-ontological resources into ontologies. Therefore, the main goal of this deliverable is to present a set of methods and tools for re-engineering non-ontological resources into ontologies. Thus

- For handling classification schemes, and thesauri, we present methods based on a re-engineering model for non-ontological resources that we have adapted from the general model for software re-engineering. These methods rely on the use of Patterns for Re-engineering Non-ontological Resources (PR-NOR). The methods are extensions of the methods presented on D.5.4.1 [SFdCB+08].
- With respect to folksonomies, we present a method with its respective tool that creates an ontological structure for a folksonomy of a specific domain by utilising automatically selected knowledge from online available ontologies.
- For corpora we present a method and a tool that identify patterns for the extraction of entities from unstructured text and re-engineer these entities into concepts and instances (ontology creation and population).

1.3 Deliverable Structure

The material of this deliverable is structured as follows:

- **Chapter 2** presents a review of the state of the art of methods and tools for re-engineering non-ontological resources into ontologies.
- **Chapter 3** introduces the general method, based on patterns, we follow in NeOn for re-engineering non-ontological resources. We apply this method to classification schemes (chapter 4), and thesauri (chapter 5).
- **Chapter 4** presents a method for re-engineering classification schemes into ontologies. The method proposes the use of a set of patterns for re-engineering classification schemes into ontologies.
- **Chapter 5** describes a method for re-engineering thesauri into ontologies. The method proposes the use of a set of patterns for re-engineering thesauri into ontologies.
- **Chapter 6** describes a method and a tool to create an ontological structure for a folksonomy of a specific domain by utilising automatically selected knowledge from online available ontologies. Since folksonomies are unstructured non-ontological resources, this chapter do not follow the pattern based method.
- **Chapter 7** presents a description of the methods we use in order to identify patterns for the extraction of entities from unstructured text, and to re-engineer these into concepts and instances (ontology creation and population). Since unstructured text is unstructured non-ontological resource, this chapter do not follow the pattern based method.
- **Chapter 8** describes the conclusions and future work.

1.4 Relation with the Rest of WPs within the NeOn Project

It is notable that the work reported in *this deliverable* has been tightly integrated with the efforts of other work packages in the project. In particular

- **WP1.** The resultant ontology, after the re-engineering process, will follow the networked model proposed on WP1.
- **WP5.** The methods proposed within this deliverable will be included in the NeOn Methodology for building ontology networks, specifically in the scenario for reusing and re-engineering non-ontological resources.
- **WP6.** The tools proposed in this deliverable, which give technological support to the methods, will be integrated into the NeOn ToolKit.
- **WP7** and **WP8.** Methods and tools provided in this deliverable will be applied to the NeOn Project use cases.

Chapter 2

State of the Art

During the last years, the research community has been very active in the ontology engineering field, and more recently, in reusing and re-engineering knowledge-aware resources for building ontologies rather than building them from scratch. In this document knowledge-aware resources include classification schemes, thesauri, lexica, and folksonomies. The aim of this deliverable is to look for new methods, techniques, and tools for reusing and re-engineering the terminology contained in the available knowledge-aware resources.

In this chapter we present a comparative study of the most outstanding methods and tools for re-engineering non-ontological resources into ontologies. To carry out this study we have established a common framework with which to compare the main characteristics of the different methods and tools. This chapter is partially based on the D2.2.1 [SAd$^+$07], and includes additional research works not considered in D2.2.1.

The chapter is organized as follows: first we present the non-ontological resource typology. In section 2.2, we introduce a framework for evaluating the methods and tools employed for re-engineering non-ontological resources. Section 2.3 describes the methods for re-engineering non-ontological resources. Section 2.4 deals with the tools available for re-engineering non-ontological resources. Finally, section 2.5 presents the results and conclusions of the methods and tools evaluated.

2.1 Types of Non-Ontological Resources

Non-ontological resources (NORs), which were defined in D5.4.1 [SFdCB$^+$08], are knowledge-aware resources whose semantics have not been formalized yet by an ontology. There is a big amount of non-ontological resources that embody knowledge about some particular domains and that represent some degree of consensus for a user community. These resources are present the form of textual corpora, classifications, thesauri, lexicons and folksonomies, among others. Non-ontological resources have related semantics that allows interpreting the knowledge they contain. Regardless of whether the semantics 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 them from being used as ontologies. Using the non-ontological resources as ontologies can have several benefits, e.g. interoperability, browsing/searching, and reuse among others.

The analysis of the literature has revealed that there are different ways of categorizing non-ontological resources [MS01, SAd$^+$07, GPS98, Hod00]. Thus Maedche et al. [MS01] and Sabou et al. [SAd$^+$07] classify non-ontological resources into unstructured (e.g. free text), semi-structured (e.g. folksonomies) and structured (e.g. databases) resources; whereas Gangemi et al. [GPS98] distinguish catalogues of normalized terms, glossed catalogues, and taxonomies; finally, Hodge [Hod00] proposes characteristics such as structure, complexity, relationships among terms, and historical functions for classifying them. However, an accepted and agreed on typology of non-ontological resources does not exist yet.

In this deliverable we have updated the previous typology presented in D5.4.1 [SFdCB$^+$08], included dictionaries within the lexicon category. The categorization of non-ontological resources is presented according to three different features: (1) the type of non-ontological resource, which refers to the type of inner organization of the information; (2) the data model, that is, the design data model used to represent the knowledge encoded by the resource; and (3) the resource implementation.

Fig. 2.1 shows the three levels of the categorization of non-ontological resources: type, data model and implementation levels.

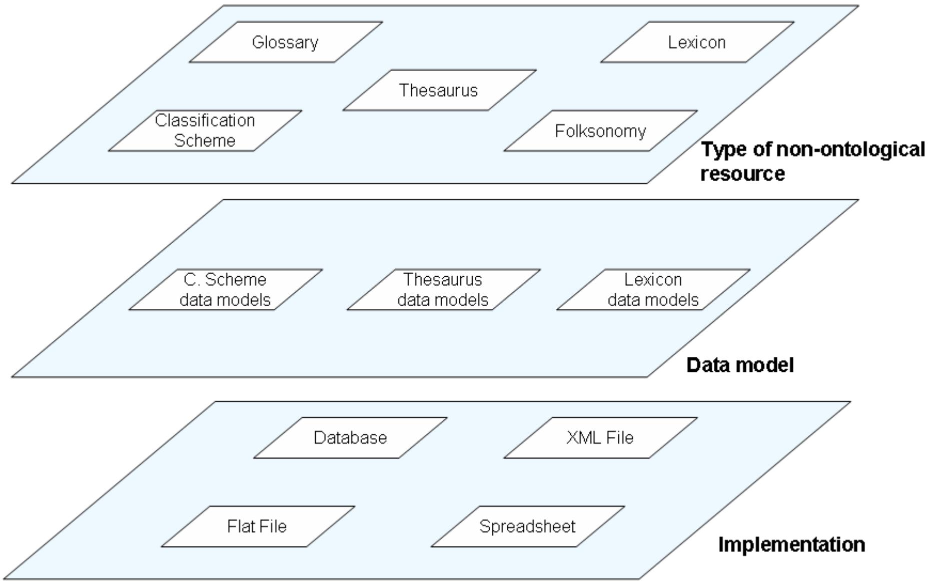


Figure 2.1: Non-Ontological Resources Categorization

1. According to the **type of non-ontological resource** we classify them into

    - *Glossaries*: A glossary is an alphabetical list of terms or words found in or relating to a specific topic or text. It may or may not include explanations, and its vocabulary may be monolingual, bilingual or multilingual [WB97]. An example of glossary is the FAO Fisheries Glossary[1].
    - *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 behaviour information [Hir04]. An example is WordNet[2], 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 that the objects have in common [ISO04]. For example, the Fishery International Standard Statistical Classification of Aquatic Animals and Plants (ISSCAAP)[3].
    - *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 retrieving of articles in large databases [ISO86]. An example of thesaurus is the AGROVOC[4] thesaurus.
    - *Folksonomies*: Folksonomies are Web 2.0 systems that allow users to upload and annotate their content effortlessly and without requiring any expert knowledge. This simplicity has made folksonomies widely successful that has resulted in a massive amount of user-generated and user-annotated web content. The main advantage of folksonomies is the implicit knowledge they contain. When users tag resources with one or more tags, they assign these resources the meaning

[1] http://www.fao.org/fi/glossary/default.asp
[2] http://wordnet.princeton.edu/
[3] http://www.fao.org/figis/servlet/RefServlet
[4] http://www.fao.org/agrovoc/

of the tag. Furthermore, the co-occurrence of tags implies a semantic correlation among them. An example of how folksonomies are used can be seen in the *del.icio.us*[5] website.

2. There are different ways of representing the knowledge encoded by the resource. A data model [Car02] is an abstract model that describes how data is represented and accessed. There are three basic styles of a data model: (1) the conceptual data model, which presents the primary entities and relationships of concern to a specific domain, (2) the logical data model, which depicts the logical entity types, the data attributes describing those entities, and the relationships between entities, and (3) the physical data model, which is related to a specific implementation of the resource. In this deliverable we will use the term **data model** when referring to the logical data model. In the following chapters we present several *data models* for each of the non-ontological resources, i.e. the data model can be different even for the same type of non-ontological resource.

3. According to the **implementation** we can classify non-ontological resources into

   - *Databases*: A database is a structured collection of records or data that is stored in a computer system.
   - *XML file*: eXtensible Markup Language is a simple, open, and flexible format used to exchange a wide variety of data on and off the Web. XML is a tree structure of nodes and nested nodes of information in which the user defines the names of the nodes.
   - *Flat file*: A flat file is a file that is usually read or written sequentially. In general, a flat file is a file containing records that have no structured inter-relationships.
   - *Spreadsheets*: An electronic spreadsheet consists of a matrix of cells into which a user can enter formulas and values.

2.2 Evaluation Framework

The goal of this section is to set up a framework for comparing the existing research works (methods and tools) and re-engineering non-ontological resources. Next, we present the identified characteristics grouped according to the non-ontological resource, the transformation process, and the resultant ontology.

2.2.1 Characteristics of the Non-ontological Resource

- ***Type*** of the non-ontological resource. According to the typology introduced in [SFdCB$^+$08], and updated in section 2.1 of this document, non-ontological resources can be (1) classification schemes, (2) folksonomies, (3) glossaries, (4) lexica and (5) thesauri.

- ***Implementation*** of the non-ontological resource. According to the typology presented in [SFdCB$^+$08], and updated in section 2.1, non-ontological resources can be implemented in (1) databases, (2) XML files, (3) flat files, or (4) spreadsheets.

- The research work has the ability to transform a ***specific*** non-ontological resource into an ontology or to transform ***any*** non-ontological resource.

- The research work tackles the non-ontological resource ***data model*** information. The data model depicts the logical entity types, the data attributes describing those entities, and the relationships between entities [Car02].

- The research work deals with the ***provenance*** information of the non-ontological resource. Provenance focuses on describing and understanding where and how data is produced, the actors involved

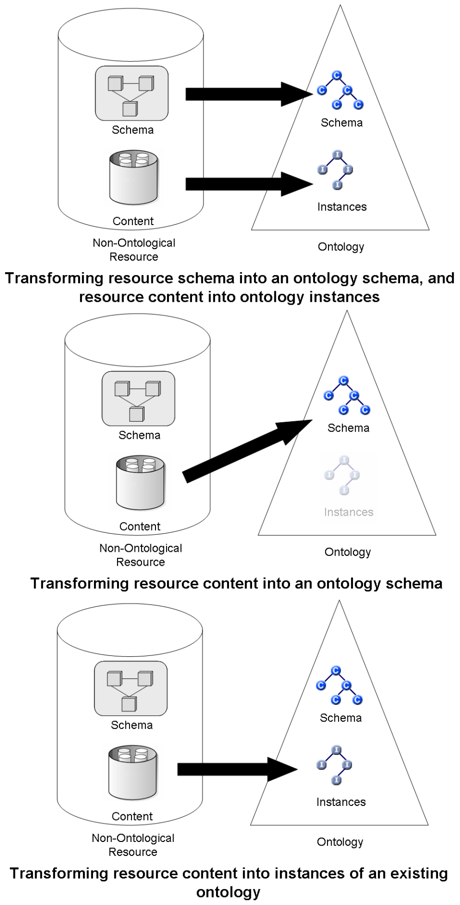
[5] http://del.icio.us/

in its production, and the processes applied before data arrived in the collection from which it is now accessed [GPC08]. In the context of this deliverable we define provenance as the reference to the non-ontological resource component for every generated ontology element, e.g., class, property, etc. For instance, an ontology attribute holds the reference to the non-ontological resource component.

2.2.2 Characteristics of the Transformation Process

This section presents the identified characteristics that are related to the transformation process. This section is divided in specific and general characteristics.

- Specific characteristics
  - The transformation process follows either (1) a ***one-step transformation*** of the resource, that is, it converts the overall non-ontological resource into an ontology, or (2) an ***incremental transformation***, that is, it converts specific components of the resource into an ontology, without applying a whole transformation.
  - The transformation process follows the ***transformation approach*** of (1) transforming the resource schema into an ontology schema, and the resource content, into ontology instances; (2) transforming the resource content into an ontology schema; or (3) transforming the resource content into instances of an existing ontology. Figure 2.2 depicts each of the possible transformation approaches.



Transforming resource schema into an ontology schema, and resource content into ontology instances

Transforming resource content into an ontology schema

Transforming resource content into instances of an existing ontology

Figure 2.2: Transformation approaches

  - The transformation process can be (1) ***automatic***, (2) ***semi-automatic*** or (3) ***manual***.

  - The transformation process is carried out by using either (1) an ***ad-hoc wrapper***, or (2) 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 could be declarative or not.
  - The transformation process handles the ***semantics of the non-ontological resource relationships*** between the non-ontological resource components (e.g. *subClassOf*, *partOf*, etc).
  - 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 on the resultant ontology [vAGS06].
  - The transformation process uses ***additional resources*** to carry out the conversion.

- General characteristics

  - The ***transformation aspects*** are contemplated at the (1) syntactic or (2) semantic levels. The ***syntactic level*** deals with the *ability to structure the representation in structured sentences, formulas or assertions*. The syntactic level includes the transformations of resource component definitions, according to the grammars of the source and target formats [Cor05]. The ***semantic level*** deals with the *ability to construct the propositional meaning of the representation* [Cor05].
  - The research work provides some ***methodological guidelines*** to support the transformation process.
  - The list of ***employed techniques*** serves to guide the transformation process, e.g., mapping rules, re-engineering patterns.
  - If a specific ***tool*** is provided, then it should give technological support to the transformation process.

2.2.3 Characteristics of the Resultant Ontology

- The generated ***ontology components*** are classes, attributes, relations, or instances.
- The ***ontology implementation language***: OWL, RDF(S).
- The research work generates a ***single ontology*** or ***several ontologies***. We do not distinguish if the ontologies generated are interconnected or not.

2.3 Non-ontological Resource Re-engineering Methods

In this section, we describe the most significant methods for re-engineering non-ontological resources taking into account the characteristics identified in section 2.2. This section is divided in two subsections: methods centered on the non-ontological resource type, section 2.3.1, and methods centered on the non-ontological resource implementation, section 2.3.2.

2.3.1 Methods Centered on the Non-ontological Resource Type

In this section we present the most outstanding methods we have found in the literature relating to the re-engineering of the following non-ontological resources: classification schemes, folksonomies, lexica and thesauri.

Methods for Transforming Classification Schemes into Ontologies

The two main methods for transforming classification schemes are GenTax [HdB07] and Hakkarainen et al.'s method [HHST06]. Next both methods are described.

- **GenTax** is a method presented by Hepp et al. in [HdB07] 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. Hepp et al. 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 **transforms semi-automatically the entire resource content into an ontology schema**. Human intervention in the transformation is limited to checking some conceptual properties and identifying frequent anomalies. The basic idea of this method is to derive two ontology classes (1) one generic concept and (2) one broader taxonomic concept from each category. The method **employs an *ad-hoc* wrapper for the transformation**. Gentax **deals with syntactic transformation aspects** and how symbols are structured in the non-ontological resource and ontology formats. Further, Gentax **contemplates semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. However, this method **does not tackle the internal data model of the resource**. On the other hand, how the resource data is represented and accessed for the transformation is not described. GenTax **does not keep the resource provenance information**, so the resultant ontology does not keep the reference to the non-ontological resource.

  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 relation** 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-DLP or RDF(S)**.

- **Hakkarainen et al.** [HHST06] present a study of the semantic relationship between the ISO 15926-2[6] and OWL DL. The ISO 15926-2 specifies a data model that semi-formally defines the meaning of the life-cycle information in a single context supporting the views engineers, equipment engineers, operators, maintenance engineers and other specialists. The **ISO 15926-2** is built on EXPRESS[7], and stored in a **flat file**, to specify its data model. The standard consists of 201 entity data types; the top level entity data type is *thing*, with its subtypes *possible_individual* and *abstract_object*. All other entities are subtypes of them.

  This method consists of (a) two transformation protocols, which are based on **transformation rules**, and (b) two inverse transformation protocols, with the purpose of examining the possible loss of semantics. Transformation protocols include **a formal specification of the conversions**. These protocols are based on the approach **to transforming resource schema into an ontology schema, and resource content into instances of the ontology**. Then, the protocols translate **the relations between resource components into *subClassOf* and *ad-hoc* relations**. However, not a single specific tool is

[6] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29557

[7] the EXPRESS file is a computer-interpretable of ISO 15926-2 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38047

mentioned. This method **does not keep the provenance information** of the resource, and therefore, the resultant ontology does not keep the reference to the non-ontological resource.

The defined transformation protocols are

- TM1, which considers semantic interpretation of the ISO 15926-2 components straightforward; basically, one ISO 15926-2 component corresponds to exactly one OWL primitive. TM1 is most appropriate if a one-to-one mapping is desired.
- TM2, which considers extended semantic interpretation; it also takes into account the semantics of each instance of the components; basically, one ISO 15926-2 component corresponds to several OWL primitives and viceversa. TM2 is most appropriate if the transformation is performed to take advantage of the reasoning facilities provided by OWL, thus adding functionality not natively present in ISO 5926-2.

The transformation protocols manage a **single ontology**. The ontology components generated are **classes, attributes, and relations**. The resultant ontology is expressed in **OWL DL**.

Methods for Transforming Folksonomies into Ontologies

The two main methods for transforming folksonomies are T-ORG [ASC07], by Abbasi et al., and [MDA07a] by Maala et al. Next we describe both of them.

- **Abbasi et al.** [ASC07] 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 required categories. The authors use **lexico-syntactic patterns** and Google API for searching the appropriate categories of the tags. This method follows the approach **to transforming the resource content into instances of an existing ontology**, and their authors have implemented the **T-ORG tool**, described in section 2.4, 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 is not described. This method **does not keep the resource provenance information**, therefore, the resultant 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

  - Selecting the ontology. The user selects the ontologies relevant to the categories. Concepts from these ontologies are used as categories. The authors rely on Swoogle[8] 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.
  - 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.

[8] `http://swoogle.umbc.edu`

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.

- **Maala et al.**'s method [MDA07a] depicts a conversion process from Flickr[9] tags to RDF descriptions. The authors present a method to automatically convert a set of tags into a RDF description in the context of photos on Flickr. However, it must be noted that they do not mention at all the implementation of the Flickr tags.

  In this method the authors analyze the tagging habits and the tagging content of the photos. To accomplish this, they rely on additional resources for the conversion such as (1) WordNet, which has been completed with extra information, and (2) place resources, a database containing geographical locations and an ontology of things. This method follows the approach **to transforming the resource content into ontology instances**, an automatic method; however, not a single specific tool is mentioned. The method uses **a formal specification of the conversion** between entities of the resource and the ontology. It **does not tackle the internal data model of the folksonomy**, and nor does it describe how the resource data is represented and accessed for the transformation. This method **does not keep the resource provenance information**, so the resultant ontology does not keep the reference to the non-ontological resource. The method takes into account **syntactic transformation aspects**, and considers how symbols are structured in the non-ontological resource and ontology formats, but it **does not consider fully semantic transformation aspects**, it only considers a small set of the semantic interpretation of the resource elements when defining transformations to ontology elements. The steps of the method are the following:

  - A tag is transformed into its non inflectional form using a stemmer.
  - Then, each tag is categorized in one of the following six categories: location, time, event, people, camera, and activity.
  - All tags grouped in the aforementioned categories are ordered from the smallest to the largest, a precise order measure is not provided.
  - For each tag a triple (*r*,*category*,*e*) is created; where the photo is denoted by *r*, the *category* can be one the six aforementioned categories, and the tag is denoted by *e*.

  This method generates a **single ontology** and the ontology components generated are **instances**; these are expressed in **RDF**.

Methods for Transforming Lexica into Ontologies

The two main methods for transforming lexica are presented in [vAGS06] and [GNV03, GGMO03] and both of them are focused in WordNet. In the following both methods are described.

- **van Assem et al.**'s method [vAGS06] proposes a standard conversion of WordNet [Fel98] into the RDF/OWL representation language. WordNet is used mainly for annotation and retrieval in different domains and it is also used to ground other vocabularies such as FOAF[10]. The three core concepts in WordNet are the following: (1) the *synset*, which groups word senses with a synonymous meaning and has four disjoint types of synset: nouns, verbs, adjectives and adverbs; (2) the *word sense*, which gives a specific sense to a word when it is used; and the *word*. WordNet defines seventeen relations, of which ten define relations between synsets, five between word senses, one between a synset and a sentence, and one between a sysnset and a verb construction pattern.

  This method is based on version 2.0 of Princenton's **WordNet Prolog distribution**[11]. The version contains documentation of the source files, of which there are eighteen: one file represents synsets, word

[9] http://www.flickr.com/
[10] http://xmlns.com/foaf/0.1/
[11] http://wordnet.princeton.edu/obtain

senses and words and seventeen for each relationship. This method **tackles the internal data model of the lexicon**, and devises how the lexicon data is represented and accessed for the transformation. The method also **provides resource provenance information**, so the resultant ontology keeps the reference to WordNet.

The process for designing the conversion consists in (1) analyzing existing conversions, which helps to understand the different ways in which WordNet is used on the Semantic Web; (2) formulating the requirements; (3) analyzing the source files and documentation; (4) designing the RDF/OWL schema; (5) designing a program for converting Prolog data to RDF/OWL; (6) drafting a Working Group Note explaining the requirements and design choices; and (7) reviewing the draft note and schema/data fields.

The authors have been established the following requirements: (a) the method should be a **full conversion**; (b) it should be convenient to work with; (c) it should reflect as mush as possible the original structure of WordNet; and (d) it should provide OWL semantics while still being interpretable by pure RDF(S) tools.

The method follows the approach **to transforming resource schema into an ontology schema and resource content into instances of the ontology**; for the transformation **an *ad-hoc* wrapper** has been employed. The method takes into account **syntactic transformation aspects**, how symbols are structured in WordNet and ontology formats. But, this method **does not consider fully semantic transformation aspects**, since one of the requirements stipulates that interpretation should be avoided (requirement c). The transformation is performed automatically with the Swi-Prolog[12] tool.

The method consists in

- Creating a set of classes for each of the main components of WordNet including classes for word, synset and sense.
- Modelling words, synsets and senses belonging to WordNet as instances of the previously created classes.
- Coding part of the semantics related to each instance by means of the URIs used to identify each instance.

The method produces **one single ontology**. The ontology components generated are **classes, attributes, relations, and instances**. The resultant ontology is expressed in **RDF(S)/OWL Full**.

- **Gangemi et al.** [GNV03, GGMO03] present a method that explains how WordNet information can be bootstrapped, mapped, refined and modularized. This method employs with WordNet 1.6, which is **stored in relational databases**. This is a hybrid method because it employs top-down techniques and tools from formal ontology and bottom-up techniques from computational linguistics and machine learning. This hybrid method can automatically extract association relations from WordNet, and interpret those associations in terms of a set of conceptual relations, formally defined in the DOLCE[13] ontology. It follows the approach to **transforming the resource content into an ontology schema**. The method uses **a formal specification of the conversions** between WordNet components and the ontology ones. It **takes into account syntactic transformation aspects**, as well as how symbols are structured in the lexicon and ontology formats. It also **takes into account semantic transformation aspects**, and the semantic interpretation of the resource elements when defining transformations to ontology elements. The method **tackles the internal data model of the resource**, and describes how the resource data is represented and accessed for the transformation. This method **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to WordNet. The method consists of the following two steps:

[12] http://www.swi-prolog.org/packages/semweb.html
[13] http://www.loa-cnr.it/DOLCE.html

  - Bottom-up learning of association links (A-links). In this step WordNet glosses (natural language definitions) are analysed and A-links, between a synset and the synsets in its gloss, are created. For each gloss the following tasks are performed: (1) POS-tagging of glosses with the ARIOSTO Natural Language processor, and extraction of relevant words; (2) disambiguation of glosses by an algorithm; and (3) creation of explicit association links (A-links) from synsets.
  - Top-down learning. In this step the foundational top ontology DOLCE is used to interpret A-links (i.e. association links) in terms of axiomatic conceptual relations. This is a technique partly automatic that involves generating solutions on the basis of the available axioms and creating a specialized partition of the axioms in order to capture more domain-specific knowledge. In this step a **description-logic classifier, e.g. LOOM**[14], is used.

The method manages **one single ontology**. The generated ontology components are **classes, attributes, and relations** and the ontology is implemented in **DAML+OIL**.

Methods for Transforming Thesauri into Ontologies

The six main methods for transforming thesauri are presented in [Hah03, HS03, vAMSW04, vAMMS06, WSWS01, HVTS08, SLL$^+$04, LS06]. They are described next.

- **Hahn et al.** in [Hah03, HS03] present a method that extracts conceptual knowledge from an **informal medical thesaurus, UMLS**[15], and **semi-automatically converts** this conceptual knowledge into a formal description logics, LOOM[16]. It is an interesting to note that this method join the massive coverage offered by informal medical terminologies with the high level of *expressiveness* and *reasoning capabilities* supported by rigid knowledge representation systems in order to develop formally solid medical knowledge bases on a larger scale. The authors formalize a model of partonomic reasoning that does not exceed the expressiveness of the well-understood concept language ALC[17]. Hahn et al. aim to extract conceptual knowledge from two major subdomains of the UMLS, anatomy and pathology, in order to construct a formally sound knowledge base based on ALC-type description logic language.

  Hahn et al.'s method follows the approach used **for transforming resource content into an ontology schema**, and employs an ***ad-hoc* wrapper** for the transformation. In the whole transformation process the ontology engineer has to take decisions relating to the **syntax and semantics of the resulting representation**. This method contemplates how symbols are structured in the non-ontological resource and ontology formats, and the semantic interpretation of the resource elements when defining transformations to ontology elements. It also contemplates **the internal data model of the thesaurus**. The method provides a description of how the resource data is represented and accessed for the transformation; however, it does not provide the resource provenance information, so the resultant ontology does not keep the reference to the thesaurus. This method consists of the following steps:

  - Automatic generation of terminological expressions. Terminological axioms at the level of description logics are generated from the relational table structures (MS Access) **imported from UMLS (ASCII files)**. In this step relations such as *partOf/hasPart*, *isA* or *hasLocation* are taken into account. For partonomic modelling, Ontology Design Patterns are used.
  - Automatic consistency checking by the LOOM classifier. The raw knowledge base is then immediately checked by the description logic classifier to see whether it contains definitional cycles and inconsistencies.

[14] `http://www.isi.edu/isd/LOOM/`
[15] `http://www.nlm.nih.gov/research/umls/`
[16] `http://www.isi.edu/isd/LOOM/`
[17] ALC allows for the construction of concept hierarchies.

  - Manual restitution of consistency. If inconsistencies or cyclic knowledge structures are encountered, a biomedical domain expert resolves the inconsistencies or cycles. After that, the classifier has to be re-run for checking whether the modified knowledge base is inconsistent with the changes made.
  - Manual modification of the knowledge base. In this step relations that were not taking into account in previous steps (e.g. *siblingOf* or *associatedWith*) are included.

  The method produces a single ontology. The ontology components generated are **classes and relations**. and they are expressed in a formal **description logic system, LOOM**.

- **van Assem et al.** in [vAMSW04] present a method for converting thesauri from their native format to RDF(S) and OWL Full. This method deals with resources **implemented in (1) a proprietary text format, (2) a relational database, and (3) an XML representation**.

  The method **semi-automatically transforms the entire resource content into an ontology schema**. The authors use an ***ad-hoc* wrapper** for the transformation. The method **does not contemplate the internal data model of the thesaurus** nor does it explain how the resource data is represented and accessed for the transformation. **Nor does it inform about the resource provenance information**, so the resultant ontology does not keep the reference to the thesaurus. This method consists of the following steps:

  - Preparation. The following characteristics of a thesaurus are analysed: (1) conceptual model, (2) relation between the conceptual and digital model, (3) relations to standards, and (4) identification of multilinguality issues.
  - Syntactic conversion. This step focuses on the syntactic aspects of the conversion process from the source implementation to RDF(S). This step consists of the following substeps: (1) a structure-preserving translation that should reflect the source structure as closely as possible and should be complete; and (2) explication of the syntax of the resource.
  - Semantic conversion. In this step the class and property definitions are augmented with additional RDF(S) and OWL constraints. The output of this step should be used in applications as *a specific interpretation of the thesaurus*, not as a standard conversion. This step consists of (1) explication of semantics, which is similar to the *explication of syntax* one, but now more expressive RDF(S) and OWL constructs may be used; and (2) specific interpretations are introduced as some application-specific requirement, e.g. an application wants to treat a `broaderTerm` hierarchy as a class hierarchy.
  - Standardization. This optional step consists of mapping a thesaurus onto a standard schema. One possible option is to map to SKOS [MB05].

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

- **van Assem et al.** in [vAMMS06] present a method for converting thesauri to the SKOS [MB05] RDF/OWL schema. This SKOS schema is a proposal for a standard being developed by W3Cs Semantic Web Best Practices Working Group.

  The development of this method is based on a process with the following components:

  - The general goal of this method is to support interoperability of thesauri encoded in RDF/OWL. The requirements are the following: (1) To produce programs that convert the digital representations of a specific thesaurus to SKOS. The resulting conversion program should produce SKOS RDF. (2) **To perform a full conversion** of the thesaurus (i.e. the resultant ontology has all information that is present in the original thesaurus) as long as this does not violate the previous requirement.

  - Comparison with existing methods. Here the authors compare the goals and requirements to those existing methods to choose a suitable one. The authors compared the methods of Soergel et al. [SLL^{+}04], Miles et al. [Mil05], and van Assem et al. [vAMSW04].
  - Developing steps of the method. The method by Miles et al. [Mil05], which has a comparable goal and requirements was employed as starting point and adapted to this purpose. The outcoming steps are
    * To analyze the digital format and the documentation of the resource. The output of this step is a catalogue of data items and constraints and a list of thesaurus features.
    * To define a mapping between input data items and output SKOS RDF. The output of this step are tables that map data items to schema items.
    * To develop an algorithm for the transformation program. The output of this step is a conversion program.
  - Applying the method. The method has been applied to three thesauri: IPSV[18], GTAA[19] and MeSH[20] because they are used in practice and represent progressively complex thesauri.
  - Evaluating the method. The case studies showed that the method gives appropriate guidance in identifying common features of thesauri. However, the authors point out that conversion of concept-based thesauri should be simpler than term-based thesauri as SKOS is concept-based.

  This method follows the approach used **for transforming resource content into an ontology schema** and an ***ad-hoc* wrapper** is used for the transformation. This method **contemplates syntactic transformation aspects** and how symbols are structured in the thesaurus and ontology formats. On the other hand, the method **contemplates the semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. However, it **does not tackle the internal data model of the thesaurus**, nor does it explains how the resource data is represented and accessed for the transformation. It **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to the thesaurus.

  The method produces **one single ontology**. The ontology components generated are **classes, attributes, and relations** and they are expressed in **SKOS RDF**.

- **Wielinga et al.** in [WSWS01] present a method for transforming the **Art and Architecture Thesaurus (AAT)** into an RDF(S) ontology. The AAT is the most elaborate and most standardized body of knowledge concerning classifications of art objects. AAT is published via a **searchable online Web interface**[21] and it is also **available in XML files**.

  This method is based on the approach **for transforming resource content into an ontology schema** and employs an ***ad-hoc* wrapper**. This method **contemplates syntactic transformation aspects** and how symbols are structured in the thesaurus and ontology formats. It, also **contemplates semantic transformation aspects** since it considers the semantic interpretation of the resource elements when defining transformations to ontology elements. It **tackles the internal data model of the thesaurus** and describes how the resource data is represented and accessed for the transformation. However, it **does not inform about the resource provenance information**, so the resultant ontology does not keep the reference to the thesaurus. The method consists of the following steps:

  - To convert the full AAT hierarchy into a hierarchy of concepts where each concept has a *label* slot corresponding with the main term in AAT and a *synonyms* slot where alternate terms are represented.

[18] Integrated Public Sector Vocabulary http://www.esd.org.uk/standards/ipsv/
[19] Common Thesaurus for Audiovisual Archives http://informatieprofessional.googlepages.com/gtaa
[20] Medical Subject Headings http://www.nlm.nih.gov/mesh/
[21] http://www.getty.edu/research/conducting_research/vocabularies/aat/

  - To augment a number of concepts with additional slots and fillers, for example, concepts representing a style or period were augmented with slots *time period from*, *time period to*, *general style* and *region*.
  - To add knowledge of the relation between possible values of fields and nodes in the knowledge base.

  The method produces **one ontology**. The ontology components generated are **classes, attributes, and relations** and they are implemented in **RDF(S)**.

- **Hyvönen et al.** in [HVTS08] present a method for transforming thesauri into ontologies. The method has been applied to the **YSA thesaurus**[22]. DOLCE[23] was employed for the transformation. The authors point out that although a syntactic transformation into SKOS [MB05] can be useful, it is not enough from a semantic viewpoint. They also stated that unless the meaning of the semantic relations of a thesaurus is made more explicit and accurate for the computer to interpret, the SKOS version is as confusing to the computer as the original thesaurus. Therefore, this method for thesaurus to ontology transformation is not a syntactic one since it is done by refining and enriching the semantic structures of a thesaurus. It follows the approach used **for transforming resource content into an ontology schema**. To accomplish this, the transformation is made automatically with an *ad-hoc tool*, and then is refined by hand in order to distinguish multiple meanings and to build a full *subClassOf* hierarchy based on NT/RT relationships of thesauri. However, no information about the implementation of the YSA thesaurus is provided. This method **contemplates syntactic transformation aspects** and how symbols are structured in the thesaurus and ontology formats. It also **contemplates semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. This method **tackle the internal data model of the thesaurus**, since it describes how the resource data is represented and accessed for the transformation. The method **provides resource provenance information**, so the resultant ontology keeps the reference to the thesaurus. It is based on the following semantic refinements and extensions in the thesaurus structure:

  - Missing links in the *subClassOf* hierarchy. The *Broader Term* (BT) relations do not, usually, structure the terms into a full-blown hierarchy but into a forest of separate smaller subhierarchies. Their central structuring principle in constructing the hierarchies is to avoid multiple inheritance.
  - Ambiguity of the BT relations. BT relation may mean either *subClassOf* relation, *partOf* relation or *instanceOf* relation.
  - Non-transitivity of the BT relation. The transitivity of the BT relation chains is not guaranteed from the instance-class-relation point of view.
  - Ambiguity of concept meanings. Many terms in thesauri are ambiguous and cannot be related properly to each other in the hierarchy using the *subClassOf* relation.

  The resultant ontology, based on the YSA thesaurus, is the General Finnish Ontology YSO[24]. The ontology components generated are **classes, attributes, and relations** and they are expressed in **RDF(S)**.

- **Soergel et al.** in [SLL$^+$04], and **Lauser et al.** in [LS06] present a method for the re-engineering of traditional thesaurus, AGROVOC[25], into a fully-fledged ontology. The original **AGROVOC thesaurus** is **stored in a database**.

  Soergel et al. explore the applicability of the *rules-as-you-go approach* to improve the re-engineering process. The method is based on the approach **to transforming resource content into an ontology**

[22] http://vesa.lib.helsinki.fi/
[23] http://www.loa-cnr.it/DOLCE.html
[24] http://www.yso.fi/onto/yso
[25] http://www.fao.org/aims/ag_intro.htm

schema. The method employs an ***ad-hoc* wrapper**. It **contemplates, on the one hand, syntactic transformation aspects**, and on the other, how symbols are structured in the thesaurus and ontology formats. It also **contemplates semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. The method **tackles the internal data model of the thesaurus** and describes how the resource data is represented and accessed for the transformation. This method **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to the thesaurus. The purpose here is to have a computer-assisted approach where a human editor teaches a computer program rules for the refinement of relationship, FAO[26] has a program where some of the re-engineering is done primarily by people.

The steps of the transformation process are

- To define the ontology structure.
- To fill in values from one or more legacy KOS to the extent possible.
- To edit manually using an ontology editor and make existing information more precise by adding new information.

In order to automate the process Soerger et al. **plan to build an inventory of patterns**, namely, content ontology design patterns specific for the agricultural domain. They also review a set of specific relationships, e.g. ***subClassOf* and *ad-hoc* relations**, that can be included in the resultant ontology.

Lauser et al. present the basic OWL model, which was extracted manually from the analysis of AGROVOC schema, using the results of the Soergel et al.'s work; and they point out as future work the conversion of the AGROVOC content into ontology instances. They plan to develop a Web based tool for maintaining the resultant ontology.

The method produces **one ontology**. The generated ontology components are **classes, attributes, and relations** and they are expressed in **OWL DL**.

2.3.2 Methods Centered in the Non-ontological Resource Implementation

In this section we present the most relevant methods we have found in the literature related with the re-engineering of non-ontological resources centered in their implementation. Research works to transform databases, XML files, flat files and spreadsheet files into ontologies are commented in this section.

Methods for Transforming Databases into Ontologies

The three main method for transforming databases are presented in [SSV02, BCGP04, Bar07]. Next we describe them.

- **Stojanovic et al.** in [SSV02] present an integrated and semi-automatic approach to generating shared-understable metadata of data-intensive Web applications.

  This method is based on mapping the given relational schema into ontologies using a reverse engineering process. The method deals with any non-ontological resources **stored in a database** and **transforms the database content into instances of an existing ontology (in form of RDF files)** on demand dynamically by applying the generic mapping rules specified by the authors. A **formal specification of the conversions** between entities of the resource and the ontology is used. This method creates **mapping rules for (1)concepts, (2)inheritance, and (3) relations**. The method **contemplates syntactic transformation aspects** and how symbols are structured in the database and ontology formats. It also **contemplates semantic transformation aspects** and semantic interpretation of

[26] http://www.fao.org

the resource elements when defining transformations to ontology elements. This method **tackles the internal data model of the resource** and describes how the resource data is represented and accessed for the transformation. It **does not provide the resource provenance information**, therefore the resultant ontology does not keep the reference to the database. The proposed method consists of the following steps:

- To capture information from relational schema through reverse engineering, user interaction is necessary in this step.
- To analyze the information obtained and map database entities into ontological entities, by using a set of mapping rules. This step consists of: (1) alignment of the top-level terms; (2) use of concept creation rules to determine the set of relations in relational schema related to a concept; and (3) use of attribute creation rules to assign relation's attributes to the attributes of a concept.
- To evaluate, validate and refine the mapping.
- To create a knowledge base, i.e. data migration. This step consists of: (1) creation of the instances; and (2) definition of the relations between instances.

The intranet of the AIFB Institute[27] is here presented as a case study. For the automation of the mapping process they used **KAON-REVERSE**[28] a tool for semi-automatically connecting relational database to ontologies.

This method produces **one ontology** and generates **ontology instances**. The resultant ontology is expressed in **F-Logic**[29], the **ontology instances are expressed in RDF**.

- **Barrasa et al.** in [BCGP04, Bar07] present an integrated framework for the formal specification, evaluation and exploitation of the semantic correspondences between ontologies and relational data sources.

  The framework consists of the following two main components:

  - R_2O, which is a declarative language for the description of arbitrarily complex mapping expressions between ontology elements (concepts, attributes and relations) and relational elements (relations and attributes). The strength of the R_2O language lies in its expressivity and in its DBMS independence. The elements of the language providing such qualities are conditions, operations, and the rule-style mapping definition for attributes.
  - ODEMapster processor, which generates Semantic Web instances from relational instances based on the mapping description expressed in an R_2O document. ODEMapster offers two operation modes: **query driven upgrade** (on demand query translation) and **massive upgrade batch process** that generates all possible Semantic Web individuals from the data repository.

  This method follows the approach **to transforming resource content into instances of an existing ontology** and uses **a formal specification of the conversions** between entities of the resource and the ontology.

  This method **contemplates syntactic transformation aspects** and how symbols are structured in the database and ontology formats. It also **contemplates semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. The method **tackles the internal data model of the resource**, and it describes how the resource data is represented and accessed for the transformation. This method **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to the database. This method consists in

[27] http://www.aifb.uni-karlsruhe.de
[28] http://kaon.semanticweb.org/alphaworld/reverse/
[29] http://flora.sourceforge.net/aboutFlogic.php

  - Discovering semi-automatically mappings between the database and ontology elements, user interaction is necessary in some special cases.
  - Expressing those mappings in a formal language, R_2O.
  - Evaluating and verifying those mappings, this is done manually.
  - Exploiting those mappings for retrieving the data using ODEMapster.

  This method produces **one single ontology** and generates **ontology instances**. The ontology instances are expressed in **RDF**.

Methods for Transforming XML Files into Ontologies

The three main methods for transforming XML files into ontologies are presented in [GC05, AM05, CXH04]. Next, we describe them.

- **García et al.** in [GC05] 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 ontologies OWL validators have been used; it has also been used the XSD2OWL tool. For checking the resulting ontologies, the user interaction is necessary.
  - 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 the MPEG-7[30] XML Schemas generating a MPEG-7 ontology[31]. 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 method produces **one single ontology**. The ontology components generated are **classes, attributes, relations, and instances** and they are expressed in **RDF/OWL Full**.

- **An et al.** in [AM05] **An et al.** present a method to translating an XML web document into an instance of an OWL-DL ontology.

  Here the authors take advantage of the semi-automatic mapping discovery tool [ABM05] for the relationship between XML schema and the ontology. They define a formal model for the mapping formulas. This method follows the approach **to transforming the resource schema into the ontology schema, and the resource content into instances of the ontology**. The method uses **a formal specification of the conversions** between entities of the XML and the ontology. It **contemplates syntactic transformation aspects**, and how symbols are structured in the XML and ontology formats. It also **contemplates semantic transformation aspects**, and the semantic interpretation of the resource elements when defining transformations to ontology elements.

  The method produces **one single ontology**. The ontology components generated are **classes, attributes, relations, and instances** and they are expressed in **OWL-DL**.

[30] http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

[31] http://rhzomik.upf.edu/ontologies/mpeg7ontos

- **Cruz et al**. in [CXH04] present a method to transforming XML schema into RDF(S) ontology preserving the XML document structure, i.e., modelling the knowledge implicit in XML schema using RDF(S).

  This method follows the approach **to transforming the resource schema into the ontology schema, and resource content into ontology instances**. The method uses **a formal specification of the conversions** between entities of the XML and the ontology. A specific tool for supporting the method has been developed for this purpose. This method **contemplates syntactic transformation aspects**, and how symbols are structured in the XML and ontology formats. It also **contemplates semantic transformation aspects** and the semantic interpretation of the resource elements when defining transformations to ontology elements. It **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. This method **does not provide the resource provenance information**, therefore the resultant ontology does not keep the reference to the XML.

  The method consists of

  - Element-level transformation, which defines the basic classes and properties of the RDF(S) ontology according to the following transformations: (1) XML attribute is mapped to a Property; (2) XML Simple-type element is mapped to a Property; and (3) XML complex-type element is mapped to a Class.
  - Structure-level transformation, which encodes the hierarchical structures of the XML schema into the RDF(S) ontology. The *element-attribute* relationship is encoded as *class-to-literal* relationship, and the *element-subelement* relationship is encoded as *class-to-class* relationship in RDF(S). Besides, the authors have defined a new RDF(S) predicate *rdfx:contain* to represent the *class-to-class* relationship.
  - Query driven data migration, which transforms the query expressed in RDQL[32] into XQuery[33] query and creates the RDF instances that satisfies the query.

  The method produces **a single ontology**. The ontology components generated are **classes, attributes, and relations** and they are expressed in **RDF(S)**.

Methods for Transforming Flat Files into Ontologies

The main method to transforming flat file is presented in [FB06] and described next.

- **Foxvog et al.** in [FB06] present a method to transforming **Electronic Data Interchange (EDI)**[34] **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. There are two major EDI standards EDIFACT [Ber94], defined as an open standard by the United Nations, and ASC X12[35], primarily used in the United States. ASC X12 calls message types *Transaction Sets* which are composed of strings and loops of *Data Segments* in a specified format. Each *Data Segment* has a specified format of *Data Elements*. The method is centered on the ASC X12 standard. ASC X12 messages are **stored in flat files**.

  This method follows the approach **to transforming 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

[32] http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[33] http://www.w3.org/TR/xquery/

[34] http://www.ifla.org/VI/5/reports/rep4/42.htm#chap2

[35] http://www.x12.org/

accessed for the transformation. It **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to the flat file.

The method for transforming ASC X12 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**

2.3.3 Comparison of the Methods

Tables 2.1, 2.2 and 2.3 show the methods presented according to the characteristics related to the non-ontological resource, the transformation process and the resultant ontology.

Research **work**	**Type of resource**	**Resource implemented in**	**Specific/Any**	**Data model is known**	**Provenance information**
Hepp et al. [HdB07]	Classification scheme, thesauri	Database	Any	No	No
Hakkarainen et al. [HHST06]	Classification scheme	Flat file	ISO15926-2	Yes	No
Abbasi et al. [ASC07]	Folksonomy		Any	No	No
Maala et al. [MDA07a]	Folksonomy		Flickr	No	No
van Assem et al. [vAGS06]	Lexica	Prolog	WordNet ver. 2.0	Yes	Yes
Gangemi et al. [GNV03, GGMO03]	Lexica	Database	WordNet ver. 1.6	Yes	No
Hahn et al. [Hah03, HS03]	Thesauri	ASCII files	UMLS	Yes	No
van Assem et al. [vAMSW04]	Thesauri	proprietary text format, relational database, XML	Any	No	No
van Assem et al. [vAMMS06]	Thesauri		IPSV, GTAA, MeSH	No	No
Wielinga et al. [WSWS01]	Thesauri	XML	AAT	Yes	No
Hyvönen et al. [HVTS08]	Thesauri		YSA	Yes	Yes
Soergel et al. [SLL$^+$04, LS06]	Thesauri	Database	AGROVOC	Yes	No
Stojanovic et al. [SSV02]		Database	Any	Yes	No
Barrasa et al. [BCGP04, Bar07]		Database	Any	Yes	No
García et al. [GC05]		XML	Any	No	No
An et al. [AM05]		XML	Any	No	No
Cruz et al. [CXH04]		XML	Any	No	No
Foxvog et al. [FB06]		Flat file	EDI X12	No	No

Table 2.1: Non-ontological resource characteristics of the methods

Research work	One-step/ Incremental	Transformation approach	Automatic/ Semi-automatic/ Manual	*Ad-hoc* wrapper/ Formal specification	Transformation aspects	Semantics of NOR Relationships	Full Conversion	Additional Resources	Methodological Guidelines	Technique	Tool support
Hepp et al. [HdB07]	One-step	2	Semi-automatic	*Ad-hoc* wrapper	syntactic semantic	subClassOf, *ad-hoc* relation	Yes	No	Yes	Not mentioned	SKOS2GenTax
Hakkarainen et al. [HHST06]	One-step	1	Semi-automatic	Formal specification	syntactic semantic	subClassOf, *ad-hoc* relation	Yes	No	Yes	Transformation rules	Not mentioned
Abbasi et al. [ASC07]	One-step	3	Automatic	*Ad-hoc* wrapper	syntactic semantic	Not mentioned	Yes	Swoogle Google	Yes	Lexico Syntactic Patterns	T-ORG
Maala et al. [MDA07b]	One-step	3	Automatic	*Ad-hoc* wrapper	syntactic partially semantic	Not mentioned	Yes	WordNet Place resources	Yes	Not mentioned	Not mentioned
van Assem et al. [vAGS06]	One-step	1	Semi-automatic	*Ad-hoc* wrapper	syntactic partially semantic	Not mentioned	Yes	No	Yes	Not mentioned	Swi-Prolog
Gangemi et al. [GNV03,GGMO03]	One-step	2	Semi-automatic	Formal specification	syntactic semantic	Not mentioned	Yes	DOLCE	Yes	Natural Language techniques	Not mentioned
Hahn et al. [U.03,HS03]	One-step	2	Semi-automatic	*Ad-hoc* wrapper	syntactic semantic	subClassOf, partOf, *ad-hoc* relations	No	No	Yes	Ontology Design Patterns for partonomic modelling	*Ad-hoc* tool
van Assem et al. [vAMSW04]	One-step	2	Semi-automatic	*Ad-hoc* wrapper	syntactic semantic	subClassOf, *ad-hoc* relations	Yes	No	Yes	Not mentioned	*Ad-hoc* tool
van Assem et al. [vAMMS06]	One-step	2	Automatic	*Ad-hoc* wrapper	syntactic semantic	Not mentioned	Yes	No	Yes	Not mentioned	Swi-Prolog
Wielinga et al. [WTWS01]	One-step	2	Semi-automatic	*Ad-hoc* wrapper	syntactic semantic	Not mentioned	Yes	No	Yes	Not mentioned	*Ad-hoc* tool
Hyvonen et al. [HVTS08]	One-step	2	Semi-automatic	Not mentioned	syntactic semantic	Not mentioned	Yes	DOLCE	Yes	Not mentioned	*Ad-hoc* tool
Soergel et al. [SLL04,LS06]	One-step	2	Manual	*Ad-hoc* wrapper	syntactic semantic	subClassOf, *ad-hoc* relations	Yes	No	Yes	Not mentioned	Not mentioned
Stojanovic et al. [SSV02]	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Yes	Mapping rules	KAON-REVERSE
Barrasa et al. [BCGP04,Bar07]	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Yes	Mapping rules	ODEMapster
García et al. [GC05]	One-step	1	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Yes	Mapping rules	XSD2OWL XML2RDF
An et al. [AM05]	One-step	1	Semi-automatic	Formal specification	syntactic semantic	Not mentioned	Not mentioned	No	No	Not mentioned	Discovery tool
Cruz et al. [CXH04]	One-step	1	Semi-automatic	Formal specification	syntactic	Not mentioned	Yes	No	Yes	Mapping rules	*Ad-hoc* tool
Foxvog et al. [FB06]	One-step	1	Semi-automatic	*Ad-hoc* wrapper	syntactic semantic	Not mentioned	No	No	Yes	Not mentioned	*Ad-hoc* tool

Table 2.2: Transformation process of the methods

Research Work	Components	Implementation language	Single/Several
Hepp et al. [HdB07]	classes, relations	RDF(S) / OWL-DLP	Single
Hakkarainen et al. [HHST06]	classes, attributes, relations	OWL-DL	Single
Abbasi et al. [ASC07]	instances		Several
Maala et al. [MDA07a]	instances	RDF	Single
van Assem et al. [vAGS06]	classes, attributes, relations, instances	RDF(S) / OWL Full	Single
Gangemi et al. [GNV03, GGMO03]	classes, attributes, relations, instances	DAML+OIL	Single
Hahn et al. [Hah03, HS03]	classes, relations	LOOM / ALC	Single
van Assem et al. [vAMSW04]	classes, attributes, relations	RDF(S) / OWL Full	Single
van Assem et al. [vAMMS06]	classes, attributes, relations	SKOS RDF	Single
Wielinga et al. [WSWS01]	classes, attributes, relations	RDF(S)	Single
Hyvönen et al. [HVTS08]	classes, attributes, relations	RDF(S)	Single
Soergel et al. [SLL$^+$04, LS06]	classes, attributes, relations	OWL-DL	Single
Stojanovic et al. [SSV02]	instances	F-Logic / RDF	Single
Barrasa et al. [BCGP04, Bar07]	instances	RDF	Single
García et al. [GC05]	classes, attributes, relations, instances	OWL Full/ RDF	Single
An et al. [AM05]	classes, attributes, relations, instances	OWL-DL	Single
Cruz et al. [CXH04]	classes, attributes, relations	RDF(S)	Single
Foxvog et al. [FB06]	classes, attributes, relations, instances	CycL / OWL Full / WSML	Several

Table 2.3: Ontology characteristics of the methods

2.4 Non-ontological Resource Re-engineering Tools

In this section, we describe the most significant non-ontological resource re-engineering tools according to the characteristics identified in section 2.2. We organize this section in two subsections: tools centered in the non-ontological resource type, section 2.4.1, and tools centered in the non-ontological resource implementation, section 2.4.2. Some of these tools give support to the methods presented in section 2.3.

2.4.1 Tools Centered in the Non-ontological Resource Type

In this section we present some of the tools we found in the literature relating to the building of ontologies by re-engineering non-ontological resources. We introduce some tools to transform classification schemes, folksonomies, lexica and thesauri into ontologies are described.

Tool for Transforming Classification Schemes into Ontologies

SKOS2GenTax[36] is an online tool that converts hierarchical classifications available in the W3C SKOS[37] format into RDF(S) or OWL DL ontologies. SKOS2GenTax uses the GenTax algorithm described in [HdB07]. The input resource can be specified by its URL or it can be uploaded directly to the Web site. This resource has to be **available in SKOS RDF format**.

Tool for Transforming Folksonomies into Ontologies

T-ORG, a system to organize folksnomies by classifying the tags attached to them into predefined categories is presented by Abbasi et al. in [ASC07]. The input resource is a **flat folksonomy tagspace**. T-ORG gives technological support to the method described in [ASC07].

2.4.2 Tools Centered in the non-ontological resource implementation

In this section we present some of the tools we found in the literature related with the re-engineering of non-ontological resources centered in their implementation. We introduce some research to transform databases, XML files, spreadsheet files and flat files into ontologies.

Tools for Transforming Databases into Ontologies

The four main tools for transforming databases are KAON-REVERSE, ODEMapster, D2R Server and Top-Braid Composer. In the following we describe each one of them.

- **KAON-REVERSE**[38] is a tool that supports the reverse engineering method presented in [SSV02] for transforming **databases** into ontologies.
- **ODEMapster**[39] is the processor in charge of carrying out the exploitation of the mappings defined using R_2O [Bar07]. This tool is intended to create instances of an existing ontology on demand or in a batch processing.
- **D2R Server**[40] is a tool for publishing the content of relational databases on the Semantic Web.

[36] http://www.heppnetz.de/projects/skos2gentax/
[37] http://www.w3.org/2004/02/skos/
[38] http://kaon.semanticweb.org/alphaworld/reverse/
[39] http://parla.dia.fi.upm.es/software/index.jsp?sw=http//www.oeg-upm.net/software/software.owl#ODEMapster
[40] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server

This tool is intended to create instances of an existing ontology on demand or in a batch processing. Therefore, D2R supports the approach **to transforming resource content into instances of an existing ontology**. D2R Server performs a **semi-automatic conversion** and uses **a formal specification of the conversions** between entities of the database schema and the ontology schema. D2R Server **contemplates syntactic transformation aspects**, and how symbols are structured in the database and ontology formats. It also **contemplates semantic transformation aspects**, and the semantic interpretation of the resource elements when defining transformations to ontology elements. However, D2R Server **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. The tool **does not provide the resource provenance information**, so the resultant ontology does not keep the reference to the database. D2R Server consists of:

- A D2RQ mapping language, a declarative mapping language for describing the relation between an ontology and a relational data model.
- A D2RQ engine, a plug-in for the Jena and Sesame Semantic Web toolkits, which uses the mappings to rewrite Jena and Sesame API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks.

This tool produces **one single ontology** and generates **ontology instances**. The resultant ontology instances are expressed in **RDF**.

- **TopBraid Composer**[41] is an enterprise-class modeling environment for developing Semantic Web Ontologies. TopBraid Composer can convert databases into ontologies. This tool has a relational database importer, D2RQ[42]. It follows the approach **to transforming resource schema into an ontology schema and the resource content into instances of the ontology**. TopBraid Composer performs **a semi-automatic conversion** and uses **a formal specification of the conversions** between entities of the database schema and the ontology schema. TopBraid Composer **contemplates syntactic transformation aspects**, and how symbols are structured in the database and ontology formats. It also **contemplates semantic transformation aspects**, and the semantic interpretation of the resource elements when defining transformations to ontology elements. TopBraid Composer **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. This tool **does not provide the resource provenance information**, therefore the resultant ontology does not keep the reference to the database.

  TopBraid Composer, for converting databases into ontologies, performs the following tasks:

  - Static import schema, where tables become classes, columns become properties and link tables become object properties.
  - Dynamic import of actual data, where rows become instances on the fly, i.e. data can stay where it is.

  This tool follows, for converting XML into ontologies, the following two approaches:

  - **Transforming XML schema into an ontology schema and XML content into ontology instances**. In this case the tool employs **a formal specification of the conversions** between entities of the resource and the ontology.
  - **Transforming XML content into an ontology schema**. Here the tool employs an ***ad-hoc* wrapper** for transforming XML elements into ontology classes, and XML attributes into datatype properties.

[41] http://www.topbraidcomposer.com/

[42] http://www4.wiwiss.fu-berlin.de/bizer/d2rq/

TopBraid Composer can also convert flat files into ontologies. It transforms the **flat file columns into an ontology schema and the flat file rows into instances of the ontology**. It can also convert spreadsheets into ontologies. Its input resources are **Excel spreadsheets**. This tool follows the approach **to transforming resource schema into an ontology schema and the resource content into instances of the ontology**.

This tool produces **one single ontology**. The ontology components generated are **classes, attributes, relations, and instances**. The resultant ontology is expressed in **RDF/OWL (Full, DL or Lite)**.

Tools for Transforming XML Files into Ontologies

The main tools are XSD2OWL and XML2RDF, and TopBraid Composer, described above. In the following section we describe XSD2OWL and XML2RDF.

- **XSD2OWL** and **XML2RDF**[43] are tools that support the method for transforming XML files into ontologies [GC05]. The input files are (1) an xml schema definition (XSD) file, which describes the xml schema; and (2) an xml file, which contains the xml instances.

Tools for Transforming Flat Files into Ontologies

The four main tools for transforming flat files are TopBraid Composer, described above, ConvertToRdf, flat2rdf and Java BibTeX-To-RDF converter. Next we describe the remaining three.

- **ConvertToRdf**[44] 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 transforming 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 **contemplates semantic transformation aspects**, and the semantic interpretation of the resource elements when defining transformations to ontology elements.

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

- **flat2rdf**[45] is a simple *Perl script* that converts **classic unix text database files** into RDF.

  Its input resources are classic unix text files (e.g. `/etc/passwd`). This tool **transforms the flat file content into ontology instances**. flat2rdf performs a **semi-automatic conversion** and employs **a formal specification of the conversions** between entities of the resource and the ontology. flat2rdf **contemplates syntactic transformation aspects**, and how symbols are structured in the file and ontology formats.

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

- **Java BibTeX-To-RDF Converter**[46] allows converting **BibTeX files** to an RDF format according the SWRC ontology[47].

  The input resources are plain BiBTex files (i.e. text files). This tool **transforms the text file content into ontology instances** and performs an **automatic conversion**. It employs an ***ad-hoc* wrapper**.

[43] `http://rhizomik.net/redefer/`
[44] `http://www.mindswap.org/~mhgrove/convert/`
[45] `http://simile.mit.edu/repository/RDFizers/flat2rdf/`
[46] `http://www.aifb.uni-karlsruhe.de/WBS/pha/bib/index.html`
[47] `http://ontoware.org/projects/swrc/`

Java BibTeX-To-RDF Converter **contemplates syntactic transformation aspects**, and how symbols are structured in the file and ontology formats.

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

Tools for Transforming Spreadsheet Files into Ontologies

The three main tools for transforming spreadsheet files are TopBraid Composer, described at the beginning of this section, Excel2rdf, and RDF123. Next we describe the remaining two.

- **Excel2rdf**[48] 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 transforming resource content into instances of an existing ontology** and performs a **semi-automatic conversion** with an ***ad-hoc* wrapper**. 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** is a highly flexible open source tool for transforming semi-automatically spreadsheet data to RDF that works on CSV files and also Google spreadsheets. This tool was presented by Han et al. in [HFP$^+$06] 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 works on CSV files and also Google spreadsheets.

  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, is 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 (1) same data can be available in different domains just by associating it with different map files; and (2) 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 resultant **ontology instances** are expressed in **RDF**.

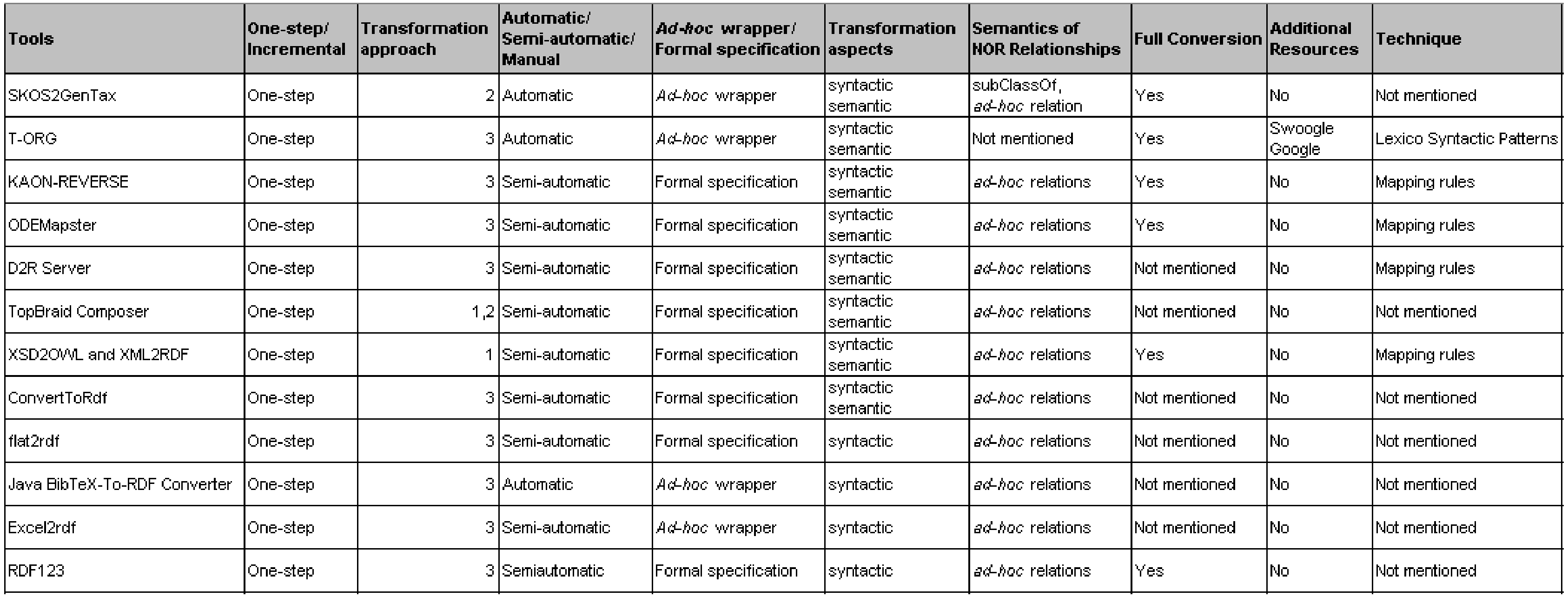
[48] http://www.mindswap.org/~Erreck/excel2rdf.shtml

2.4.3 Comparison of the Tools

Tables 2.4, 2.5 and 2.6 show the tools presented according to the characteristics related to the non-ontological resource, transformation process and resultant ontology.

Tool	Type of resource	Resource implemented in	Specific/Any	Data model is known	Provenance information
SKOS2GenTax	Classification schemes, thesauri	SKOS RDF	Any	No	No
T-ORG	Folksonomy		Any	No	No
KAON-REVERSE		Database	Any	Yes	No
ODEMapster		Database	Any	Yes	No
D2R Server		Database	Any	No	No
TopBraid Composer		Database, XML, Flat file, Spreadsheet	Any	No	No
XSD2OWL and XML2RDF		XML	Any	No	No
ConvertToRdf		Delimited text data file	Any	No	No
flat2rdf		Flat file	Unix text file	No	No
Java BibTeX-To-RDF Converter		Flat file	Bibtex file	No	No
Excel2rdf		Spreadsheet	Any	No	No
RDF123		Spreadsheet	Any	No	No

Table 2.4: Non-ontological resource characteristics of the tools



Tools	One-step/ Incremental	Transformation approach	Automatic/ Semi-automatic/ Manual	*Ad-hoc* wrapper/ Formal specification	Transformation aspects	Semantics of NOR Relationships	Full Conversion	Additional Resources	Technique
SKOS2GenTax	One-step	2	Automatic	*Ad-hoc* wrapper	syntactic semantic	subClassOf, *ad-hoc* relation	Yes	No	Not mentioned
T-ORG	One-step	3	Automatic	*Ad-hoc* wrapper	syntactic semantic	Not mentioned	Yes	Swoogle Google	Lexico Syntactic Patterns
KAON-REVERSE	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Mapping rules
ODEMapster	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Mapping rules
D2R Server	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Not mentioned	No	Mapping rules
TopBraid Composer	One-step	1,2	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Not mentioned	No	Not mentioned
XSD2OWL and XML2RDF	One-step	1	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Yes	No	Mapping rules
ConvertToRdf	One-step	3	Semi-automatic	Formal specification	syntactic semantic	*ad-hoc* relations	Not mentioned	No	Not mentioned
flat2rdf	One-step	3	Semi-automatic	Formal specification	syntactic	*ad-hoc* relations	Not mentioned	No	Not mentioned
Java BibTeX-To-RDF Converter	One-step	3	Automatic	*Ad-hoc* wrapper	syntactic	*ad-hoc* relations	Not mentioned	No	Not mentioned
Excel2rdf	One-step	3	Semi-automatic	*Ad-hoc* wrapper	syntactic	*ad-hoc* relations	Not mentioned	No	Not mentioned
RDF123	One-step	3	Semiautomatic	Formal specification	syntactic	*ad-hoc* relations	Yes	No	Not mentioned

Table 2.5: Transformation process of the tools

Tool	Components	Implementation language	Single/Several
SKOS2GenTax	classes, attributes, relations	OWL DLP/ RDF(S)	Single
T-ORG	instances		Several
KAON-REVERSE	classes, attributes, relations, instances	F-Logic / RDF	Single
ODEMapster	instances	RDF	Single
D2R Server	instances	RDF	Single
TopBraid Composer	classes, attributes, relations, instances	RDF/OWL (Full, DL or Lite)	Single
XSD2OWL and XML2RDF	classes, attributes, relations, instances	OWL Full/ RDF	Single
ConvertToRdf	instances	RDF	Single
flat2rdf	instances	RDF	Single
Java BibTeX-To-RDF Converter	instances	RDF	Single
Excel2rdf	instances	RDF	Single
RDF123	instances	RDF	Several

Table 2.6: Ontology characteristics of the tools

2.5 Results and Conclusions

After having analyzed the state of the art of methods and tools for re-engineering non-ontological resources, we present the results of applying the evaluation framework described in section 2.2. The results are provided according to the characteristics of the identified groups, that is, non-ontological resource, transformation process, and resultant ontology.

2.5.1 Results According to Non-ontological Resource

Table 2.1 and table 2.4 summarize the methods and tools presented according to the characteristics of the non-ontological resource: type of resource, resource implemented in, specific or any resource, whether the data model is known, and the provenance information.

Methods

- According to the type of non-ontological resource: one method is focused on classification schemes, other method on classification schemes and thesauri, two are focused on folksonomies, two are focused on lexica, six methods are focused on thesauri. The remaining six are not focused on the non-ontological resource type since they are independent of the resource type.

- According to the implementation of the non-ontological resource. Five methods focused on resources implemented in databases, four methods, on resources implemented in XML, three methods on resources implemented in flat files, one method deals with resources implemented in Prolog, other method deals with resources implemented in proprietary format, relational database and XML, and four methods are independent of the resource implementation.

- According to the ability to transform a specific non-ontological resource or to transform any non-ontological resource, there are ten methods with the ability to transform a specific non-ontological resource that follow: ISO15926-2[49], Flickr[50], WordNet[51], UMLS[52], IPSV[53], GTAA[54], MeSH[55], AAT[56], YSA[57], AGROVOC[58], and EDI X12[59], and eight methods that deal with any non-ontological resource.

- According to the data model information. Nine methods tackle information about the internal data model of the non-ontological resource, and nine methods do not include information about the internal data model of the resource.
- According to the provenance information. Two methods keep the reference to the non-ontological resource, and sixteen methods do not provide the provenance information about the resource after the transformation.

Tools

- According to the type of non-ontological resource. One tool deals with classification schemes and thesauri, one tool tackles folksonomies. The remaining ten tools are not focused on the non-ontological resource type since they are independent of the resource type.
- According to the implementation of the non-ontological resource. One tool takes on resources implemented in SKOS RDF; three tools operated on resources implemented in databases; one tool deals with resources implemented in databases, XML, flat files, and spreadsheets; three tools tackle resource implemented in flat files; one tool deals with XML implementation; two tools deal with resources implemented in spreadsheets. The remaining tool is not focused on the non-ontological resource implementation.
- According to the ability to transform a specific non-ontological resource or to transform any non-ontological resource. Two tools have the ability to transform a specific non-ontological resource, e.g. Unix text files and Bibtex files. The remaining ten tools deal with any non-ontological resource.
- According to the data model information. Two tools take into account information about the internal data model of the non-ontological resource. The remaining ten tools do not take information about the internal data model of the resource.
- According to the provenance information. All of the tools do not keep the reference to the non-ontological resource, they not provide the provenance information about the resource after the transformation.

We can conclude that most of the methods and tools presented are based on *ad-hoc* transformations for the resource type, the resource implementation or a specific resource. Only a few provide the provenance information of the resource, and also a few take advantage of the resource data model, an important artifact in the re-engineering process. In conclusion, we can state that there is a clear need for some sort of re-engineering methods and tools that

- deal with the overall non-ontological resources, i.e. classification schemes, thesauri, and lexica,
- take into account the internal data model of the resource,

[49] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29557

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

[51] http://wordnet.princeton.edu/

[52] http://www.nlm.nih.gov/research/umls/

[53] Integrated Public Sector Vocabulary http://www.esd.org.uk/standards/ipsv/

[54] Common Thesaurus for Audiovisual Archives http://informatieprofessional.googlepages.com/gtaa

[55] Medical Subject Headings http://www.nlm.nih.gov/mesh/

[56] http://www.getty.edu/research/conducting_research/vocabularies/aat/

[57] http://vesa.lib.helsinki.fi/

[58] http://www.fao.org/agrovoc/

[59] http://www.ifla.org/VI/5/reports/rep4/42.htm#chap2

- keep the provenance information of the resource.

2.5.2 Results According to Transformation Process

Table 2.2 and table 2.5 summarize the methods and tools presented according to the characteristics of the transformation process: one-step or incremental transformation, transformation approach, automatic/semi-automatic or manual, *ad-hoc* wrapper or formal specification, transformation aspects, semantics of NOR relationships, full conversion of the resource, use of additional resources to carry out the transformation, provision of methodological guidelines, employed technique, and tool support.

Methods

- According to the transformation process, one-step transformation or incremental transformation. The eighteen methods perform one-step transformation of the non-ontological resource.

- According to transformation approach. Six methods follow the approach to transforming the resource schema into an ontology schema and the resource content into ontology instances, eight methods follow the approach to transforming the resource content into an ontology schema, four methods follow the approach for transforming the resource content into instances of an existing ontology.

- According to whether the transformation process is automatic, semi-automatic, or manual. Three methods perform an automatic transformation, fourteen methods perform a semi-automatic transformation, and one method performs a manual transformation.

- According to how the transformation process is carried out, using an *ad-hoc* wrapper or using a formal specification of the conversions. Ten methods perform the transformation by using an *ad-hoc* wrapper, seven methods perform the transformation by using a formal specification of the conversions, and one method does not mention anything about how the transformation process is carried out.

- According to the transformation aspects contemplated, i.e. syntactic or semantic. Fifteen methods contemplate syntactic and semantic transformation aspects, since they tackle how symbols are structured in the resource and ontology formats, as well the semantic interpretation of the resource elements when defining transformations to ontology elements. Two methods contemplate syntactic and partially semantic transformation aspects. The remaining method contemplates only the syntactic transformation aspects.

- According to the semantics of the non-ontological resource relationships. Four methods deal with *subClassOf* and *ad-hoc* relations; one deals with *subClassOf*, *partOf* and *ad-hoc* relations; three methods deal with *ad-hoc* relations. The remaining ten methods do not provide information about this criteria.

- According to whether the method performs a full transformation or not. Fifteen methods perform a full conversion of the resource, two methods perform a partial conversion of the resource, and one does not provide information about this criteria.

- According to whether the method uses additional resources for the transformation or not. Four methods use additional resources, such as, Swoogle, Google, WordNet, and DOLCE ontology, for the transformation. The remaining fourteen methods do not use any additional resource for the transformation.

- According to whether the method provides some methodological guidelines or not. Seventeen methods provide methodological guidelines for the transformation, only one does not provide any methodological guideline for the transformation.

- According to the list of employed techniques. Four methods employ mapping rules for the transformation, one method employs transformation rules, other one method employs lexico syntactic patterns, one employs natural language techniques, one method employs ontology design patterns. The remaining nine methods do not provide information about this criteria.

- According to the support tool for the transformation. Eight methods have a tool that provides technological support to the transformation, six methods have an *ad-hoc* tool for the transformation. The remaining four methods do not provide information about a tool.

Tools

- According to the transformation process, i.e. one-step transformation or incremental transformation. The twelve tools perform one-step transformation of the non-ontological resource.

- According to the transformation approach. One tool follows the approach to transforming the resource schema into an ontology schema and the resource content into ontology instances, other tool follows the approach for transforming the resource content into an ontology schema, other tool follows two approaches (i) to transforming the resource schema into an ontology schema and the resource content into ontology instances and (ii) to transforming the resource content into an ontology schema. The remaining nine tools follow the approach to transforming the resource content into instances of an existing ontology.

- According to whether the transformation process is automatic, semi-automatic, or manual. Three tools perform an automatic transformation, and nine perform a semi-automatic transformation.

- According to how the transformation process is carried out, i.e. using an *ad-hoc* wrapper or using a formal specification of the conversions. Four tools perform the transformation by using an *ad-hoc* wrapper, and eight perform the transformation by using a formal specification of the conversions.

- According to the transformation aspects contemplated, i.e. syntactic or semantic. Eight tools take contemplate syntactic and semantic transformation aspects, since they tackle how symbols are structured in the resource and ontology formats, as well the semantic interpretation of the resource elements when defining transformations to ontology elements. Four tools contemplate only the syntactic transformation aspects.

- According to the semantics of the non-ontological resource relationships. One tool deals with *subClassOf* and *ad-hoc* relations, ten deal with *ad-hoc* relations and the remaining tool does not provide information about this criteria.

- According to whether the tool performs a full transformation or not. Six tools perform a full conversion of the resource, and six do not provide information about this criteria.

- According to whether the method uses additional resources for the transformation or not. One tool uses additional resources, such as, Swoogle, and Google for the transformation. The remaining eleven tools do not use any addtional resource for the transformation.

- According to the list of employed techniques. Four tools employ mapping rules for the transformation, one tool employs lexico syntactic patterns, and seven tools do not provide information about this criteria.

After having analyzed the characteristics related to the transformation process, we can conclude that research efforts have been mostly centered on using *ad-hoc* wrappers for the transformation. Only a few take into account the semantics of the non-ontological resource relationships. Also a few rely on additional resources for performing the transformation.

Re-engineering patterns embody expertise about how to guide a re-engineering process, they improve the efficiency of the re-engineering process, make the transformation process easier, and improve the reusability of non-ontological resources. Therefore, re-engineering patterns are a suitable technique for the transformation. However, none of the methods use re-engineering patterns as a technique for the transformation.

In conclusion, we can state that there is a clear need for some sort of re-engineering methods and tools that

- propose a formal specification of the transformation between entities of the resource and the ontology,
- contemplate the semantics of the non-ontological resource relationships,
- propose re-engineering patterns to guide the re-engineering process,
- support the re-engineering process activities by using the aforementioned re-engineering patterns,

2.5.3 Results According to the Ontology

Table 2.3 and table 2.6 summarize the methods and tools presented according to the characteristics of the resultant ontology: the ontology components, the ontology implementation language and whether one or more ontologies are generated.

Methods

- According to the ontology components. Seven methods generate classes, attributes and relations; five methods generate classes, attributes, relations and instances; two generate classes and relations; four generate only instances.
- According to the ontology implementation language. One method deals with RDF(S) and OWL-DLP ontologies; three methods deal OWL-DL ontologies; two deal with RDF instances; two deal with RDF(S) and OWL Full ontologies; one method deals with DAML+OIL ontologies; other method deals with LOOM/ACL ontologies; only one deals with SKOS RDF ontologies; three deal with RDF(S) ontologies; one deals with F-Logic/RDF ontologies, other deals with OWL Full/RDF ontologies; and one deals with CycL, OWL Full and WSML ontologies.
- According to whether the method generates one or several ontologies. There are two methods which generate several ontologies, the remaining sixteen methods generate one single ontology.

Tools

- According to the ontology components. One tool generates classes, attributes and relations; three tools generate classes, attributes, relations and instances; and eight tools generate only instances.
- According to the ontology implementation language. One tool deals with RDF(S) and OWL-DLP ontologies; other tool deals with F-Logic and RDF ontologies; other deals with OWL Full and RDF ontologies; one tool deals with ontologies implemented in OWL (Full, DL or Lite) and RDF ontologies; the remaining seven deal with RDF instances.
- According to whether the tool generates one or several ontologies. Two tools generate several ontologies, and ten tools generate one single ontology.

After having analyzed the characteristics related to the resultant ontology, we can conclude that there is a lack of re-engineering methods and tools which support several ontologies. Most of the presented research efforts only generate one single ontology.

Chapter 3

NeOn Method for Re-engineering Non-ontological Resources

In this chapter we provide a general overview of the NeOn pattern based approach for re-engineering non-ontological resources, i.e., classification schemes, and thesauri. We outline the pattern-based method for re-engineering non-ontological resources and present a template for describing the patterns.

3.1 NeOn Method for Re-engineering Non-ontological Resources

In this section we present our method for non-ontological resource re-engineering. We have opted for a pattern-based approach to carrying out the non-ontological re-engineering process. This method will be applied to classification schemes (chapter 4), and thesauri (chapter 5). Then, we present the proposed template used to describe the patterns for re-engineering them.

In summary, the NeOn method for re-engineering non-ontological resources into ontologies aims to:

- perform a conversion, as fully as possible, of knowledge included in the resource into ontologies,
- transform the resource into an ontology in one single step,
- take advantage of the data model underlying the non-ontological resource to guide the re-engineering process,
- employ re-engineering patterns to guide the transformation process; these re-engineering patterns do not deal with the resource implementation since they are focused in the upper levels, i.e., non-ontological resource types and non-ontological resource data models.
- generate two kinds of ontologies: taxonomies and lightweight ontologies.
  - a taxonomy [SFBG+07] is the way of organizing an ontology as a hierarchical structure of classes only related by subsumption relations.
  - a lightweight ontology [SFBG+07] adds the following features to the taxonomy structure: (a) a class can be related to other classes through an *ad-hoc* relation; (b) object and datatype properties can be defined and used to relate classes; and (c) a specific domain and range can be associated with defined object and datatype properties.
- offer the user the possibility to choose what kind of ontology (s)he wants to generate,
- generate ontologies at a conceptualization level, independent of the ontology implementation language,
- keep the provenance information of the resultant ontology.

Since folksonomies and free text are unstructured non-ontological resources, chapter 6 (folksonomies) and 7 (named entities) do not follow the pattern-based approach, they follow their own approach which is specific for their kind of non-ontological resource.

3.1.1 Re-engineering Patterns

In software engineering, re-engineering patterns [PS98] 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 and describe a process of re-engineering without proposing a complete methodology; the patterns can sometimes suggest a type of tool that one could use.

Re-engineering patterns for ontology are defined in [PGD$^+$08] as transformation rules applied to create a new ontology (target model) from elements of a source model that can be either an ontology or a non-ontological resource, e.g., a thesaurus concept, a data model pattern, a UML model, a linguistic structure, etc.

In the aforementioned work, re-engineering patterns are not integrated within a method to carry out the re-engineering process. Moreover, a template to describe re-engineering patterns in a unified way is not proposed. One of the goals of this deliverable is precisely to propose a method to carry out the re-engineering process of non-ontological resources into ontologies using re-engineering patterns. These patterns will generate the ontologies at a conceptualization level, independent of the ontology implementation language.

According to [PGD$^+$08], 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.

3.1.2 Patterns for Re-engineering Non-Ontological Resources

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

According to the non-ontological resource categorization presented in section 2.1, the data model can be different even for the same type of non-ontological resource. For every data model we can define a process 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 resultant 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 [SFBG$^+$07]. The current inventory of *NeOn Ontology Modelling 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. We decided to model the resultant ontologies following these recommendations: taxonomy and lightweight ontology. 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. We plan to include the algorithms and implementations later on in a framework which will implement the transformation process. Also we will include a section to generate ontologies following the Linking Open Data[1] recommendations.

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

Next, we present the proposed template used to describe the patterns for re-engineering non-ontological resources (PR-NOR). To present the patterns for re-engineering non-ontological resources we have adapted the tabular template for ontology design patterns used in [SFBG+07]. The template adapted and the meaning of each field is shown in Table 3.1.

Table 3.1: Pattern for Re-engineering Non-Ontological Resource Template

Slot	Value
General Information	
Name	Name of the pattern
Identifier	An acronym composed of component type + abbreviated name of the component + number
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 non-ontological resources.
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 non-ontological resource.
Example	Description in natural language of an example of the non-ontological resource.
Graphical Representation	
General	Graphical representation of the non-ontological resource.
Example	Graphical representation of the example of non-ontological resource.
OUTPUT: Designed Ontology	
General	Description in natural language of the ontology created after applying the pattern for re-engineering the non-ontological resource.
Graphical Representation	
(UML) General Solution Ontology	Graphical representation, using the UML profile [BH06], of the ontology created for the non-ontological resource being re-engineered.
(UML) Example Solution Ontology	Example showing a graphical representation, using the UML profile [BH06], of the ontology created for the non-ontological resource 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 non-ontological resource example, using the above sequence of activities.
Relationships (Optional)	
Relations to other modelling components	Description of any relation to other PR-NOR patterns or other ontology design patterns.

3.1.3 General Model for Non-Ontological Resource Re-engineering

In a nutshell, our method to non-ontological resource re-engineering considers as input a pool of non-ontological resources and patterns for re-engineering non-ontological resources. The latter provide solutions to the problem of transforming non-ontological resources into ontologies.

Based on the software re-engineering model presented in D.5.4.1 [SFdCB+08] we propose our re-engineering model for non-ontological resource re-engineering in Fig.3.1.

The NOR re-engineering process consists of the following activities, which are defined in a Glossary of Activities in the Ontology Engineering[SFGP08]:

1. *Non-Ontological Resource Reverse Engineering*, whose goal is to analyze a NOR to identify its un-

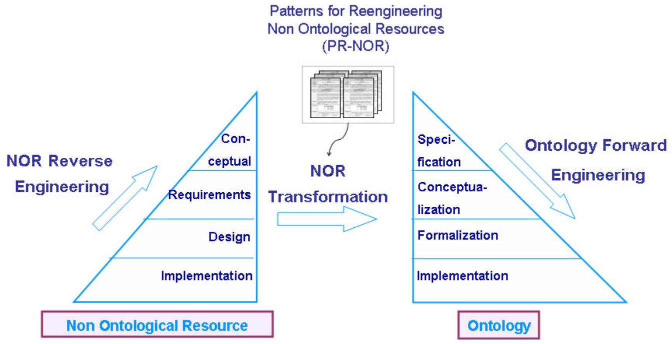


Figure 3.1: Re-engineering Model for Non-Ontological Resources

derlying 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 spreadsheet among others, we can consider them as software resources, and therefore, we use the software abstraction levels (implementation, design, requirements and conceptual) shown in Fig. 3.1 within this activity. Here the requirements and the essential design, structure and content of the NOR must be recaptured.

2. *Non-Ontological Resource Transformation*, whose goal is to generate a conceptual model from the NOR. We propose the use of Patterns for Re-engineering Non-Ontological Resources (PR-NOR) 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.

3. *Ontology Forward Engineering*, whose goal is to output a new implementation of the ontology on the basis of the new conceptual model. We use the ontology levels of abstraction to depict this activity because they are directly related to the ontology development process.

3.1.4 Non-ontological Resources Re-engineering Process

The non-ontological resource re-engineering process consists of the activities depicted in Fig.3.2. This process is based on the one described in D.5.4.1 [SFdCB$^+$08]. In general, we follow the same process, but use the set of patterns for re-engineering non-ontological resources described in this deliverable.

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.

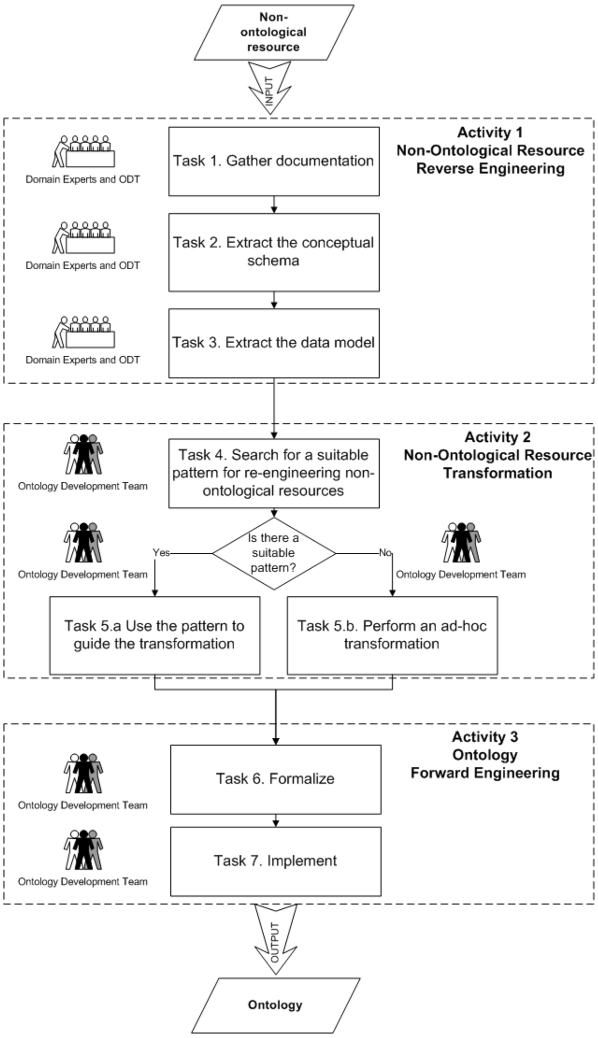

Non-ontological resource
INPUT
Activity 1
Non-Ontological Resource
Reverse Engineering
Domain Experts and ODT
Task 1. Gather documentation
Domain Experts and ODT
Task 2. Extract the conceptual schema
Domain Experts and ODT
Task 3. Extract the data model
Activity 2
Non-Ontological Resource
Transformation
Ontology Development Team
Task 4. Search for a suitable pattern for re-engineering non-ontological resources
Ontology Development Team
Is there a suitable pattern?
Yes
No
Ontology Development Team
Task 5.a Use the pattern to guide the transformation
Task 5.b. Perform an ad-hoc transformation
Activity 3
Ontology
Forward Engineering
Ontology Development Team
Task 6. Formalize
Ontology Development Team
Task 7. Implement
OUTPUT
Ontology

Figure 3.2: Re-engineering process for Non-Ontological Resources

- **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. We 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 re-engineering non-ontological resource the NeOn library of patterns[2] 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

[2] `http://www.ontologydesignpatterns.org`

subClassOf and *ad-hoc* relations. Finally, 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.

- **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.

Chapter 4

Methods for Re-engineering Classification Schemes

4.1 Introduction

Classification schemes [KBH$^+$97] have a role in aiding information retrieval in a network environment, specially for providing browsing structures for subject-based information gateways on the Web. Advantages of using classification schemes include improved subject browsing facilities, and improved interoperability with other services. Classification schemes are likely the most valuable input for creating, at reasonable cost, ontologies in many domains. They contain, readily available, a wealth of category definitions plus a hierarchy, and they reflect some degree of community consensus [HdB07]. In this chapter we present a definition of classification schemes, the data models for representing classification schemes and the method for re-engineering classification schemes into ontologies.

4.2 Classification Scheme

A classification scheme [ISO04] is the descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common. For example, the Fishery International Standard Statistical Classification of Aquatic Animals and Plants (ISSCAAP)[1].
Based on [ISO04] we identify the following classification scheme components in Figure 4.1:

- *cs_name*, which is the name of the classification scheme.
- *Classification scheme item*, which represents the individual item within a classification scheme. It has the following elements:
  - *csi_name*, which is the name of the classification scheme item.
  - One or more *csi_attributes*
- *Classification scheme item relationship*, which is the relationship among items within a classification scheme. Such relation serves to assist navigation through a large number of classification scheme items. It has the *csir_name* element, which is the name of the classification scheme item relationship.

[1] http://www.fao.org/figis/servlet/RefServlet

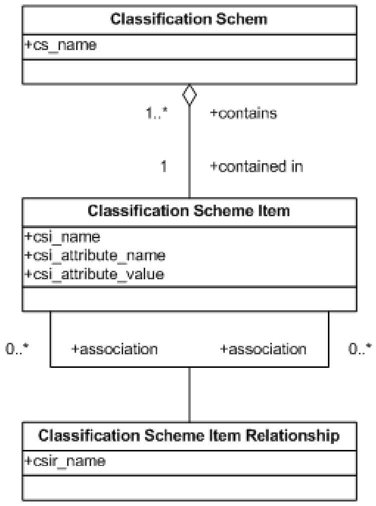


Figure 4.1: UML representation of the classification scheme main components [ISO04]

4.2.1 Classification Scheme Data Models

As we mentioned in section 2.1 there are different ways of representing the knowledge encoded by a particular resource. In this section we present the existing data models for classification schemes. In order to exemplify the data models for classification schemes, we use an excerpt from the FAO classification scheme of water areas[2] presented in Figure 4.2.

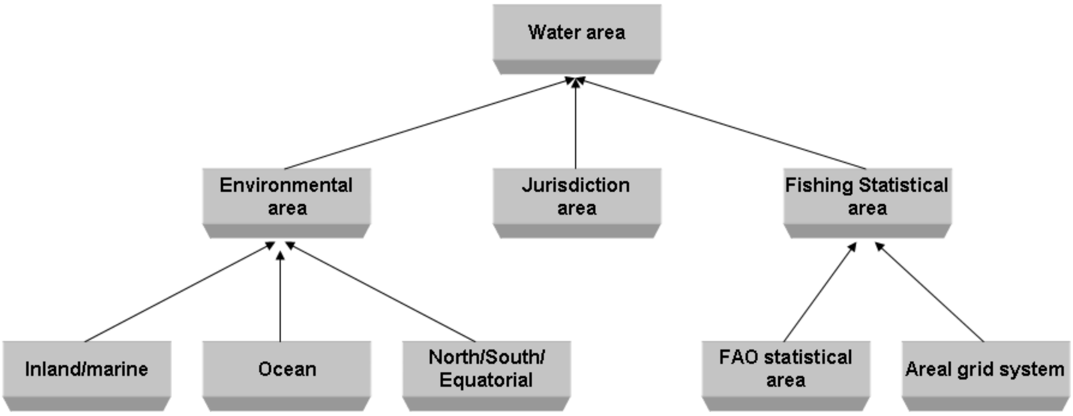
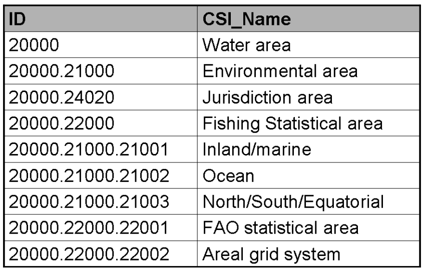


Figure 4.2: Excerpt of the Water Area classification scheme.

Path Enumeration Data Model

A path enumeration data model [Bra05] 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. In other words, every classification scheme item has unique code (i.e. a key value), distinguishing it from the others. Also, every classification scheme item has a path, consisting of the concatenated unique string codes of all the parents, until the root. Figure 4.3 illustrates this data model.

[2] http://www.fao.org/figis/servlet/RefServlet

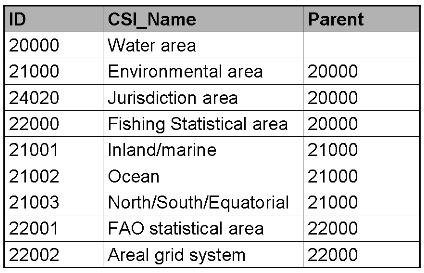


ID	CSI_Name
20000	Water area
20000.21000	Environmental area
20000.24020	Jurisdiction area
20000.22000	Fishing Statistical area
20000.21000.21001	Inland/marine
20000.21000.21002	Ocean
20000.21000.21003	North/South/Equatorial
20000.22000.22001	FAO statistical area
20000.22000.22002	Areal grid system

Figure 4.3: Path enumeration data model.

Adjacency List Data Model

An adjacency list [Bra05] data model is a recursive structure for hierarchy representations comprising a list of nodes with a linking column to their parent nodes. In this case, every classification scheme item has the parent code. Figure 4.4 shows this data model.

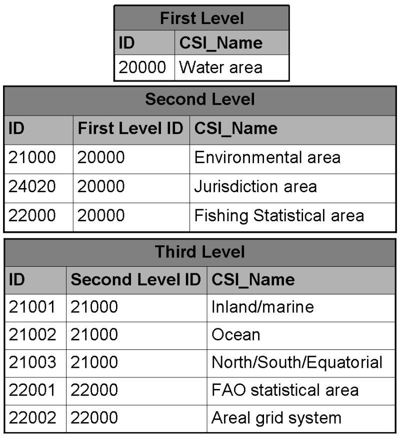


ID	CSI_Name	Parent
20000	Water area	
21000	Environmental area	20000
24020	Jurisdiction area	20000
22000	Fishing Statistical area	20000
21001	Inland/marine	21000
21002	Ocean	21000
21003	North/South/Equatorial	21000
22001	FAO statistical area	22000
22002	Areal grid system	22000

Figure 4.4: Adjacency list data model.

Snowflake Data Model

A snowflake data model [MZ06] is a normalized structure for hierarchy representations. In this case, the classification scheme items are grouped by levels or entities. There are as many groups as levels the classification scheme has. In this model every classification scheme item has the parent code (i.e. parent key value), just like the adjacency list data model. However, the difference is that in the snowflake data model the classification scheme items are grouped by levels or entities, and therefore hierarchy levels must be known in advance. Figure 4.5 illustrates this model. Optionally, a relation between the groups, or entities, can be exist.



First Level	
ID	CSI_Name
20000	Water area

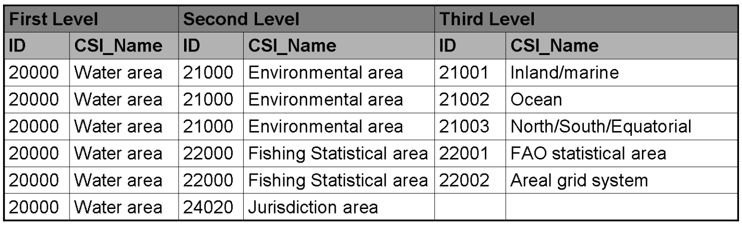
Second Level		
ID	First Level ID	CSI_Name
21000	20000	Environmental area
24020	20000	Jurisdiction area
22000	20000	Fishing Statistical area

Third Level		
ID	Second Level ID	CSI_Name
21001	21000	Inland/marine
21002	21000	Ocean
21003	21000	North/South/Equatorial
22001	22000	FAO statistical area
22002	22000	Areal grid system

Figure 4.5: Adjacency list data model.

Flattened Data Model

A flattened data model [MZ06] is a denormalized structure for hierarchy representations. In this case, each hierarchy level is represented on a different column. There are as many columns as levels the classification scheme has. The hierarchy is represented using one single entity where each hierarchy level is stored on a different column. In this case, hierarchy levels must be known in advance. Figure 4.6 depicts this model.

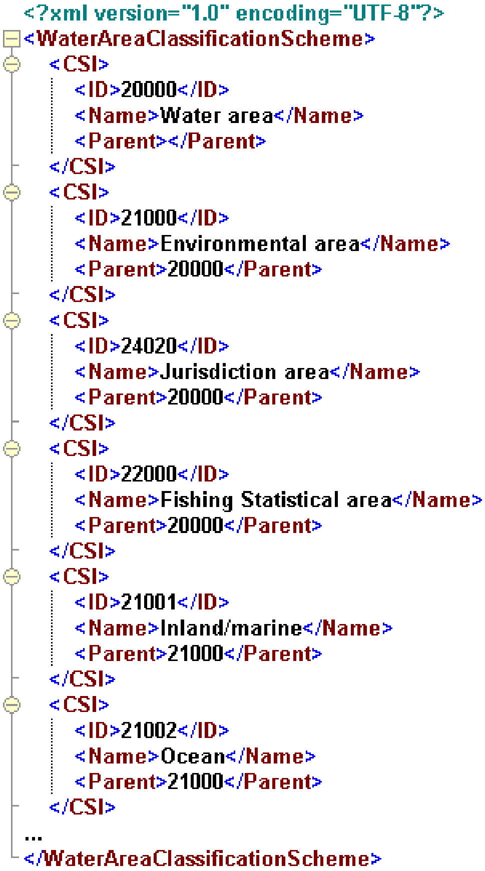


First Level		Second Level		Third Level	
ID	CSI_Name	ID	CSI_Name	ID	CSI_Name
20000	Water area	21000	Environmental area	21001	Inland/marine
20000	Water area	21000	Environmental area	21002	Ocean
20000	Water area	21000	Environmental area	21003	North/South/Equatorial
20000	Water area	22000	Fishing Statistical area	22001	FAO statistical area
20000	Water area	22000	Fishing Statistical area	22002	Areal grid system
20000	Water area	24020	Jurisdiction area		

Figure 4.6: Flattened data model.

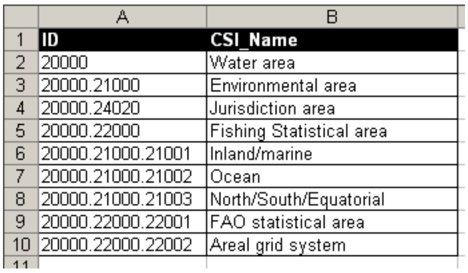
4.2.2 Classification Scheme Implementations

Finally these data models can be implemented as any of the identified types on section 2.1, i.e. databases, XML files, flat files, and spreadsheets. A direct implementation would be as tables in a relational database or in a spreadsheet. Figure 4.2.2 presents an XML implementation of the adjacency list model of the water area classification, and Figure 4.2.2 presents a spreadsheet implementation of the path enumeration model of the same classification scheme.



```
<?xml version="1.0" encoding="UTF-8"?>
<WaterAreaClassificationScheme>
  <CSI>
    <ID>20000</ID>
    <Name>Water area</Name>
    <Parent></Parent>
  </CSI>
  <CSI>
    <ID>21000</ID>
    <Name>Environmental area</Name>
    <Parent>20000</Parent>
  </CSI>
  <CSI>
    <ID>24020</ID>
    <Name>Jurisdiction area</Name>
    <Parent>20000</Parent>
  </CSI>
  <CSI>
    <ID>22000</ID>
    <Name>Fishing Statistical area</Name>
    <Parent>20000</Parent>
  </CSI>
  <CSI>
    <ID>21001</ID>
    <Name>Inland/marine</Name>
    <Parent>21000</Parent>
  </CSI>
  <CSI>
    <ID>21002</ID>
    <Name>Ocean</Name>
    <Parent>21000</Parent>
  </CSI>
...
</WaterAreaClassificationScheme>
```

Figure 4.7: Water Area Classification Scheme XML Implementation for the Adjacency List Data Model.



	A	B
1	ID	CSI_Name
2	20000	Water area
3	20000.21000	Environmental area
4	20000.24020	Jurisdiction area
5	20000.22000	Fishing Statistical area
6	20000.21000.21001	Inland/marine
7	20000.21000.21002	Ocean
8	20000.21000.21003	North/South/Equatorial
9	20000.22000.22001	FAO statistical area
10	20000.22000.22002	Areal grid system
11		

Figure 4.8: Water Area Classification Scheme Spreadsheet Implementation for the Path Enumeration Data Model.

In a nutshell, Figure 4.9 shows how a given type of Classification Scheme 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, Figure 4.9 shows a classification scheme modeled following a path enumeration model. In this case, the classification scheme is implemented in a database and in an XML file.

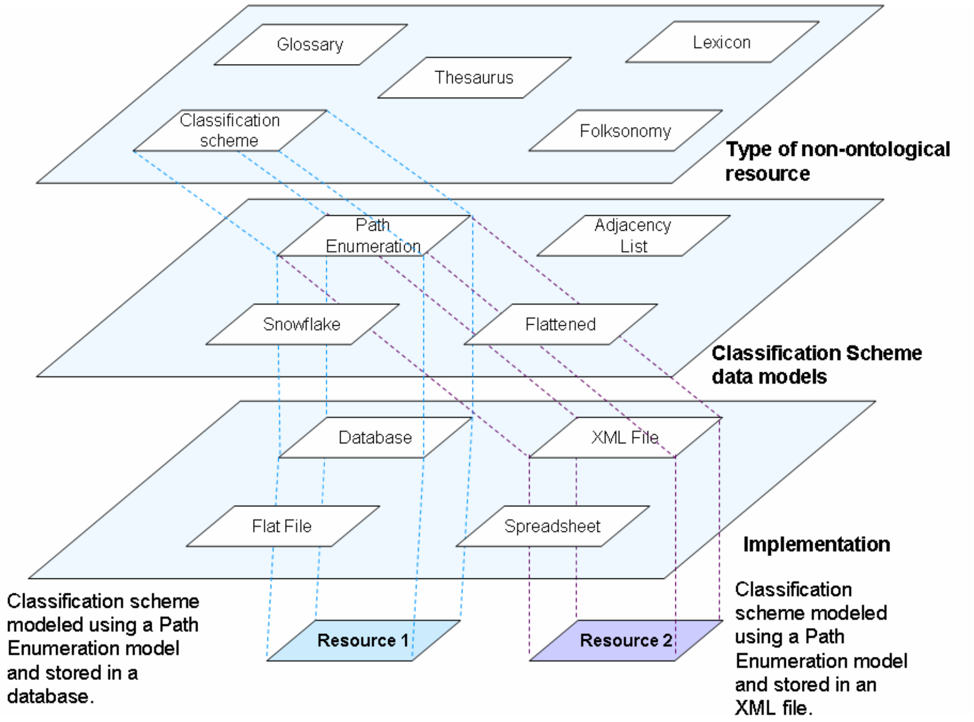

Glossary
Lexicon
Thesaurus
Classification scheme
Folksonomy
Type of non-ontological resource
Path Enumeration
Adjacency List
Snowflake
Flattened
Classification Scheme data models
Database
XML File
Flat File
Spreadsheet
Implementation
Classification scheme modeled using a Path Enumeration model and stored in a database.
Resource 1
Resource 2
Classification scheme modeled using a Path Enumeration model and stored in an XML file.

Figure 4.9: Classification Scheme Categorization

4.3 Patterns for Re-engineering Classification Schemes into Ontologies

In this section we present the re-engineering patterns (PR-NOR) for re-engineering classification schemes into ontologies. These patterns come from the experience of ontology engineers in developing ontologies using classification schemes in several projects (SEEMP[3], NeOn[4], and Knowledge Web[5]). The patterns are:

- PR-NOR-CLTX-01. Pattern for re-engineering a classification scheme which follows the path enumeration data model, into a taxonomy. In that case, the semantics of the relations between classification

[3] http://www.seemp.org
[4] http://www.neon-project.org
[5] http://knowledgeweb.semanticweb.org

scheme items are *subClassOf*.

- PR-NOR-CLTX-02. Pattern for re-engineering a classification scheme which follows the adjacency list data model, into a taxonomy. In that case, the semantics of the relations between classification scheme items are *subClassOf*.
- PR-NOR-CLTX-03. Pattern for re-engineering a classification scheme which follows the snowflake data model, into a taxonomy. In that case, the semantics of the relations between classification scheme items are *subClassOf*.
- PR-NOR-CLTX-04. Pattern for re-engineering a classification scheme which follows the flattened data model, into a taxonomy. In that case, the semantics of the relations between classification scheme items are *subClassOf*.
- PR-NOR-CLLO-03. Pattern for re-engineering a classification scheme which follows the snowflake data model, into a lightweight ontology. In that case, which the semantics of the relations between classification scheme items are *subClassOf*, and *ad-hoc* relations.
- PR-NOR-CLLO-04. Pattern for re-engineering a classification scheme which follows the flattened data model, into a lightweight ontology. In that case, which the semantics of the relations between classification scheme items are *subClassOf*, and *ad-hoc* relations.

4.3.1 Patterns for Re-engineering Classification Schemes into Taxonomies

In this section we present four re-engineering patterns for re-engineering classification schemes into taxonomies. Since a taxonomy [SFBG$^+$07] is a hierarchical structure of classes only related by subsumption relations, these patterns follow the transformation approach, mentioned in section 2.2.2, for transforming resource content into an ontology schema, because it is the suitable for dealing with the resource data models and the target taxonomies.

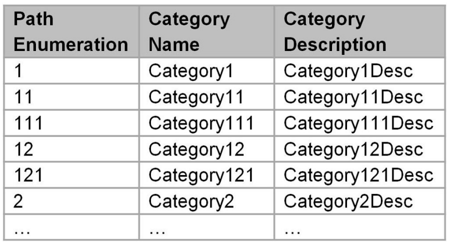
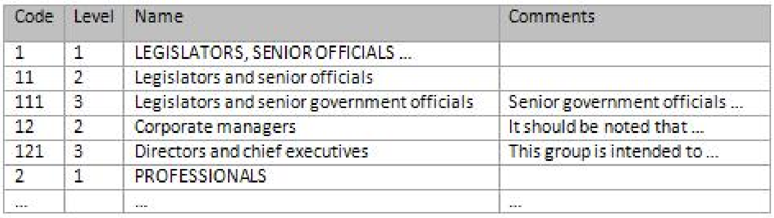
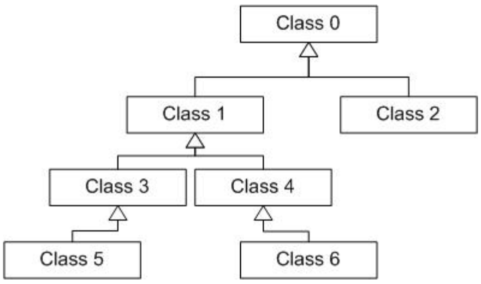
PR-NOR-CLTX-01. Pattern for re-engineering a classification scheme which follows the path enumeration data model

The pattern for re-engineering non-ontological resource shown in Table 4.1 suggests a guide to transform a classification scheme into an ontology. The classification scheme is modeled/represented with a path enumeration data model. This pattern aims at creating a taxonomy from the classification scheme, being the semantics of the relations between classification scheme items the *subClassOf* relationship.

Table 4.1: Pattern for Re-engineering a Classification Scheme which follows the path enumeration data model

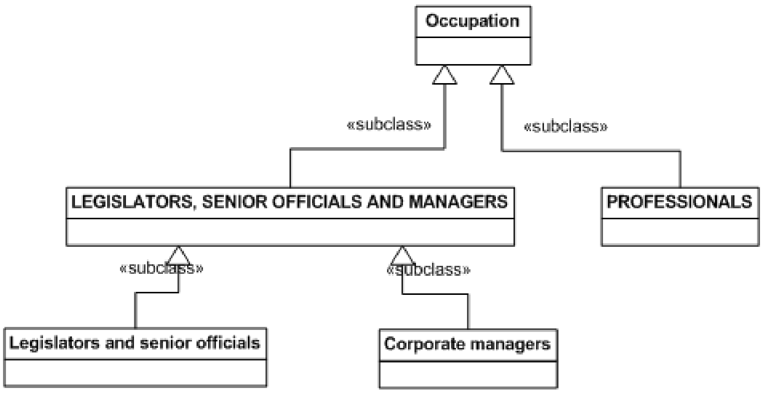
Slot	Value
General Information	
Name	Classification scheme to Taxonomy (path enumeration model)
Identifier	PR-NOR-CLTX-01
Type of Component	Pattern for Re-engineering Non-Ontological Resource (PR-NOR)
Use Case	
General	Re-engineering a classification scheme which follows the path enumeration model to design a taxonomy.
Example	Suppose that someone wants to build an ontology based on the International Standard Classification of Occupations (for European Union purposes) ISCO-88 (COM). This classification scheme follows the path enumeration data model.
Pattern for Re-engineering Non-Ontological Resource	
INPUT: Resource to be Re-engineered	

Table 4.1: Pattern for Re-engineering a Classification Scheme which follows the path enumeration data model(continued)

<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a classification scheme which follows the path enumeration model.
A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parents and children concepts may vary depending of the context.
The path enumeration data model [Bra05] for classification schemes take advantage of that there is one and only one path from the root to every item in the classification. The path enumeration model stores that path as string by concatenating either the edges or the keys of the classification scheme items in the path.</td></tr>
<tr><td><b>Example</b></td><td>The International Standard Classification of Occupations (for European Union purposes), 1988 version: ISCO-88 (COM) published by Eurostat[6] is modeled with the path enumeration data model.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>General</b></td><td>
<table>
<tr><th>Path Enumeration</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>Category1</td><td>Category1Desc</td></tr>
<tr><td>11</td><td>Category11</td><td>Category11Desc</td></tr>
<tr><td>111</td><td>Category111</td><td>Category111Desc</td></tr>
<tr><td>12</td><td>Category12</td><td>Category12Desc</td></tr>
<tr><td>121</td><td>Category121</td><td>Category121Desc</td></tr>
<tr><td>2</td><td>Category2</td><td>Category2Desc</td></tr>
<tr><td>…</td><td>…</td><td>…</td></tr>
</table></td></tr>
<tr><td><b>Example</b></td><td>
<table>
<tr><th>Code</th><th>Level</th><th>Name</th><th>Comments</th></tr>
<tr><td>1</td><td>1</td><td>LEGISLATORS, SENIOR OFFICIALS ...</td><td></td></tr>
<tr><td>11</td><td>2</td><td>Legislators and senior officials</td><td></td></tr>
<tr><td>111</td><td>3</td><td>Legislators and senior government officials</td><td>Senior government officials ...</td></tr>
<tr><td>12</td><td>2</td><td>Corporate managers</td><td>It should be noted that ...</td></tr>
<tr><td>121</td><td>3</td><td>Directors and chief executives</td><td>This group is intended to ...</td></tr>
<tr><td>2</td><td>1</td><td>PROFESSIONALS</td><td></td></tr>
<tr><td>...</td><td></td><td>...</td><td>...</td></tr>
</table></td></tr>
<tr><td colspan="2"><b>OUTPUT: Designed Ontology</b></td></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the taxonomy architectural pattern (AP-TX-01) [SFBG+07]. Each category in the classification scheme is mapped to a class, and the semantics of the relationship between children and parent categories are mapped to <i>subClassOf</i> relations.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6

</td></tr>
</table>

Table 4.1: Pattern for Re-engineering a Classification Scheme which follows the path enumeration data model(continued)

Slot	Value
(UML) Example Solution Ontology	 Occupation «subclass» «subclass» LEGISLATORS, SENIOR OFFICIALS AND MANAGERS PROFESSIONALS «subclass» «subclass» Legislators and senior officials Corporate managers
PROCESS: How to Re-engineer	
General	1. Identify the classification scheme items whose their path enumeration values are equal to their key values, i.e. classification scheme items without parents. Formally, every $ce_i \in CE, v_{path} = v_k$. 2. For each one of the above identified classification scheme items ce_i: 2.1. Create the corresponding ontology class, C_i class. 2.2. Identify the classification scheme items, ce_j, which are children of ce_i, by using the path enumeration values. 2.3. For each one of the above identified classification scheme items ce_j: 2.3.1. Create the corresponding ontology class, C_j class. 2.3.2. Set up the *subClassOf* relation between C_j and C_i. 2.3.3. Repeat from step 2.2 for ce_j as a new ce_i. 3. If there are more than one classification scheme items without parent ce_i 3.1. Create an *ad-hoc* class as the root class of the ontology. 3.2. Set up the *subClassOf* relation between C_i class and the root class.
Example	1. Create the `LEGISLATORS, SENIOR OFFICIALS AND MANAGERS` class. 2. Create the `Legislators and senior officials` class, and set up the *subClassOf* relation between the `Legislators and senior officials` class and the `LEGISLATORS, SENIOR OFFICIALS AND MANAGERS` class. 3. Create the `Corporate managers` class, and set up the *subClassOf* relation between the `Corporate managers` class and the `LEGISLATORS, SENIOR OFFICIALS AND MANAGERS` class. 4. Create the `PROFESSIONALS` class. 5. Create the `Occupation` class. 6. Set up the *subClassOf* relation between the `LEGISLATORS, SENIOR OFFICIALS AND MANAGERS` class and the `Occupation` class. 7. Set up the *subClassOf* relation between the `PROFESSIONALS` class and the `Occupation` class.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: TX-AP-01 [SFBG$^+$07]

PR-NOR-CLTX-02. Pattern for re-engineering a classification scheme which follows the adjacency list data model

The pattern for re-engineering non-ontological resource shown in Table 4.2 suggests a guide to transform a classification scheme into an ontology. The classification scheme is modeled/represented with an adjacency list data model. This pattern aims at creating a taxonomy from the classification scheme, being the semantics of the relations between classification scheme items the *subClassOf* relationship.

Table 4.2: Pattern for Re-engineering a Classification Scheme which follows the adjacency list data model

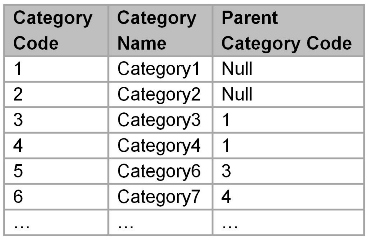
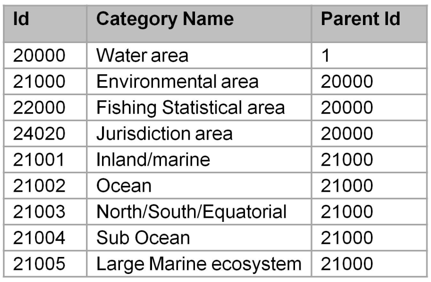
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td colspan="2"><b>General Information</b></td></tr>
<tr><td><b>Name</b></td><td>Classification scheme to Taxonomy (adjacency list model)</td></tr>
<tr><td><b>Identifier</b></td><td>PR-NOR-CLTX-02</td></tr>
<tr><td><b>Type of Component</b></td><td>Pattern for Re-engineering Non-Ontological Resource (PR-NOR)</td></tr>
<tr><td colspan="2"><b>Use Case</b></td></tr>
<tr><td><b>General</b></td><td>Re-engineering a classification scheme which follows the adjacency list model to design a taxonomy.</td></tr>
<tr><td><b>Example</b></td><td>Suppose that someone wants to build an ontology based on the water areas classification published by FAO. This classification scheme follows the adjacency list data model.</td></tr>
<tr><td colspan="2"><b>Pattern for Re-engineering Non-Ontological Resource</b></td></tr>
<tr><td colspan="2"><b>INPUT: Resource to be Re-engineered</b></td></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a classification scheme which follows the adjacency list model.
A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity.
The semantics of the hierarchical relation between parents and children concepts may vary depending of the context.
The adjacency list data model [Bra05] for hierarchical classifications proposes to create an entity which holds a list of items with a linking column associated to their parent items.</td></tr>
<tr><td><b>Example</b></td><td>The FAO classification for water areas groups them according to some different criteria as environment, statistics, and jurisdiction, among others.
This classification scheme is available at
<code>http://www.fao.org/figis/servlet/RefServlet</code></td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>General</b></td><td>
<table>
<tr><th>Category Code</th><th>Category Name</th><th>Parent Category Code</th></tr>
<tr><td>1</td><td>Category1</td><td>Null</td></tr>
<tr><td>2</td><td>Category2</td><td>Null</td></tr>
<tr><td>3</td><td>Category3</td><td>1</td></tr>
<tr><td>4</td><td>Category4</td><td>1</td></tr>
<tr><td>5</td><td>Category6</td><td>3</td></tr>
<tr><td>6</td><td>Category7</td><td>4</td></tr>
<tr><td>…</td><td>…</td><td>…</td></tr>
</table></td></tr>
<tr><td><b>Example</b></td><td>
<table>
<tr><th>Id</th><th>Category Name</th><th>Parent Id</th></tr>
<tr><td>20000</td><td>Water area</td><td>1</td></tr>
<tr><td>21000</td><td>Environmental area</td><td>20000</td></tr>
<tr><td>22000</td><td>Fishing Statistical area</td><td>20000</td></tr>
<tr><td>24020</td><td>Jurisdiction area</td><td>20000</td></tr>
<tr><td>21001</td><td>Inland/marine</td><td>21000</td></tr>
<tr><td>21002</td><td>Ocean</td><td>21000</td></tr>
<tr><td>21003</td><td>North/South/Equatorial</td><td>21000</td></tr>
<tr><td>21004</td><td>Sub Ocean</td><td>21000</td></tr>
<tr><td>21005</td><td>Large Marine ecosystem</td><td>21000</td></tr>
</table></td></tr>
<tr><td colspan="2"><b>OUTPUT: Designed Ontology</b></td></tr>
</table>

Table 4.2: Pattern for Re-engineering a Classification Scheme which follows the adjacency list data model(continued)

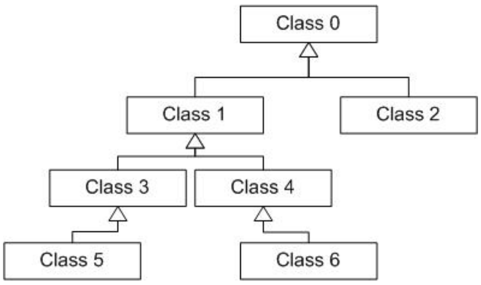
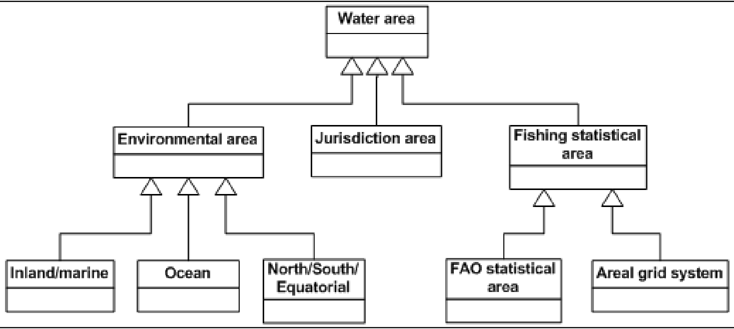
Slot	Value
General	The generated ontology will be based on the taxonomy architectural pattern (AP-TX-01) [SFBG$^+$07]. Each category in the classification scheme is mapped to a class, and the semantics of the relationship between children and parent categories are mapped to *subClassOf* relations.
Graphical Representation	
(UML) General Solution Ontology	 Class 0 Class 1 Class 2 Class 3 Class 4 Class 5 Class 6
(UML) Example Solution Ontology	 Water area Environmental area Jurisdiction area Fishing statistical area Inland/marine Ocean North/South/ Equatorial FAO statistical area Areal grid system
PROCESS: How to Re-engineer	
General	1. Identify the classification scheme items which do not have a parent key value, i.e. classification scheme items without parents. Formally, every $ce_i \in CE$, which $ca_{parentID} =$ NULL. 2. For each one of the above identified classification scheme items ce_i: 2.1. Create the corresponding ontology class, C_i class. 2.2. Identify the classification scheme items, ce_j, which are children of ce_i, by using the parent key values. 2.3. For each one of the above identified classification scheme items ce_j: 2.3.1. Create the corresponding ontology class, C_j class. 2.3.2. Set up the *subClassOf* relation between C_j and C_i. 2.3.3. Repeat from step 2.2 for ce_j as a new ce_i. 3. If there are more than one classification scheme items without parent ce_i 3.1. Create an *ad-hoc* class as the root class of the ontology. 3.2. Set up the *subClassOf* relation between C_i class and the root class.

Table 4.2: Pattern for Re-engineering a Classification Scheme which follows the adjacency list data model(continued)

Slot	Value
Example	1. Create the `Water area` class. 2. Create the `Fishing Statistical area` class, and set up the *subClassOf* relation between the `Fishing Statistical area` class and the `Water area` class. 3. Create the `Environmental area` class, and set up the *subClassOf* relation between the `Environmental area` class and the `Water area` class. 3.1. Create the `Inland/marine` class, and set up the *subClassOf* relation between the `Inland/marine` class and the `Environmental area` class. 3.2. Create the `Ocean` class, and set up the *subClassOf* relation between the `Ocean` class and the `Environmental area` class. 3.3. Create the `North/South/Equatorial` class, and set up the *subClassOf* relation between the `North a South a Equatorial` class and the `Environmental area` class. 3.4. Create the `Sub Ocean` class, and set up the *subClassOf* relation between the `Sub Ocean` class and the `Environmental area` class. 3.5. Create the `Large Marine ecosystem` class, and set up the *subClassOf* relation between the `Large Marine ecosystem` class and the `Environmental area` class. 4. Create the `Jurisdiction area` class, and set up the *subClassOf* relation between the `Jurisdiction area` class and the `Water area` class.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: TX-AP-01 [SFBG^{+}07]

PR-NOR-CLTX-03. Pattern for re-engineering a classification scheme which follows the snowflake data model

The pattern for re-engineering non-ontological resource shown in Table 4.3 suggests a guide to transform a classification scheme into an ontology. The classification scheme is modeled/represented with a snowflake data model. This pattern aims at creating a taxonomy from the classification scheme, being the semantics of the relations between classification scheme items the *subClassOf* relationship.

Table 4.3: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model.

Slot	Value
General Information	
Name	Classification scheme to Taxonomy (snowflake model)
Identifier	PR-NOR-CLTX-03
Type of Component	Pattern for Re-engineering Non-Ontological Resource (PR-NOR)
Use Case	
General	Re-engineering a classification scheme which follows the snowflake model to design a taxonomy.
Example	Suppose that someone wants to build an ontology based on an occupation hierarchical classification, which follows the snowflake data model.
Pattern for Re-engineering Non-Ontological Resource	
INPUT: Resource to be Re-engineered	

Table 4.3: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model(continued)

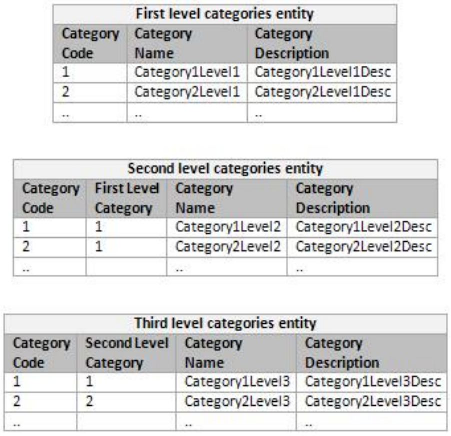
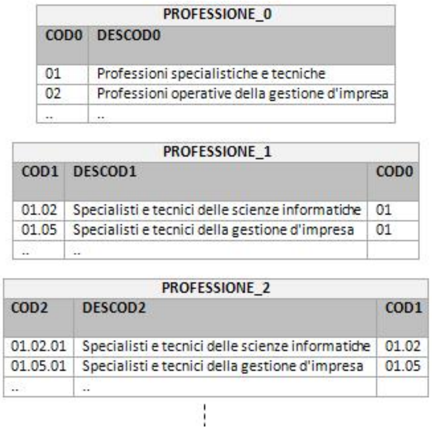
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a classification scheme which follows the snowflake model.
A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parents and children concepts may vary depending of the context.
The snowflake data model [MZ06] is a normalized structure for hierarchy representations. In this case, the classification scheme items are grouped by levels or entities. There are as many groups as levels the classification scheme has.</td></tr>
<tr><td><b>Example</b></td><td>Snowflakes models are widely used on data warehouses to build hierarchical classifications on structures known as dimensions. Some examples of dimension are Time, Product Category, Geography, Occupations, etc.
In this pattern the example is a occupation hierarchical classification hold on four different tables, one for each level (PROFESSIONI_0, PROFESSIONI_1, PROFESSIONI_2, PROFESSIONI_3).</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>General</b></td><td>

<table>
<tr><th colspan="3">First level categories entity</th></tr>
<tr><th>Category Code</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>Category1Level1</td><td>Category1Level1Desc</td></tr>
<tr><td>2</td><td>Category2Level1</td><td>Category2Level1Desc</td></tr>
<tr><td>..</td><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="4">Second level categories entity</th></tr>
<tr><th>Category Code</th><th>First Level Category</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>1</td><td>Category1Level2</td><td>Category1Level2Desc</td></tr>
<tr><td>2</td><td>1</td><td>Category2Level2</td><td>Category2Level2Desc</td></tr>
<tr><td>..</td><td></td><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="4">Third level categories entity</th></tr>
<tr><th>Category Code</th><th>Second Level Category</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>1</td><td>Category1Level3</td><td>Category1Level3Desc</td></tr>
<tr><td>2</td><td>2</td><td>Category2Level3</td><td>Category2Level3Desc</td></tr>
<tr><td>..</td><td></td><td>..</td><td>..</td></tr>
</table>
</td></tr>
<tr><td><b>Example</b></td><td>

<table>
<tr><th colspan="2">PROFESSIONE_0</th></tr>
<tr><th>COD0</th><th>DESCOD0</th></tr>
<tr><td>01</td><td>Professioni specialistiche e tecniche</td></tr>
<tr><td>02</td><td>Professioni operative della gestione d'impresa</td></tr>
<tr><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="3">PROFESSIONE_1</th></tr>
<tr><th>COD1</th><th>DESCOD1</th><th>COD0</th></tr>
<tr><td>01.02</td><td>Specialisti e tecnici delle scienze informatiche</td><td>01</td></tr>
<tr><td>01.05</td><td>Specialisti e tecnici della gestione d'impresa</td><td>01</td></tr>
<tr><td>..</td><td>..</td><td></td></tr>
</table>
<table>
<tr><th colspan="3">PROFESSIONE_2</th></tr>
<tr><th>COD2</th><th>DESCOD2</th><th>COD1</th></tr>
<tr><td>01.02.01</td><td>Specialisti e tecnici delle scienze informatiche</td><td>01.02</td></tr>
<tr><td>01.05.01</td><td>Specialisti e tecnici della gestione d'impresa</td><td>01.05</td></tr>
<tr><td>..</td><td>..</td><td></td></tr>
</table>
</td></tr>
<tr><th colspan="2">OUTPUT: Designed Ontology</th></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the taxonomy architectural pattern (AP-TX-01) [SFBG$^+$07]. Each category in the classification scheme is mapped to a class, and the semantics of the relationship between children and parent categories are mapped to <i>subClassOf</i> relations.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
</table>

Table 4.3: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model(continued)

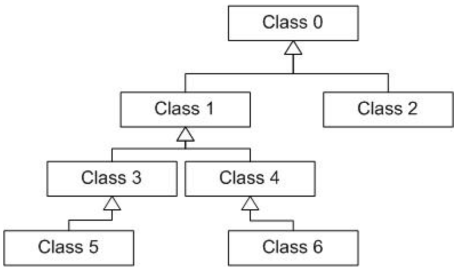
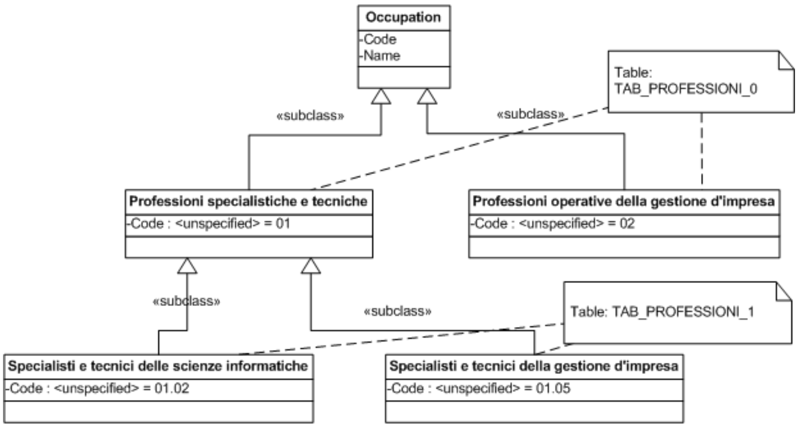
Slot	Value
(UML) General Solution Ontology	 Class 0 Class 1 Class 2 Class 3 Class 4 Class 5 Class 6
(UML) Example Solution Ontology	 Occupation -Code -Name Table: TAB_PROFESSIONI_0 «subclass» «subclass» Professioni specialistiche e tecniche -Code : <unspecified> = 01 Professioni operative della gestione d'impresa -Code : <unspecified> = 02 «subclass» «subclass» Table: TAB_PROFESSIONI_1 Specialisti e tecnici delle scienze informatiche -Code : <unspecified> = 01.02 Specialisti e tecnici della gestione d'impresa -Code : <unspecified> = 01.05
PROCESS: How to Re-engineer	
General	1. Select all the classification scheme items from the first level. 2. For each one of the above selected classification scheme items ce_i: 2.1. Create the corresponding ontology class, C_i class. 2.2. Identify the classification scheme items, ce_j, on the next level, which are children of ce_i, by using the parent key values. 2.3. For each one of the above identified classification scheme items ce_j: 2.3.1. Create the corresponding ontology class, C_j class. 2.3.2. Set up the *subClassOf* relation between C_j and C_i. 2.3.3. Repeat from step 2.2 for ce_j as a new ce_i. 3. If there are more than one classification scheme items from the first level ce_i 3.1. Create an *ad-hoc* class as the root class of the ontology. 3.2. Set up the *subClassOf* relation between C_i class and the root class.
Example	1. Create the `Professioni specialistiche e tecniche` class. 1.1. Create the `Specialist e tecnici delle scienze informatiche` class and set up the *subClassOf* relation between the `Specialist e tecnici delle scienze informatiche` class and the `Professioni specialistiche e tecniche` class. 1.2. Create the `Specialist e tecnici delle gestione dimpresa` class and set up the *subClassOf* relation between the `Specialist e tecnici delle gestione dimpresa` class and the `Professioni specialistiche e tecniche` class. 2. Create the `Professioni operative della gestione dimpresa` class. 3. Create the `Occupation` class. 4. Set up the *subClassOf* relation between the `Professioni specialistiche e tecniche` class and the `Occupation` class. 5. Set up the *subClassOf* relation between the `Professioni operative della gestione dimpresa` class and the `Occupation` class.

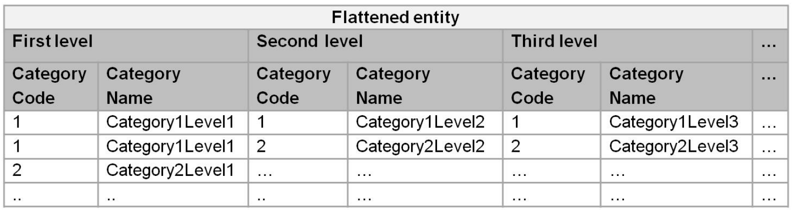
Table 4.3: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model(continued)

Slot	Value
Relationships	
Relations to other modelling components	Use the Architectural Pattern: TX-AP-01 [SFBG+07]

PR-NOR-CLTX-04. Pattern for re-engineering a classification scheme which follows the flattened data model

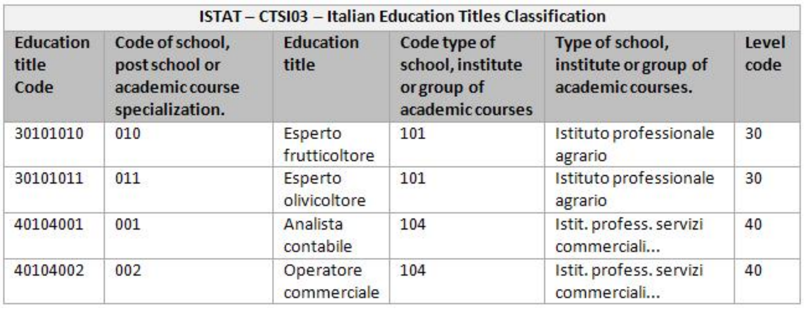
The pattern for re-engineering non-ontological resource shown in Table 4.4 suggests a guide to transform a classification scheme into an ontology. The classification scheme is modeled/represented with a flattened data model. This pattern aims at creating a taxonomy from the classification scheme, being the semantics of the relations between classification scheme items the *subClassOf* relationship.

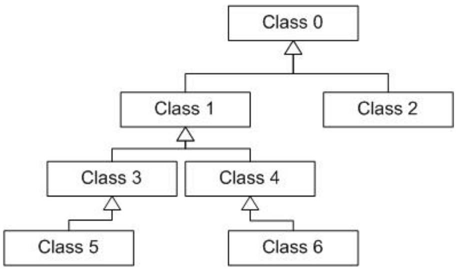
Table 4.4: Pattern for Re-engineering a Classification Scheme which follows the flattened data model.

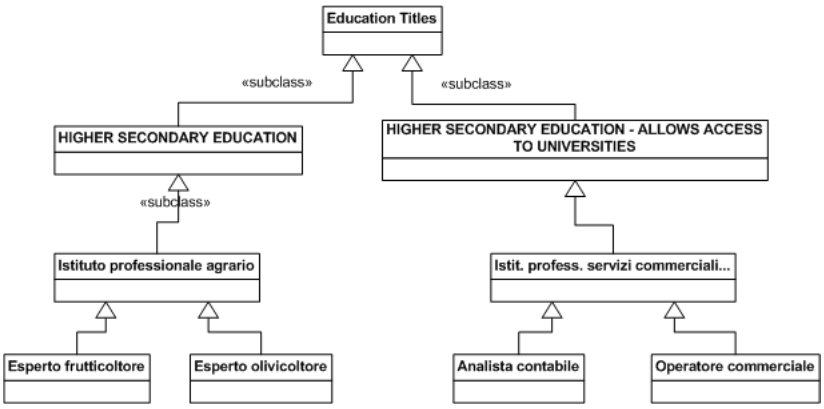
Slot	Value
General Information	
Name	Classification scheme to Taxonomy (flattened model)
Identifier	PR-NOR-CLTX-04
Type of Component	Pattern for Re-engineering Non-Ontological Resource (PR-NOR)
Use Case	
General	Re-engineering a classification scheme which follows the flattened model to design a taxonomy.
Example	Suppose that someone wants to build an ontology based on a classification published as one table with a column for each classification level.
Pattern for Re-engineering Non-Ontological Resource	
INPUT: Resource to be Re-engineered	
General	A non-ontological resource holds a classification scheme which follows the flattened data model. A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parents and children concepts may vary depending of the context. The flattened data model [MZ06] is a denormalized structure for hierarchy representations. In this case, each hierarchy level is represented on a different column. There are as many columns as levels the classification scheme has. Therefore each row has the complete path from the root to a leaf node.
Example	The Classification of Italian Education Titles published by the National Institute of Statistics (ISTAT) is represented following a flattened model. The first level of the classification (level code) is related to the education title level which comprises values as elementary, media, university, master, etc. The second level of the classification is the type of school or institute which offers the education title. The last level is the education title itself; it has a specific specialization code and also a code which is the concatenation of the previous code levels.
Graphical Representation	
General	

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	...	...	...	...	...
..	..	..	...	...	...	...

Table 4.4: Pattern for Re-engineering a Classification Scheme which follows the flattened data model(continued)

<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>Example</b></td><td>


<table>
<tr><th colspan="6">ISTAT – CTSI03 – Italian Education Titles Classification</th></tr>
<tr><th>Education title Code</th><th>Code of school, post school or academic course specialization.</th><th>Education title</th><th>Code type of school, institute or group of academic courses</th><th>Type of school, institute or group of academic courses.</th><th>Level code</th></tr>
<tr><td>30101010</td><td>010</td><td>Esperto frutticoltore</td><td>101</td><td>Istituto professionale agrario</td><td>30</td></tr>
<tr><td>30101011</td><td>011</td><td>Esperto olivicoltore</td><td>101</td><td>Istituto professionale agrario</td><td>30</td></tr>
<tr><td>40104001</td><td>001</td><td>Analista contabile</td><td>104</td><td>Istit. profess. servizi commerciali...</td><td>40</td></tr>
<tr><td>40104002</td><td>002</td><td>Operatore commerciale</td><td>104</td><td>Istit. profess. servizi commerciali...</td><td>40</td></tr>
</table>
</td></tr>
<tr><th colspan="2">OUTPUT: Designed Ontology</th></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the taxonomy architectural pattern (AP-TX-01) [SFBG+07]. Each category in the classification scheme is mapped to a class, and the semantics of the relationship between children and parent classification scheme items are mapped to <i>subClassOf</i> relations.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>


</td></tr>
<tr><td><b>(UML) Example Solution Ontology</b></td><td>


Education Titles
«subclass»
«subclass»
HIGHER SECONDARY EDUCATION
HIGHER SECONDARY EDUCATION - ALLOWS ACCESS TO UNIVERSITIES
«subclass»
Istituto professionale agrario
Istit. profess. servizi commerciali...
Esperto frutticoltore
Esperto olivicoltore
Analista contabile
Operatore commerciale
</td></tr>
<tr><th colspan="2">PROCESS: How to Re-engineer</th></tr>
</table>

Table 4.4: Pattern for Re-engineering a Classification Scheme which follows the flattened data model(continued)

Slot	Value
General	1. Select all the classification scheme items from the first level, using the level column and taking care to avoid duplicity. 2. For each one of the above selected classification scheme items ce_i: 2.1. Create the corresponding ontology class, C_i class. 2.2. Identify the classification scheme items, ce_j, on the next level, which are children of ce_i, by using the path and level columns. 2.3. For each one of the above identified classification scheme items ce_j: 2.3.1. Create the corresponding ontology class, C_j class. 2.3.2. Set up the *subClassOf* relation between C_j and C_i. 2.3.3. Repeat from step 2.2 for ce_j as a new ce_i. 3. If there are more than one classification scheme items from the first level ce_i 3.1. Create an *ad-hoc* class as the root class of the ontology. 3.2. Set up the *subClassOf* relation between C_i class and the root class.
Example	1. Create the HIGHER SECONDARY EDUCATION class. 1.1. Create the Istituto professionale agrario class and set up the *subClassOf* relation between the Istituto professionale agrario class and the HIGHER SECONDARY EDUCATION class. 1.1.1. Create the Esperto frutticoltore class and set up the *subClassOf* relation between the Esperto frutticoltore class and the Istituto professionale agrario class. 1.1.2. Create the Esperto olivicoltore class and set up the *subClassOf* relation between the Esperto olivicoltore class and the Istituto professionale agrario class. 2. Create the HIGHER SECONDARY EDUCATION - ALLOWS ACCESS TO UNIVERSITIES class. 3. Create the Education Title class. 4. Set up the *subClassOf* relation between the HIGHER SECONDARY EDUCATION class and the Education Title class. 5. Set up the *subClassOf* relation between the HIGHER SECONDARY EDUCATION - ALLOWS ACCESS TO UNIVERSITIES class and the Education Title class.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: TX-AP-01 [SFBG+07]

4.3.2 Patterns for Re-engineering Classification Schemes into Lightweight Ontologies

In this section we present the re-engineering patterns (PR-NOR) for re-engineering classification schemes into lightweight ontologies. Since a lightweight ontology [SFBG+07] has additional *ad-hoc* relations between classes, and, path enumeration and adjacency list data models only deal with one relationship, the patterns proposed in this section are only based on snowflake and flattened data models, because they deal with more than one relation. These patterns follow the transformation approach, mentioned in section 2.2.2, for transforming resource schema into an ontology schema, and resource content into ontology instances, because it is the suitable for dealing with the resource data models and the target lightweight ontologies.

PR-NOR-CLLO-03. Pattern for re-engineering a classification scheme which follows the snowflake data model

The pattern for re-engineering classification scheme shown in Table 4.5 suggests a guide to transform a classification scheme into a lightweight ontology. The classification scheme is modeled with a snowflake data model. This pattern aims at creating a lightweight ontology from the classification scheme, which the semantics of the relations between classification scheme items are *subClassOf* and *ad-hoc* relations.

Table 4.5: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model

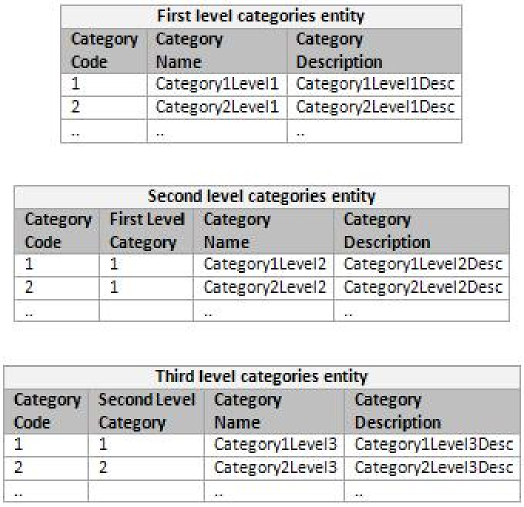
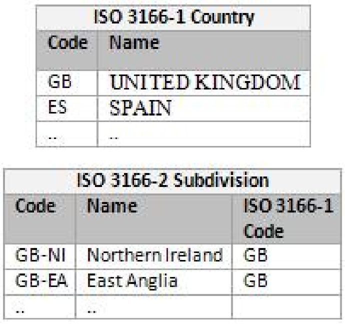
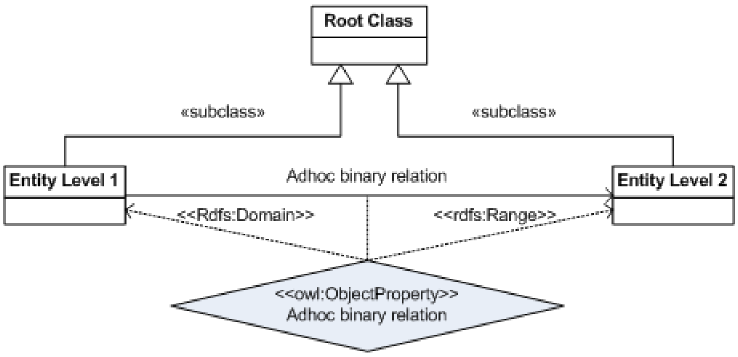
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><th colspan="2">General Information</th></tr>
<tr><td><b>Name</b></td><td>Classification scheme to Lightweight Ontology (snowflake model)</td></tr>
<tr><td><b>Identifier</b></td><td>PR-NOR-CLLO-03</td></tr>
<tr><td><b>Type of Component</b></td><td>Pattern for Re-engineering Non-Ontological Resources (PR-NOR)</td></tr>
<tr><th colspan="2">Use Case</th></tr>
<tr><td><b>General</b></td><td>Re-engineering a classification scheme which follows the snowflake model to design a Lightweight Ontology.</td></tr>
<tr><td><b>Example</b></td><td>Suppose that someone wants to build a lightweight ontology based on the ISO 3166 standard for the representation of names of countries and their subdivisions. This standard is divided in ISO 3166-1 for countries, and ISO 3166-2 for subdivisions (regions).</td></tr>
<tr><th colspan="2">Pattern for Re-engineering Non-Ontological Resources</th></tr>
<tr><th colspan="2">INPUT: Resource to be Re-engineered</th></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a classification scheme which follows the snowflake model.
A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parents and children concepts may vary depending on the context.
The snowflake data model [MZ06] is a normalized structure for hierarchy representations. In this case, the classification scheme items are grouped by levels or entities. There are as many groups as levels the classification scheme has.</td></tr>
<tr><td><b>Example</b></td><td>The ISO 3166 standard (codes for the representation of names of countries and their subdivisions) is divided in ISO 3166-1 for countries, and ISO 3166-2 for country subdivisions (regions).
For the example, ISO 3166-1 and ISO 3166-2 are holding on different entities. The relation semantics between the sub-ordinate and the super-ordinate concepts is <i>partOf</i>.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>General</b></td><td>
<table>
<tr><th colspan="3">First level categories entity</th></tr>
<tr><th>Category Code</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>Category1Level1</td><td>Category1Level1Desc</td></tr>
<tr><td>2</td><td>Category2Level1</td><td>Category2Level1Desc</td></tr>
<tr><td>..</td><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="4">Second level categories entity</th></tr>
<tr><th>Category Code</th><th>First Level Category</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>1</td><td>Category1Level2</td><td>Category1Level2Desc</td></tr>
<tr><td>2</td><td>1</td><td>Category2Level2</td><td>Category2Level2Desc</td></tr>
<tr><td>..</td><td></td><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="4">Third level categories entity</th></tr>
<tr><th>Category Code</th><th>Second Level Category</th><th>Category Name</th><th>Category Description</th></tr>
<tr><td>1</td><td>1</td><td>Category1Level3</td><td>Category1Level3Desc</td></tr>
<tr><td>2</td><td>2</td><td>Category2Level3</td><td>Category2Level3Desc</td></tr>
<tr><td>..</td><td></td><td>..</td><td>..</td></tr>
</table>
</td></tr>
</table>

Table 4.5: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model(continued)

<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>Example</b></td><td>

<table>
<tr><th colspan="2">ISO 3166-1 Country</th></tr>
<tr><th>Code</th><th>Name</th></tr>
<tr><td>GB</td><td>UNITED KINGDOM</td></tr>
<tr><td>ES</td><td>SPAIN</td></tr>
<tr><td>..</td><td>..</td></tr>
</table>
<table>
<tr><th colspan="3">ISO 3166-2 Subdivision</th></tr>
<tr><th>Code</th><th>Name</th><th>ISO 3166-1 Code</th></tr>
<tr><td>GB-NI</td><td>Northern Ireland</td><td>GB</td></tr>
<tr><td>GB-EA</td><td>East Anglia</td><td>GB</td></tr>
<tr><td>..</td><td>..</td><td></td></tr>
</table>
</td></tr>
<tr><th colspan="2">OUTPUT: Designed Ontology</th></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG+07]. Each classification scheme item group (i.e. entity) is mapped to a class. An <i>ad-hoc</i> binary relation is defined between the new classes according to the semantics of the relation between super-ordinate and sub-ordinate classification scheme items. Each classification scheme item included on an entity is mapped to an instance of the entity class. The semantics of the relationship between sub-ordinate and super-ordinate instances are mapped to an <i>ad-hoc</i> binary relation instance.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>


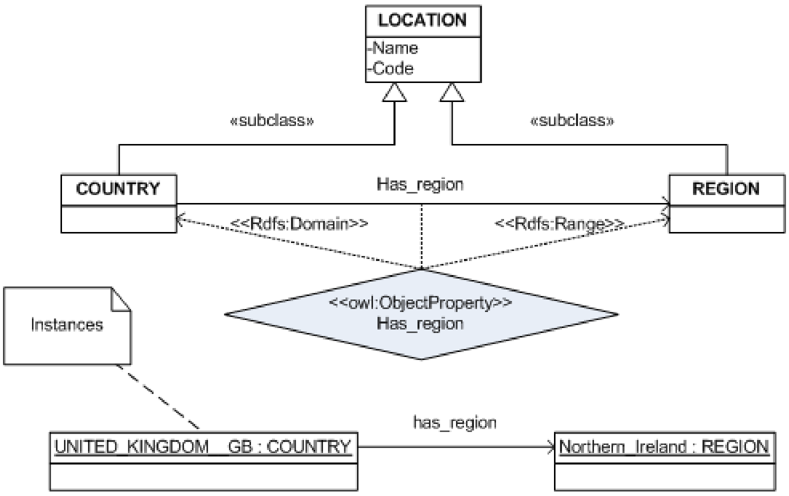
</td></tr>
<tr><td><b>(UML) Example Solution Ontology</b></td><td>

LOCATION
-Name
-Code
«subclass»
«subclass»
COUNTRY
Has_region
REGION
<<Rdfs:Domain>>
<<Rdfs:Range>>
Instances
<<owl:ObjectProperty>>
Has_region
has_region
UNITED_KINGDOM__GB : COUNTRY
Northern_Ireland : REGION
</td></tr>
<tr><th colspan="2">PROCESS: How to Re-engineer</th></tr>
</table>

Table 4.5: Pattern for Re-engineering a Classification Scheme which follows the snowflake data model(continued)

Slot	Value
General	1. Identify the different classification scheme item groups, G_i, i.e. entities, and create a class for each entity, C_i. 2. If there is a relationship between the entity classes then create it as an *ad-hoc* binary relation, $r_i \in R$. 3. If there is a super-class for the new entity related classes then create it and set the appropriate *subClassOf* relation between the entity classes and the super-class. 4. For each classification scheme item, ce_i on each entity of the snowflake model, create an instance of the appropriate entity class, I_j. 5. If you have created an *ad-hoc* binary relation between the entity classes then you have to create the relation instance between the entity class instance.
Example	1. Create a `COUNTRY` class for the ISO 3166-1 Countries entity and a `REGION` class for the ISO 3166-2 Subdivisions entity. 2. Create the *hasRegion* binary relation with `COUNTRY` as domain and `REGION` as range. 3. Create a `LOCATION` class and assert that `COUNTRY` and `REGION` are *subClassOf* `LOCATION`. 4. For each classification scheme item on the ISO 3166-1 Countries entity create an instance of the `COUNTRY` class. 5. For each `COUNTRY` instance look for its `REGION` on the ISO 3166-2 Subdivisions entity and create an instance of `REGION` for each subdivision found. Also create an instance of the *hasRegion* relation associated to the current country instance and related to the current region instance.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG$^+$07]

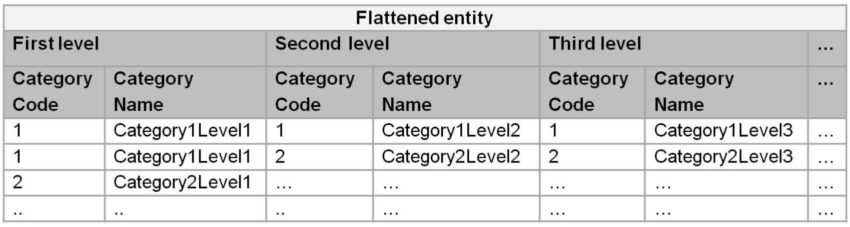
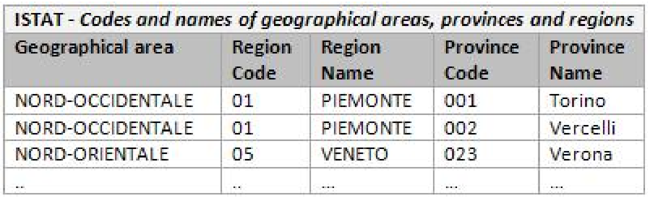
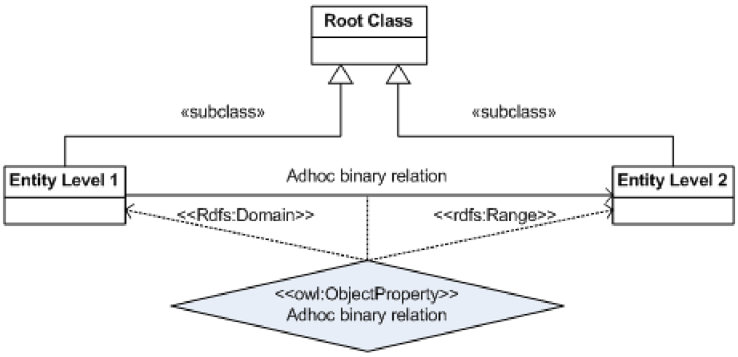
PR-NOR-CLLO-04. Pattern for re-engineering a classification scheme which follows the flattened data model

The pattern for re-engineering classification scheme shown in Table 4.6 suggests a guide to transform a classification scheme into a lightweight ontology. The classification scheme is modeled with a flattened data model. This pattern aims at creating a lightweight ontology from the classification scheme, being the semantics of the relations between classification scheme items the *subClassOf* and *ad-hoc* relationships.

Table 4.6: Pattern for Re-engineering a Classification Scheme which follows the flattened model

Slot	Value
General Information	
Name	Classification scheme to Lightweight Ontology (flattened model)
Identifier	PR-NOR-CLLO-04
Type of Component	Pattern for Re-engineering Non-Ontological Resources (PR-NOR)
Use Case	
General	Re-engineering a classification scheme which follows the flattened model to design a Lightweight Ontology.
Example	Suppose that someone wants to build a lightweight ontology based on the ISTAT[7] standard called *Codes and Names of geographical areas, provinces and regions*. This standard follows the flattened data model.
Pattern for Re-engineering Non-Ontological Resources	

Table 4.6: Pattern for Re-engineering a Classification Scheme which follows the flattened model(continued)

<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><th colspan="2">INPUT: Resource to be Re-engineered</th></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a classification scheme which follows the flattened model. A classification scheme is a rooted tree of concepts, in which each concept groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parents and children concepts may vary depending on the context.
A flattened data model [MZ06] is a denormalized structure for hierarchy representations. In this case, each hierarchy level is represented on a different column. There are as many columns as levels the classification scheme has.</td></tr>
<tr><td><b>Example</b></td><td>The <i>ISTAT standard, Codes and names of geographical areas, provinces and regions</i>, provides codes and names of Italian geographical areas, provinces and regions and it follows a flattened data model in a single denormalized table with six attributes (geography area, province and region codes and names).</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>General</b></td><td>

<table>
<tr><th colspan="7">Flattened entity</th></tr>
<tr><th colspan="2">First level</th><th colspan="2">Second level</th><th colspan="2">Third level</th><th>…</th></tr>
<tr><th>Category Code</th><th>Category Name</th><th>Category Code</th><th>Category Name</th><th>Category Code</th><th>Category Name</th><th>…</th></tr>
<tr><td>1</td><td>Category1Level1</td><td>1</td><td>Category1Level2</td><td>1</td><td>Category1Level3</td><td>…</td></tr>
<tr><td>1</td><td>Category1Level1</td><td>2</td><td>Category2Level2</td><td>2</td><td>Category2Level3</td><td>…</td></tr>
<tr><td>2</td><td>Category2Level1</td><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>..</td><td>..</td><td>..</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
</table>
</td></tr>
<tr><td><b>Example</b></td><td>

<table>
<tr><th colspan="5">ISTAT - <i>Codes and names of geographical areas, provinces and regions</i></th></tr>
<tr><th>Geographical area</th><th>Region Code</th><th>Region Name</th><th>Province Code</th><th>Province Name</th></tr>
<tr><td>NORD-OCCIDENTALE</td><td>01</td><td>PIEMONTE</td><td>001</td><td>Torino</td></tr>
<tr><td>NORD-OCCIDENTALE</td><td>01</td><td>PIEMONTE</td><td>002</td><td>Vercelli</td></tr>
<tr><td>NORD-ORIENTALE</td><td>05</td><td>VENETO</td><td>023</td><td>Verona</td></tr>
<tr><td>..</td><td>..</td><td>...</td><td>...</td><td>...</td></tr>
</table>
</td></tr>
<tr><th colspan="2">OUTPUT: Designed Ontology</th></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG+07]. Each flattened model level is mapped to a class. If there is a superclass related to the new entity classes it has to be created as the root and the <i>subClassOf</i> relation has to be set between the entity classes and the root class. Optionally an <i>ad-hoc</i> binary relation is defined between the new classes according to the semantics of the relationship between super-ordinate and subordinate categories. Each data related to a classification scheme item level is mapped to an instance of the classification scheme item level class. Because of the denormalized nature of the model, it is necessary to avoid the instance duplicity. The semantics of the relationship between subordinate and super-ordinate instances is mapped to an <i>ad-hoc</i> binary relation instance.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>


</td></tr>
</table>

Table 4.6: Pattern for Re-engineering a Classification Scheme which follows the flattened model(continued)

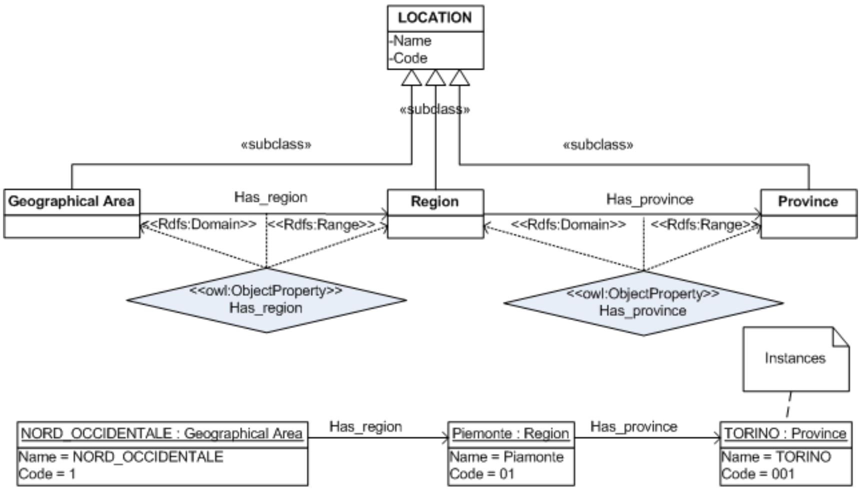
Slot	Value
(UML) Example Solution Ontology	 LOCATION -Name -Code «subclass» «subclass» «subclass» Geographical Area Has_region Region Has_province Province <<Rdfs:Domain>> <<Rdfs:Range>> <<Rdfs:Domain>> <<Rdfs:Range>> <<owl:ObjectProperty>> Has_region <<owl:ObjectProperty>> Has_province Instances NORD_OCCIDENTALE : Geographical Area Name = NORD_OCCIDENTALE Code = 1 Has_region Piemonte : Region Name = Piamonte Code = 01 Has_province TORINO : Province Name = TORINO Code = 001
PROCESS: How to Re-engineer	
General	1. Identify the different classification scheme item groups, i.e. G_i levels, and create a class for each level, C_i. 2. If there is a relationship between the entity classes then create it as an *ad-hoc* binary relation, $r_i \in R$. 3. If there is a super-class for the new entity related classes then create it and set the appropriate *subClassOf* relation between the entity classes and the super-class. 4. Identify the classification scheme items from the first level, $ce_i \in G_1$, using the level column and taking care to avoid duplicity. 5. For each one of the above selected classification scheme items ce_i: 5.1. Create the corresponding instance of the appropriate entity class, I_i. 5.2. Identify the classification scheme items, ce_j, on the next level, which are children of ce_i, using the path and level columns. 5.3. For each one of the identified classification scheme items, ce_j: 5.3.1. Create the corresponding instance of the appropriate entity class, I_j. 5.3.2. If there is a relation between the entity classes, C_i and C_j, create the relation instance between I_i and I_j. 5.3.3. Repeat from step 5.2 for ce_j as a new ce_i.

Table 4.6: Pattern for Re-engineering a Classification Scheme which follows the flattened model(continued)

Slot	Value
Example	1. Create the Geographical Area, Region, Province classes, according to the ISTAT entities. 2. Create the *hasRegion* binary relation with Geographical Area as domain and Region as range. 3. Create the *hasProvince* binary relation with Region as domain and Province as range. 4. Create the LOCATION class and assert that Geographical Area, Region, and Province are *subClassOf* LOCATION. 5. Create an instance of Geographical Area class for each distinct ISTAT geographical area. 6. Look for the Regions of each Geographical Area instance in the ISTAT regions and create an instance of REGION for each distinct region. Create an instance of the *hasRegion* relation associated to the current Geographical Area instance and related to the current Region instance. 7. Look for the Provinces of each Region instances in the ISTAT provinces and create an instance of Province for each distinct province. Create an instance of the *hasProvince* relation associated to the current region instance and related to the current province instance.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG+07]

4.4 NeOn Method for Re-engineering Classification Schemes

The goal of the classification scheme re-engineering process is to transform a classification scheme into an ontology. The activities of this process are based on the ones presented in 3.1.4: (1) non-ontological resource reverse engineering, which includes tasks 1 to 3, (2) non-ontological resource transformation, which includes tasks 4 and 5, and (3) ontology forward engineering, which includes tasks 6 and 7.

4.4.1 Classification Scheme Transformation

In the following, we outline the specialization of the non-ontological resource transformation activity for classification schemes, and consequently how to carry out tasks 4 and 5 included in Figure 3.2. The classification scheme transformation consists of the following tasks:

- **Task 4. Search for a suitable pattern for re-engineering classification scheme.** The goal of this task is to find out if there is any applicable re-engineering pattern for re-engineering classification scheme useful to transform the classification scheme into a conceptual model. The search for a suitable pattern for re-engineering classification scheme should be done using the NeOn library of patterns[8]. For this search, the following search criteria can be used by the ontology developer:
  - Type of non-ontological resource: classification scheme.
  - Data model: one of the aforementioned data models, e.g.: path enumeration, adjacency list, snowflake, or flattened.
  - Target ontology: taxonomy or lightweight ontology.
  - The semantics of the relations between classification scheme items: *subClassOf*, *parOf*, or other *ad-hoc* relation.

[8] http://www.ontologydesignpatterns.org

- **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 classification scheme into a conceptual model. If a suitable pattern for re-engineering classification scheme is found then the conceptual model is created from the classification scheme 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 classification scheme 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 classification scheme.

Chapter 5

Methods for Re-engineering Thesauri

5.1 Introduction

A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them. Thesauri are the most valuable input for creating, at reasonable cost, ontologies in many domains. They contain, readily available, a wealth of category definitions plus a hierarchy, and they reflect some degree of community consensus [HdB07]. In this chapter we present a definition of thesauri, the existing standards for thesauri, the data models for representing thesauri and the method for re-engineering thesauri into ontologies. Although we recognize that multilinguality is an important and complicating factor in thesaurus conversion, it is no treated in this deliverable.

5.2 Thesaurus Standards

In the field of thesaurus development there are several standards. These standards also provide some guidelines about how the thesaurus should be structured. Figure 5.1, taken from [Lab07], depicts the thesaurus standards evolution. The ISO 2788:1986 standard is the seed of the rest of the standards. The ISO 5964:1985 extends the scope of the ISO 2788:1986 adding a multilingual context. The ANSI/NISO Z39.19-2003 adds management guidelines to principles of monolingual thesauri. The ANSI/NISO Z39.19-2003 was superseded by ANSI/NISO Z39.19-2005. The BS 8723-1:2005 and BS 8723-2:2005 are the british version of the ISO 2788.

In the following we briefly describe the most important thesaurus standards.

- *ISO 2788:1986, guidelines for the establishment and development of monolingual thesauri* [ISO86]. This standard covers some aspects of the selection of indexing terms, the procedures for the control of the vocabulary, and specifically, the way of establishing relationships among these terms (particularly those relations that are used, a priori, in the thesauri), as well as the inclusion and suppression of terms, the methods of compilation, the form and the content of the thesauri, the use of automatic data processing, etc. The indications established in this standard ensure the uniformity of each of the indexing areas or entities. The techniques described by the standard are based on general principles that can be applicable to any kind of subject.

- *ISO 5964:1985, guidelines for the establishment and development of multilingual thesauri* [ISO85]. The guidelines given in this International Standard should be used in conjunction with ISO 2788, and regarded as an extension of the scope of the monolingual guidelines. The majority of procedures and recommendations contained in ISO 2788 are equally valid for a multilingual thesaurus. This applies particularly to general procedures, for example, the forms of terms, the basic thesauri relationships, and management operations such a evaluation and maintenance. Distinction is made between preferred terms and non-preferred terms.

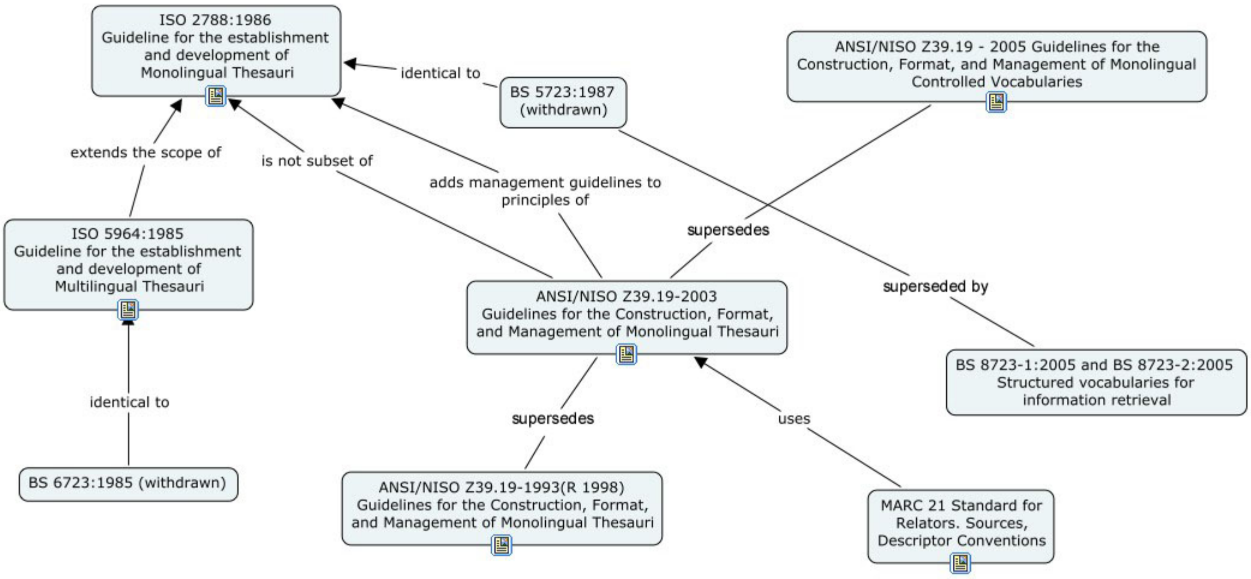

ISO 2788:1986 Guideline for the establishment and development of Monolingual Thesauri
identical to
BS 5723:1987 (withdrawn)
ANSI/NISO Z39.19 - 2005 Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies
extends the scope of
is not subset of
adds management guidelines to principles of
supersedes
ISO 5964:1985 Guideline for the establishment and development of Multilingual Thesauri
ANSI/NISO Z39.19-2003 Guidelines for the Construction, Format, and Management of Monolingual Thesauri
superseded by
BS 8723-1:2005 and BS 8723-2:2005 Structured vocabularies for information retrieval
identical to
supersedes
uses
BS 6723:1985 (withdrawn)
ANSI/NISO Z39.19-1993(R 1998) Guidelines for the Construction, Format, and Management of Monolingual Thesauri
MARC 21 Standard for Relators. Sources, Descriptor Conventions

Figure 5.1: Thesaurus Standards Evolution [Lab07]

- *ANSI/NISO Z39.19-2005, guidelines for the construction, format, and management of monolingual controlled vocabularies* [ANS05]. This standard is related to ISO 2788. It presents guidelines and conventions for the contents, display, construction, testing, maintenance, and management of monolingual controlled vocabularies. It focuses on controlled vocabularies that are used for the representation of content objects in knowledge organization systems including lists, synonym rings, taxonomies, and thesauri.

- *BS 8723-1:2005 and BS 8723-2:2005* [BS 05a, BS 05b]. The British Standard BS 8723-1 defines the terminology used throughout the rest of the BS 8723 series. It provides an excellent glossary for terminology relating to the use of thesauri for indexing and retrieval. The British Standard BS 8723-2 provides guidelines for the construction and maintenance of thesauri that are intended as retrieval tools. Guidance is also given for designers of software supporting the creation and maintenance process.

5.3 Components of a Thesaurus

According to the ANSI/NISO Z39.19-2005 [ANS05] a thesaurus is a controlled vocabulary in a known order and structured so that various relationships among terms are displayed clearly and identified by standardized relationships indicators. There are three types of relationships used in thesaurus:

- *Equivalency*. When the same concept can be expressed by two or more terms, one of these is selected as the preferred term. The relationship between preferred and non-preferred terms is an equivalence relationship in which each term is regarded as referring to the same concept. The equivalence relationship is expressed by the following conventions:
  - U or USE, which leads from a non-preferred term to the preferred term.

  - UF or USED FOR, the reciprocal relationship, which leads from the preferred term to the non-preferred term(s).

  According to the [ANS05] the equivalence relationship covers five basic types: (a) synonyms, (b) lexical variants, (c) near-synonyms, (d) generic posting, and (e) cross reference to elements of compound terms.

- *Hierarchy*. This relationship is based on degrees or levels of superordination and subordination, where the superordinate term represents a class or a whole, and subordinate terms refer to its members or parts. Reciprocity should be expressed by the following relationship indicators: (a) Broader Term (BT), a label for the superordinate (parent) term, and (b) Narrower Term (NT), a label for the subordinate (child) term. Hierarchical relationships cover three logically different and mutually exclusive situations:

  - Generic relationship. This relationship identifies the link between class and its members or species. The generic nature of a relationship may be identified by the following abbreviations: (a) BTG = Broader term (generic) and (b) NTG = Narrower term (generic).
  - Instance relationship. This relationship identifies between a general category of things or events expressed by a common noun, and an individual instance of that category, often a proper name. The hierarchical instance relationship may be indicated by the following abbreviations: (a) BTI = Broader term (instance) and (b) NTI = Narrower term (instance).
  - Whole-part relationship. This relationship covers situations in which one concept is inherently included in another, regardless of context, so that the terms can be organized into logical hierarchies, with the whole treated as a broader term. The hierarchical whole-part relationship may be indicated by the following abbreviations: (a) BTP = Broader term (partitive) and (b) NTP = Narrower term (partitive).

- *Association*. This relationship covers associations between terms that are neither equivalent nor hierarchical. The most common associative relationship used in thesauri is symmetrical and is generally indicated by the abbreviation RT (related term).

A thesaurus captures many relationships among terms, between terms and concepts, and among concepts. A term is a linguistic entity, a character string with meaning in a given language. A concept can be operationally defined as such a group of normalized terms.

5.4 Types of Thesaurus

Soergel [Soe95] identified two types of thesaurus: (1) term-based thesaurus, and (2) concept-based thesaurus. Next, we will describe each one of them.

5.4.1 Term-based Thesaurus

The term-based thesaurus is a collection of terms. Terms are the only type of entity considered. Terms may be related to other terms traditionally using aforementioned relationships, such as: Broader Term (BT), Narrower Term (NT), Related Term (RT), Use For (UF), and Use (U/USE). The ISO 2788:1986 [ISO86] standard proposes a term-based thesaurus structure. Based on ISO 2788:1986 [ISO86] we identify the following thesaurus components (see Figure 5.2):

- *PreferredTerm*, also known as *descriptor*, it is used consistently to represent concepts when indexing documents. It has the following elements: (1) LexicalValue, and (2) identifier.

- *Term*, which is not assigned to documents when indexing, but provided as user's entry point. It has the following elements: (1) LexicalValue, and (2) identifier.

- *ScopeNote*, which is a note following a term explaining its coverage, specialized usage, or rules for assigning it. The *ScopeNote* has a lexicalValue element.
- *HierarchicalRelationship*, which is a relationship between or among terms in the thesaurus that depicts broader (generic) to narrower (specific) or whole-part relationships.
- *AssociativeRelationship*, which is a relationship between or among terms in the thesaurus that leads from one term to other terms that are related to or associated with it.
- *Equivalence*, which is a relationship between or among terms in the thesaurus that leads to one or more terms that are to be used instead of the term from which the cross-reference is made.

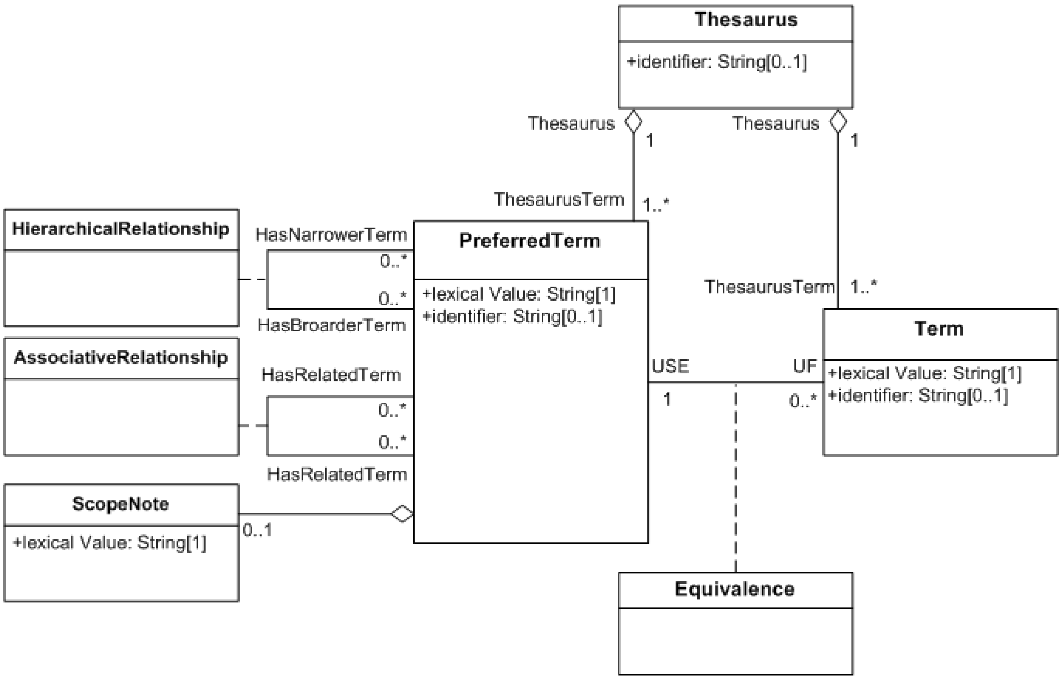


Figure 5.2: UML representation of the term-based thesaurus components [ISO86]

5.4.2 Concept-based Thesaurus

The concept-based thesaurus consists of two types of entity, concepts and terms. A concept is defined as a unit of thought, something which exists in the mind of a person. Relationships such as ’broader’ ’narrower’ and ’related’ are considered to be concept-to-concept relationships, because they convey information about the structure of the concept-space being described. That is, they convey information about meaning. The BS 8723-2:2005 [BS 05b] standard proposes a thesaurus structure that allows for a clear separation of concept information and term information, i.e. a concept-based thesaurus. Based on BS 8723-5:2005 [BS 05c] we identify the following thesaurus components (see Figure 5.3):

- *ThesaurusConcept*, which represents the individual concept within the thesaurus. It has the following elements: (1) identifier, and (2) notation. Additionally, a *ThesaurusConcept* has the following components: (1) *SimpleNonPreferredTerm*, (2) *PreferredTerm* and (3) *ScopeNote*.
- *ThesaurusTerm*, which represents an individual term whithin the thesaurus. It has the following elements: (1) LexicalValue, and (2) identifier. A *ThesaurusTerm* can be a *SimpleNonPreferredTerm* or a *PreferredTerm*.

- *SimpleNonPreferredTerm*, which represents the non-preferred expression of a concept. It is included in a thesaurus mainly to help users find the appropriate preferred term. It inherits all the elements of the *ThesaurusTerm*.

- *PreferredTerm*, which express the preferred form of a concept, as used in the thesaurus. It inherits all the elements of the *ThesaurusTerm*.

- *ScopeNote*, which is a note following a concept explaining its coverage, specialized usage, or rules for assigning it. The *ScopeNote* has the lexicalValue element.

- *HierarchicalRelationship*, which is a relationship between or among concepts in the thesaurus that depicts broader (generic) to narrower (specific) or whole-part relationships.

- *AssociativeRelationship*, which is a relationship between or among concepts in the thesaurus that leads from one concept to other concepts that are related to or associated with it.

- *Equivalence*, which is a relationship between or among terms in the thesaurus that leads to one or more terms that are to be used instead of the term from which the cross-reference is made.

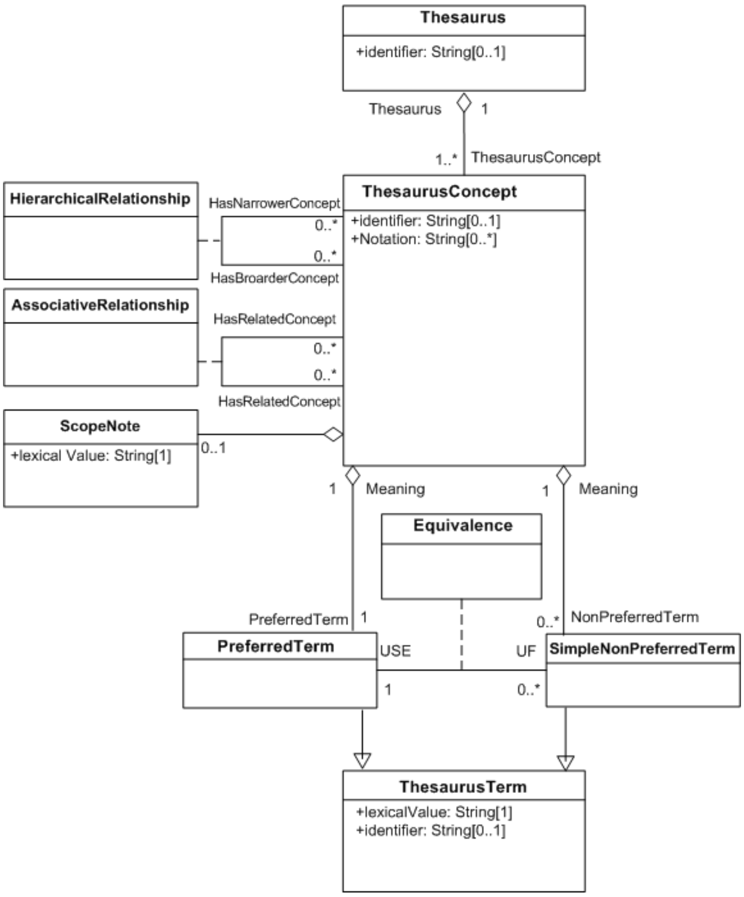
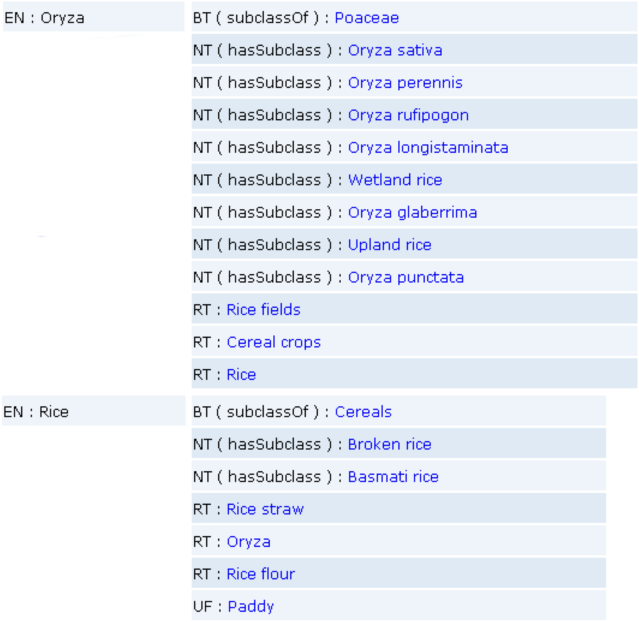


Figure 5.3: UML representation of the concept-based thesaurus components [BS 05c]

5.4.3 Thesaurus Data Models

As we mentioned in section 2.1 there are different ways of representing the knowledge encoded by a particular resource. In this section we present the data models we found for thesauri. Soergel [Soe95] identifies two ways of representing the knowledge encoded by the thesauri: (1) record-based model, and (2) relation-based model. In order to exemplify the data models for thesauri, we use an excerpt from the FAO Thesaurus, AGROVOC[1] presented in Figure 5.4. This Figure shows the terms: Oryza and Rice. Next, we describe the data models for thesauri.



EN : Oryza	BT (subclassOf) : Poaceae
	NT (hasSubclass) : Oryza sativa
	NT (hasSubclass) : Oryza perennis
	NT (hasSubclass) : Oryza rufipogon
	NT (hasSubclass) : Oryza longistaminata
	NT (hasSubclass) : Wetland rice
	NT (hasSubclass) : Oryza glaberrima
	NT (hasSubclass) : Upland rice
	NT (hasSubclass) : Oryza punctata
	RT : Rice fields
	RT : Cereal crops
	RT : Rice
EN : Rice	BT (subclassOf) : Cereals
	NT (hasSubclass) : Broken rice
	NT (hasSubclass) : Basmati rice
	RT : Rice straw
	RT : Oryza
	RT : Rice flour
	UF : Paddy

Figure 5.4: Excerpt of the AGROVOC thesaurus

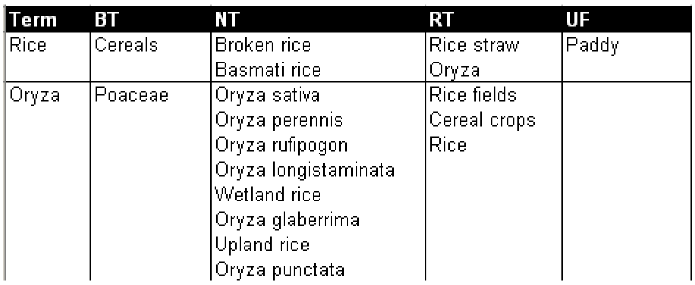
Record-based Model

The record-based model, which is a denormalized structure, uses a record for every term with the information about the term, such as synonyms, broader, narrower and related terms. In this model, the information is stored in large packages, and to access or change any piece of information we must get into the appropriate package. This model looks like the flattened model presented in section 4.2.1. We can apply this model to a term-based thesaurus (Figure 5.5-a)) and to a concept-based thesaurus (Figure 5.5-b)). In the case of the concept-based thesaurus, the information about the concepts is added to each record.

Relation-based Model

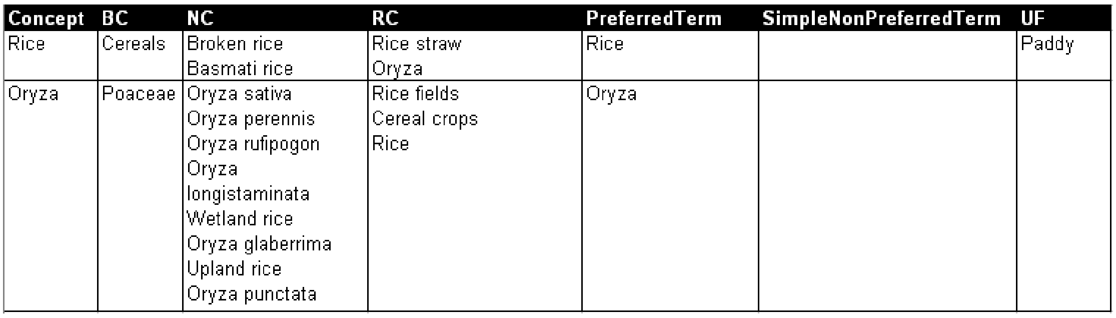
The relation-based model leads to a more elegant and efficient structure. Information is stored in individual pieces that can be arranged in different ways. Relationship types are not defined as fields in a record, but they are simply data values in a relationship record, thus new relationship types can be introduced with ease. In the case of the term-based thesaurus, Figure 5.6-a) ,there are three entities: (1) a term entity, which contains the overall set of terms, (2) a term-term relationship entity, in which each record contains two different term codes and the relationship between them, and (3) a relationship source entity, which contains the overall thesaurus relationships. In the case of the concept-based thesaurus, Figure 5.6-b), there are four entities: (1) a concept entity, which links each term with exactly one concept, thus this entity has one record for each term and for each concept as many records as there are terms, (2) a concept-concept relationship entity, in which BT, NT and RT are established explicitly between concepts, (3) a term-term relationship entity, in which UF and USE relations are established explicitly between terms, this is an optional entity, (4) a relationship source entity, which contains the overall thesaurus relationships.

[1] http://www.fao.org/agrovoc/



Term	BT	NT	RT	UF
Rice	Cereals	Broken rice Basmati rice	Rice straw Oryza	Paddy
Oryza	Poaceae	Oryza sativa Oryza perennis Oryza rufipogon Oryza longistaminata Wetland rice Oryza glaberrima Upland rice Oryza punctata	Rice fields Cereal crops Rice	

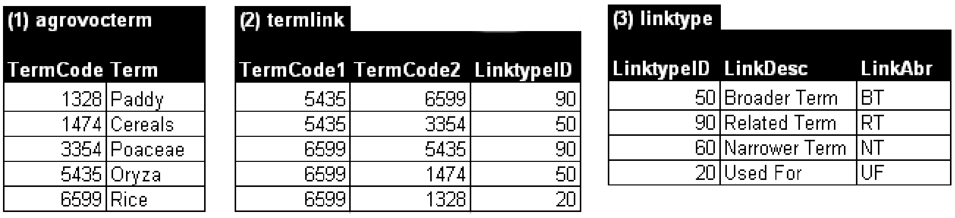
a) Term Based Thesaurus



Concept	BC	NC	RC	PreferredTerm	SimpleNonPreferredTerm	UF
Rice	Cereals	Broken rice Basmati rice	Rice straw Oryza	Rice		Paddy
Oryza	Poaceae	Oryza sativa Oryza perennis Oryza rufipogon Oryza longistaminata Wetland rice Oryza glaberrima Upland rice Oryza punctata	Rice fields Cereal crops Rice	Oryza		

b) Concept Based Thesaurus

Figure 5.5: Record-based model



(1) agrovocterm

TermCode	Term
1328	Paddy
1474	Cereals
3354	Poaceae
5435	Oryza
6599	Rice

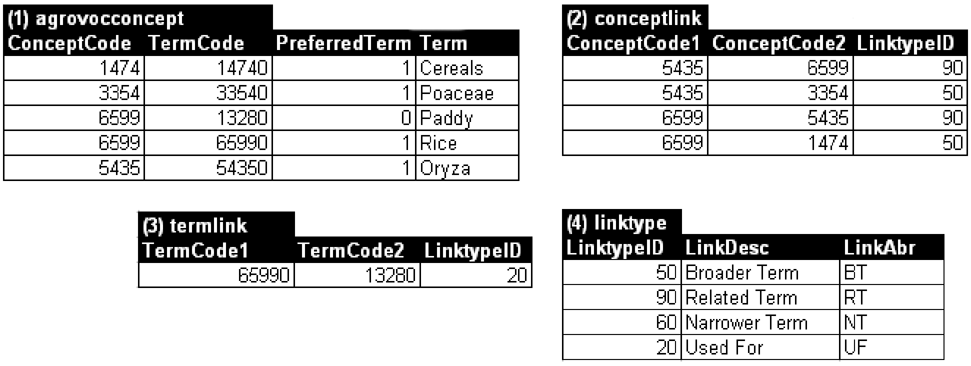
(2) termlink

TermCode1	TermCode2	LinktypeID
5435	6599	90
5435	3354	50
6599	5435	90
6599	1474	50
6599	1328	20

(3) linktype

LinktypeID	LinkDesc	LinkAbr
50	Broader Term	BT
90	Related Term	RT
60	Narrower Term	NT
20	Used For	UF

a) Term Based Thesaurus



(1) agrovocconcept

ConceptCode	TermCode	PreferredTerm	Term
1474	14740	1	Cereals
3354	33540	1	Poaceae
6599	13280	0	Paddy
6599	65990	1	Rice
5435	54350	1	Oryza

(2) conceptlink

ConceptCode1	ConceptCode2	LinktypeID
5435	6599	90
5435	3354	50
6599	5435	90
6599	1474	50

(3) termlink

TermCode1	TermCode2	LinktypeID
65990	13280	20

(4) linktype

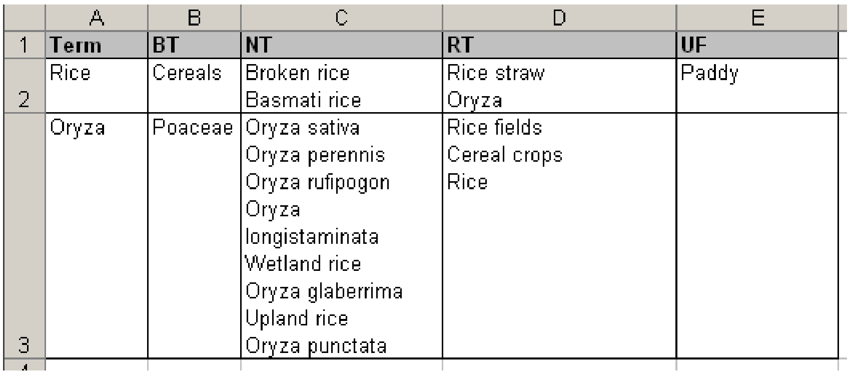
LinktypeID	LinkDesc	LinkAbr
50	Broader Term	BT
90	Related Term	RT
60	Narrower Term	NT
20	Used For	UF

b) Concept Based Thesaurus

Figure 5.6: Relation-based model

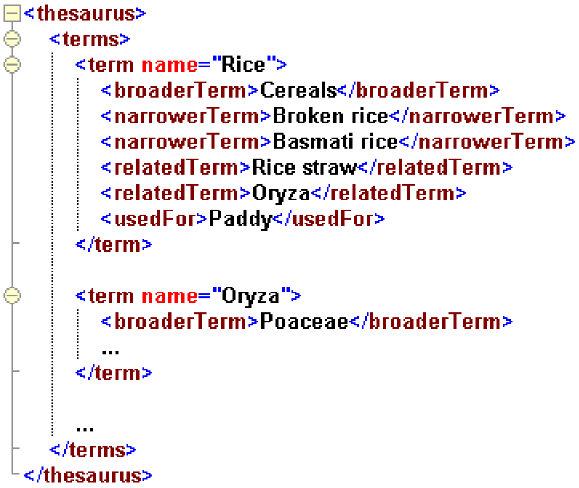
5.4.4 Thesaurus Implementations

Finally these data models can be implemented as any of the identified types on section 2.1, i.e. databases, XML files, flat files, and spreadsheets. A direct implementation would be as tables in a relational database or in a spreadsheet. Figure 5.4.4 presents a spreadsheet implementation of the record-based model of a term-based thesaurus, and Figure 5.4.4 presents an XML implementation of the record-based model of a term-based thesaurus. Both figures present the same excerpt of the AGROVOC thesaurus represented in different implementations.



	A	B	C	D	E
1	**Term**	**BT**	**NT**	**RT**	**UF**
2	Rice	Cereals	Broken rice Basmati rice	Rice straw Oryza	Paddy
3	Oryza	Poaceae	Oryza sativa Oryza perennis Oryza rufipogon Oryza longistaminata Wetland rice Oryza glaberrima Upland rice Oryza punctata	Rice fields Cereal crops Rice	

Figure 5.7: Spreadsheet implementation of the record-based model of a term-based thesaurus



```
<thesaurus>
  <terms>
    <term name="Rice">
      <broaderTerm>Cereals</broaderTerm>
      <narrowerTerm>Broken rice</narrowerTerm>
      <narrowerTerm>Basmati rice</narrowerTerm>
      <relatedTerm>Rice straw</relatedTerm>
      <relatedTerm>Oryza</relatedTerm>
      <usedFor>Paddy</usedFor>
    </term>

    <term name="Oryza">
      <broaderTerm>Poaceae</broaderTerm>
      ...
    </term>

    ...
  </terms>
</thesaurus>
```

Figure 5.8: XML implementation of the record-based model of a term-based thesaurus

Figure 5.9 shows how a given type of thesauri 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, Figure 5.9 shows a term-based thesaurus modeled following a record-based model. In this case, the thesaurus is implemented in a database and in an XML file.

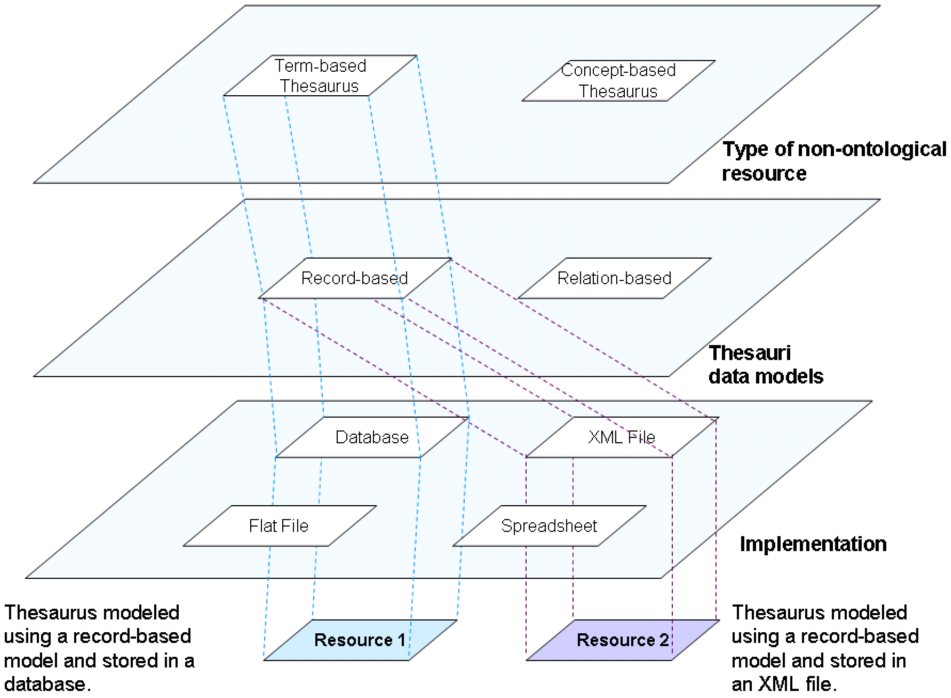

Term-based Thesaurus
Concept-based Thesaurus
Type of non-ontological resource
Record-based
Relation-based
Thesauri data models
Database
XML File
Flat File
Spreadsheet
Implementation
Thesaurus modeled using a record-based model and stored in a database.
Resource 1
Resource 2
Thesaurus modeled using a record-based model and stored in an XML file.

Figure 5.9: Thesauri Categorization

5.5 Patterns for Re-engineering Thesauri into Ontologies

In this section we present the re-engineering patterns (PR-NOR) for re-engineering thesauri into ontologies. These patterns come from the experience of ontology engineers in developing ontologies using thesauri. The patterns are:

- PR-NOR-TSLO-01. Pattern for re-engineering a term-based thesaurus which follows the record-based data model, into a lightweight ontology. In that case, the semantics of the BT/NT relations between terms are *subClassOf*.
- PR-NOR-TSLO-02. Pattern for re-engineering a term-based thesaurus which follows the relation-based data model, into a lightweight ontology. In that case, the semantics of the BT/NT relations between terms are *subClassOf*.
- PR-NOR-TSLO-03. Pattern for re-engineering a concept-based thesaurus which follows the record-based data model, into a lightweight ontology. In that case, the semantics of the BT/NT relations between concepts are *subClassOf*.
- PR-NOR-TSLO-04. Pattern for re-engineering a concept-based thesaurus which follows the relation-based data model, into a lightweight ontology. In that case, which the semantics of the BT/NT relations between concepts are *subClassOf*.

5.5.1 Patterns for re-engineering Thesauri into Lightweight Ontologies

In this section we present the re-engineering patterns (PR-NOR) for re-engineering thesauri into lightweight ontologies. In spite of these patterns consider the semantics of the BT/NT relations as *subClassOf*, it is possible to adapt these patterns for dealing with other relations, e.g. *partOf*, *ad-hoc*. These patterns follow the transformation approach, mentioned in section 2.2.2, for transforming resource schema into an ontology schema, because it is the suitable for dealing with the resource data models and the target lightweight ontologies.

PR-NOR-TSLO-01. Pattern for re-engineering a term-based thesaurus which follows the record-based data model

The pattern for re-engineering thesaurus shown in Table 5.1 suggests a guide to transform a thesaurus into a lightweight ontology. The thesaurus is a term-based one and it is modeled with a record-based data model. This pattern aims at creating a lightweight ontology from the thesaurus, being the semantics of the BT/NT relations between terms the *subClassOf* relationship.

Table 5.1: Pattern for Re-engineering a term-based thesaurus which follows the record-based model

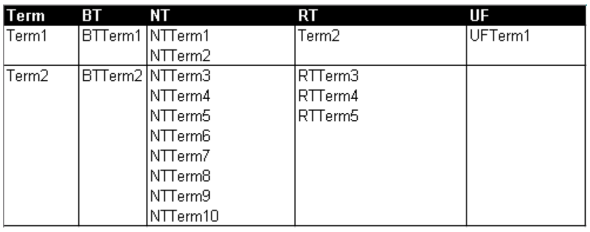
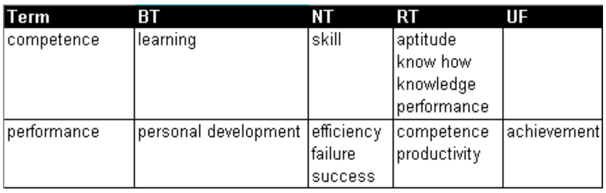
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td colspan="2"><b>General Information</b></td></tr>
<tr><td><b>Name</b></td><td>Term-based Thesaurus to Lightweight Ontology (record-based model)</td></tr>
<tr><td><b>Identifier</b></td><td>PR-NOR-TSLO-01</td></tr>
<tr><td><b>Type of Component</b></td><td>Pattern for Re-engineering Non-Ontological Resources (PR-NOR)</td></tr>
<tr><td colspan="2"><b>Use Case</b></td></tr>
<tr><td><b>General</b></td><td>Re-engineering a term-based thesaurus which follows the record-based model to design a Lightweight Ontology.</td></tr>
<tr><td><b>Example</b></td><td>Suppose that someone wants to build a lightweight ontology based on the European Training Thesaurus (ETT), which is a term-based thesaurus and it follows the record-based model.</td></tr>
<tr><td colspan="2"><b>Pattern for Re-engineering Non-Ontological Resources</b></td></tr>
<tr><td colspan="2"><b>INPUT: Resource to be Re-engineered</b></td></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a term-based thesaurus which follows the record-based model.
A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them.
The record-based data model [Soe95] is a denormalized structure, uses a record for every term with the information about the term, such as synonyms, broader, narrower and related terms.</td></tr>
<tr><td><b>Example</b></td><td>The European Training Thesaurus (ETT) constitutes the controlled vocabulary of reference in the field of vocational education and training (VET) in Europe.
The relation semantics between the sub-ordinate and the super-ordinate concepts is <i>subClassOf</i>.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>General</b></td><td>
<table>
<tr><th>Term</th><th>BT</th><th>NT</th><th>RT</th><th>UF</th></tr>
<tr><td>Term1</td><td>BTTerm1</td><td>NTTerm1
NTTerm2</td><td>Term2</td><td>UFTerm1</td></tr>
<tr><td>Term2</td><td>BTTerm2</td><td>NTTerm3
NTTerm4
NTTerm5
NTTerm6
NTTerm7
NTTerm8
NTTerm9
NTTerm10</td><td>RTTerm3
RTTerm4
RTTerm5</td><td></td></tr>
</table></td></tr>
<tr><td><b>Example</b></td><td>
<table>
<tr><th>Term</th><th>BT</th><th>NT</th><th>RT</th><th>UF</th></tr>
<tr><td>competence</td><td>learning</td><td>skill</td><td>aptitude
know how
knowledge
performance</td><td></td></tr>
<tr><td>performance</td><td>personal development</td><td>efficiency
failure
success</td><td>competence
productivity</td><td>achievement</td></tr>
</table></td></tr>
<tr><td colspan="2"><b>OUTPUT: Designed Ontology</b></td></tr>
</table>

Table 5.1: Pattern for Re-engineering a term-based thesaurus which follows the record-based model(continued)

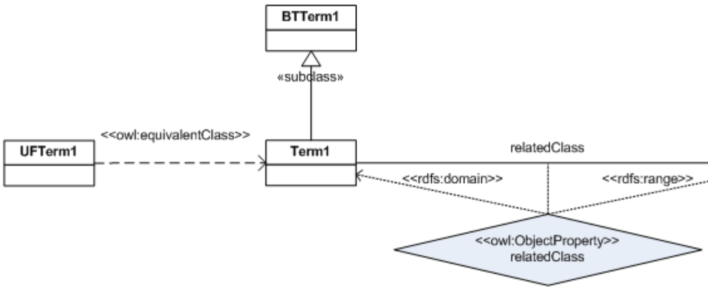
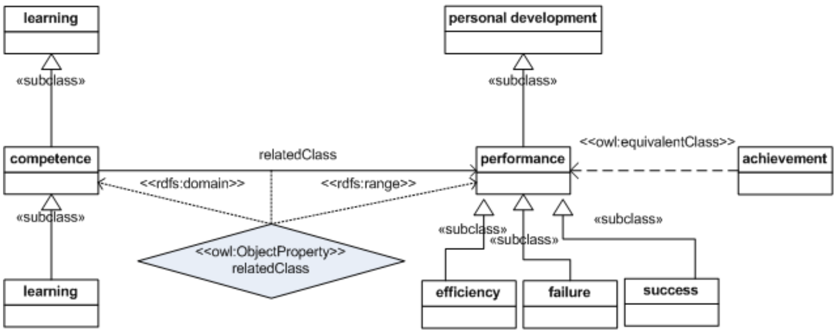
Slot	Value
General	The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG$^+$07]. Each thesaurus term is mapped to a class. A *subClassOf* relation is defined between the new classes for the BT/NT relation. A *relatedClass* relation is defined between the new classes for the RT relation. A *equivalentClass* relation is defined between the new classes for the UF/USE relation.
Graphical Representation	
(UML) General Solution Ontology	 BTTerm1 «subclass» BTTerm2 «subclass» <<owl:equivalentClass>> UFTerm1 Term1 relatedClass Term2 <<rdfs:domain>> <<rdfs:range>> <<owl:ObjectProperty>> relatedClass
(UML) Example Solution Ontology	 learning «subclass» personal development «subclass» <<owl:equivalentClass>> competence relatedClass performance achievement <<rdfs:domain>> <<rdfs:range>> «subclass» «subclass» «subclass» «subclass» <<owl:ObjectProperty>> relatedClass learning efficiency failure success
PROCESS: How to Re-engineer	
General	1. Identify the records which contain thesaurus terms without a *broader term*. 2. For each one of the above identified thesaurus terms t_i: 2.1. Create the corresponding ontology class, C_i class, if it is not created yet. 2.2. Identify the thesaurus term, t_j, which are narrower terms of t_i. They are referenced in the same record which contains t_i. 2.3. For each one of the above identified thesaurus term t_j: 2.3.1. Create the corresponding ontology class, C_j class, if it is not created yet. 2.3.2. Set up the *subClassOf* relation between C_j and C_i 2.3.3. Repeat from step 2.2 for c_j as a new c_i 2.4. Identify the thesaurus term, t_r, which are related terms of t_i. They are referenced in the same record which contains t_i. 2.5. For each one of the above identified thesaurus term t_r: 2.5.1. Create the corresponding ontology class, C_r class, if it is not created yet. 2.5.2. Set up the *relatedClass* relation between C_r and C_i 2.5.3. Repeat from step 2.4 for c_r as a new c_i 2.6. Identify the thesaurus term, t_q, which are equivalent terms of t_i. They are referenced in the same record which contains t_i. 2.7. For each one of the above identified thesaurus term t_q: 2.7.1. Create the corresponding ontology class, C_q class, if it is not created yet. 2.7.2. Set up the *equivalentClass* relation between C_q and C_i 2.7.3. Repeat from step 2.6 for c_q as a new c_i.

Table 5.1: Pattern for Re-engineering a term-based thesaurus which follows the record-based model(continued)

Slot	Value
Example	1. Create the `learning` class and the `personal development` class. 2. Create the `competence` class and assert that `competence` is *subClassOf* `learning`. 3. Create the `performance` class and assert that `performance` is *subClassOf* `development`. 4. Create the `achievement` class and assert that `achievement` is *equivalentClass* of `performance`. 5. Assert that `competence` is *relatedClass* of `performance`. 6. Create the `learning` class and assert that `learning` is *equivalentClass* of `competence`. 6.1. Create the `efficiency` class and assert that `efficiency` is *subClassOf* `performance`.ť 6.2. Create the `failure` class and assert that `failure` is *subClassOf* `performance`. 6.3. Create the `success` class and assert that `success` is *subClassOf* `performance`.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG+07]

PR-NOR-TSLO-02. Pattern for re-engineering a term-based thesaurus which follows the relation-based data model

The pattern for re-engineering thesaurus shown in Table 5.2 suggests a guide to transform a thesaurus into a lightweight ontology. The thesaurus is a term-based one and it is modeled with a relation- based data model. This pattern aims at creating a lightweight ontology from the thesaurus, being the semantics of the BT/NT relations between terms the *subClassOf* relationship.

Table 5.2: Pattern for Re-engineering a term-based thesaurus which follows the relation-based model

Slot	Value
General Information	
Name	Term-based Thesaurus to Lightweight Ontology (record-based model)
Identifier	PR-NOR-TSLO-02
Type of Component	Pattern for Re-engineering Non-Ontological Resources (PR-NOR)
Use Case	
General	Re-engineering a term-based thesaurus which follows the relation-based model to design a Lightweight Ontology.
Example	Suppose that someone wants to build a lightweight ontology based on earlier version of the AGROVOC Thesaurus, which is a term-based thesaurus and it follows the relation-based model.
Pattern for Re-engineering Non-Ontological Resources	
INPUT: Resource to be Re-engineered	
General	A non-ontological resource holds a term-based thesaurus which follows the relation-based model. A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them. The relation-based data model [Soe95] is a normalized structure, in which relationship types are not defined as fields in a record, but they are simply data values in a relationship record, thus new relationship types can be introduced with ease.

Table 5.2: Pattern for Re-engineering a Thesaurus(continued)

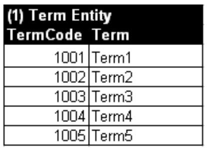
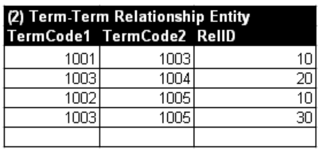
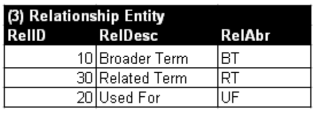
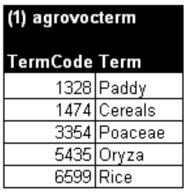
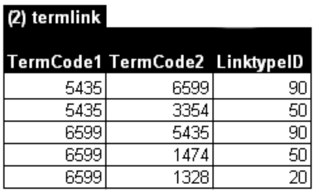
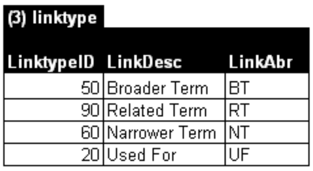
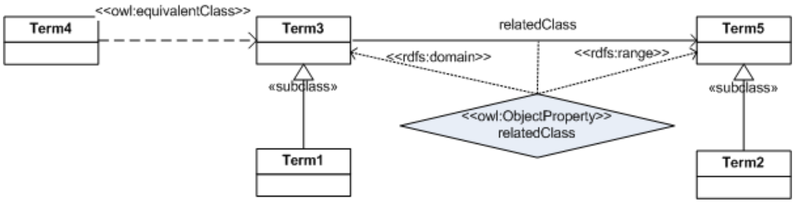
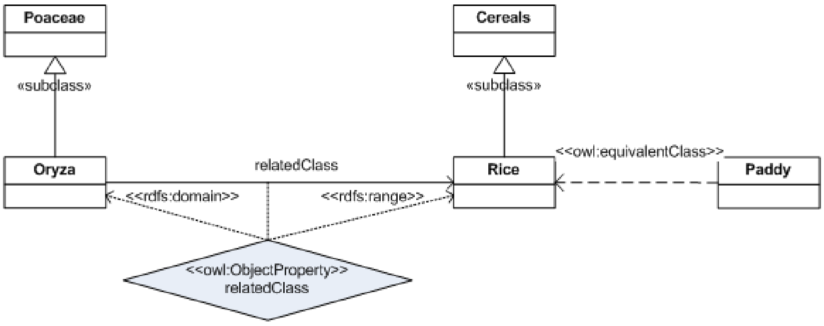
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>Example</b></td><td>The AGROVOC Thesaurus is an structured and controlled vocabulary designed to cover the terminology of all subject fields in agriculture, forestry, fisheries, food and related domains. The relation semantics between the sub-ordinate and the super-ordinate concepts is <i>subClassOf</i>.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>General</b></td><td>

<table>
<tr><th colspan="2">(1) Term Entity</th></tr>
<tr><th>TermCode</th><th>Term</th></tr>
<tr><td>1001</td><td>Term1</td></tr>
<tr><td>1002</td><td>Term2</td></tr>
<tr><td>1003</td><td>Term3</td></tr>
<tr><td>1004</td><td>Term4</td></tr>
<tr><td>1005</td><td>Term5</td></tr>
</table>

<table>
<tr><th colspan="3">(2) Term-Term Relationship Entity</th></tr>
<tr><th>TermCode1</th><th>TermCode2</th><th>RelID</th></tr>
<tr><td>1001</td><td>1003</td><td>10</td></tr>
<tr><td>1003</td><td>1004</td><td>20</td></tr>
<tr><td>1002</td><td>1005</td><td>10</td></tr>
<tr><td>1003</td><td>1005</td><td>30</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

<table>
<tr><th colspan="3">(3) Relationship Entity</th></tr>
<tr><th>RelID</th><th>RelDesc</th><th>RelAbr</th></tr>
<tr><td>10</td><td>Broader Term</td><td>BT</td></tr>
<tr><td>30</td><td>Related Term</td><td>RT</td></tr>
<tr><td>20</td><td>Used For</td><td>UF</td></tr>
</table>
</td></tr>
<tr><td><b>Example</b></td><td>

<table>
<tr><th colspan="2">(1) agrovocterm</th></tr>
<tr><th>TermCode</th><th>Term</th></tr>
<tr><td>1328</td><td>Paddy</td></tr>
<tr><td>1474</td><td>Cereals</td></tr>
<tr><td>3354</td><td>Poaceae</td></tr>
<tr><td>5435</td><td>Oryza</td></tr>
<tr><td>6599</td><td>Rice</td></tr>
</table>

<table>
<tr><th colspan="3">(2) termlink</th></tr>
<tr><th>TermCode1</th><th>TermCode2</th><th>LinktypeID</th></tr>
<tr><td>5435</td><td>6599</td><td>90</td></tr>
<tr><td>5435</td><td>3354</td><td>50</td></tr>
<tr><td>6599</td><td>5435</td><td>90</td></tr>
<tr><td>6599</td><td>1474</td><td>50</td></tr>
<tr><td>6599</td><td>1328</td><td>20</td></tr>
</table>

<table>
<tr><th colspan="3">(3) linktype</th></tr>
<tr><th>LinktypeID</th><th>LinkDesc</th><th>LinkAbr</th></tr>
<tr><td>50</td><td>Broader Term</td><td>BT</td></tr>
<tr><td>90</td><td>Related Term</td><td>RT</td></tr>
<tr><td>60</td><td>Narrower Term</td><td>NT</td></tr>
<tr><td>20</td><td>Used For</td><td>UF</td></tr>
</table>
</td></tr>
<tr><td colspan="2"><b>OUTPUT: Designed Ontology</b></td></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG+07]. Each thesaurus term is mapped to a class. A <i>subClassOf</i> relation is defined between the new classes for the BT/NT relation. A <i>relatedClass</i> relation is defined between the new classes for the RT relation. A <i>equivalentClass</i> relation is defined between the new classes for the UF/USE relation.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>

Term4
<<owl:equivalentClass>>
Term3
relatedClass
Term5
<<rdfs:domain>>
<<rdfs:range>>
«subclass»
«subclass»
<<owl:ObjectProperty>>
relatedClass
Term1
Term2
</td></tr>
<tr><td><b>(UML) Example Solution Ontology</b></td><td>

Poaceae
Cereals
«subclass»
«subclass»
Oryza
relatedClass
Rice
<<owl:equivalentClass>>
Paddy
<<rdfs:domain>>
<<rdfs:range>>
<<owl:ObjectProperty>>
relatedClass
</td></tr>
<tr><td colspan="2"><b>PROCESS: How to Re-engineer</b></td></tr>
</table>

Table 5.2: Pattern for Re-engineering a Thesaurus(continued)

Slot	Value
General	1. Identify the records which contain thesaurus terms without a *broader term*, within the term-term relationship entity. 2. For each one of the above identified thesaurus terms t_i: 2.1. Obtain the thesaurus term within the term entity. 2.2. Create the corresponding ontology class, C_i class, if it is not created yet. 2.3. Identify the thesaurus term, t_j, which are narrower terms of t_i, within the term-term relationship entity. 2.4. For each one of the above identified thesaurus terms t_j: 2.4.1. Obtain the thesaurus term within the term entity. 2.4.2. Create the corresponding ontology class, C_j class, if it is not created yet. 2.4.3. Set up the *subClassOf* relation between C_j and C_i 2.4.4. Repeat from step 2.2 for c_j as a new c_i 2.5. Identify the thesaurus term, t_r, which are related terms of t_i, within the term-term relationship entity. 2.6. For each one of the above identified thesaurus term t_r: 2.6.1. Obtain the thesaurus term within the term entity. 2.6.2. Create the corresponding ontology class, C_r class, if it is not created yet. 2.6.3. Set up the *relatedClass* relation between C_r and C_i 2.6.4. Repeat from step 2.4 for c_r as a new c_i 2.7. Identify the thesaurus term, t_q, which are equivalent terms of t_i, within the term-term relationship entity. 2.8. For each one of the above identified thesaurus term t_q: 2.8.1. Obtain the thesaurus term within the term entity. 2.8.2. Create the corresponding ontology class, C_q class, if it is not created yet. 2.8.3. Set up the *equivalentClass* relation between C_q and C_i 2.8.4. Repeat from step 2.6 for c_q as a new c_i
Example	1. Create the `Poaceae` class. 1.1. Create the `Oryza` class and assert that `Oryza` is *subClassOf* `Poaceae`. 1.1.1. Create the `Rice` class and assert that `Rice` is *relatedClass* of `Oryza`. 2. Create the `Cereals` class. 2.1. Assert that `Rice` is *subClassOf* `Cereals`. 2.2. Create the `Paddy` class and assert that `Paddy` is *equivalentClass* of `Rice`.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG$^+$07]

PR-NOR-TSLO-03. Pattern for re-engineering a concept-based thesaurus which follows the record-based data model

The pattern for re-engineering thesaurus shown in Table 5.3 suggests a guide to transform a thesaurus into a lightweight ontology. The thesaurus is a concept-based one and it is modeled with a record-based data model. This pattern aims at creating a lightweight ontology from the thesaurus, being the semantics of the BT/NT relations between terms the *subClassOf* relationship.

Table 5.3: Pattern for Re-engineering a concept-based thesaurus which follows the record-based model

Slot	Value
General Information	
Name	Concept-based Thesaurus to Lightweight Ontology (record-based model)

Table 5.3: Pattern for Re-engineering a Thesaurus(continued)

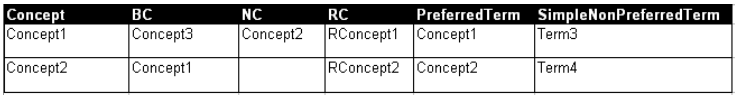
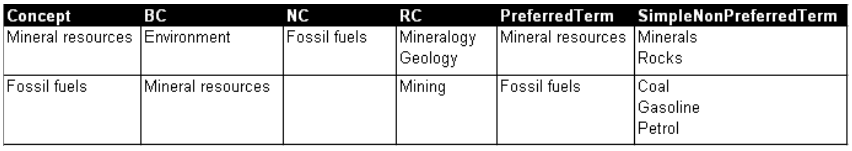
<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>Identifier</b></td><td>PR-NOR-TSLO-03</td></tr>
<tr><td><b>Type of Component</b></td><td>Pattern for Re-engineering Non-Ontological Resources (PR-NOR)</td></tr>
<tr><td colspan="2"><b>Use Case</b></td></tr>
<tr><td><b>General</b></td><td>Re-engineering a concept-based thesaurus which follows the record-based model to design a Lightweight Ontology.</td></tr>
<tr><td><b>Example</b></td><td>Suppose that someone wants to build a lightweight ontology based on the Integrated Public Sector Vocabulary (IPSV), used in UK for indexing government documents. This thesaurus is a concept-based thesaurus and it follows the record-based model.</td></tr>
<tr><td colspan="2"><b>Pattern for Re-engineering Non-Ontological Resources</b></td></tr>
<tr><td colspan="2"><b>INPUT: Resource to be Re-engineered</b></td></tr>
<tr><td><b>General</b></td><td>A non-ontological resource holds a concept-based thesaurus which follows the record-based model.
A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them.
The record-based data model [Soe95] is a denormalized structure, uses a record for every term with the information about the term, such as synonyms, broader, narrower and related terms.</td></tr>
<tr><td><b>Example</b></td><td>The Integrated Public Sector Vocabulary (IPSV) is used in UK for indexing government documents.
The relation semantics between the sub-ordinate and the super-ordinate concepts is <i>subClassOf</i>.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
<tr><td><b>General</b></td><td>
<table>
<tr><th>Concept</th><th>BC</th><th>NC</th><th>RC</th><th>PreferredTerm</th><th>SimpleNonPreferredTerm</th></tr>
<tr><td>Concept1</td><td>Concept3</td><td>Concept2</td><td>RConcept1</td><td>Concept1</td><td>Term3</td></tr>
<tr><td>Concept2</td><td>Concept1</td><td></td><td>RConcept2</td><td>Concept2</td><td>Term4</td></tr>
</table></td></tr>
<tr><td><b>Example</b></td><td>
<table>
<tr><th>Concept</th><th>BC</th><th>NC</th><th>RC</th><th>PreferredTerm</th><th>SimpleNonPreferredTerm</th></tr>
<tr><td>Mineral resources</td><td>Environment</td><td>Fossil fuels</td><td>Mineralogy
Geology</td><td>Mineral resources</td><td>Minerals
Rocks</td></tr>
<tr><td>Fossil fuels</td><td>Mineral resources</td><td></td><td>Mining</td><td>Fossil fuels</td><td>Coal
Gasoline
Petrol</td></tr>
</table></td></tr>
<tr><td colspan="2"><b>OUTPUT: Designed Ontology</b></td></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG^{+}07]. Each thesaurus concept is mapped to a class. A <i>subClassOf</i> relation is defined between the new classes for the BC (Broader Concept)/NC (Narrower Concept) relation. A <i>relatedClass</i> relation is defined between the new classes for the RT relation. A label is created for every non-preferred term of a concept.</td></tr>
<tr><td colspan="2"><b>Graphical Representation</b></td></tr>
</table>

Table 5.3: Pattern for Re-engineering a Thesaurus(continued)

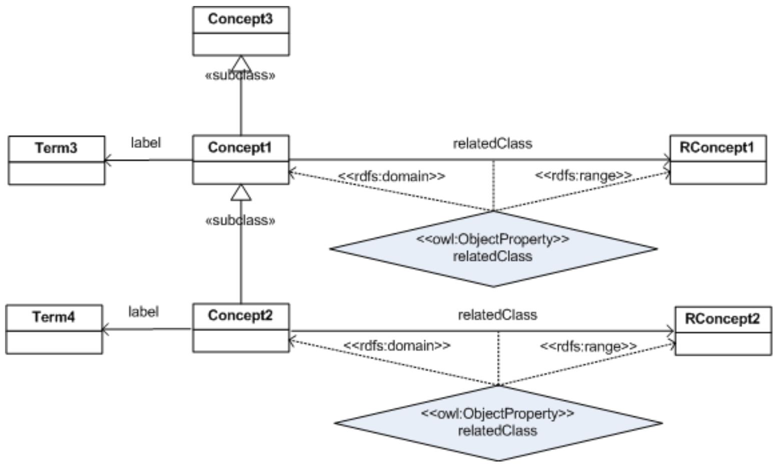
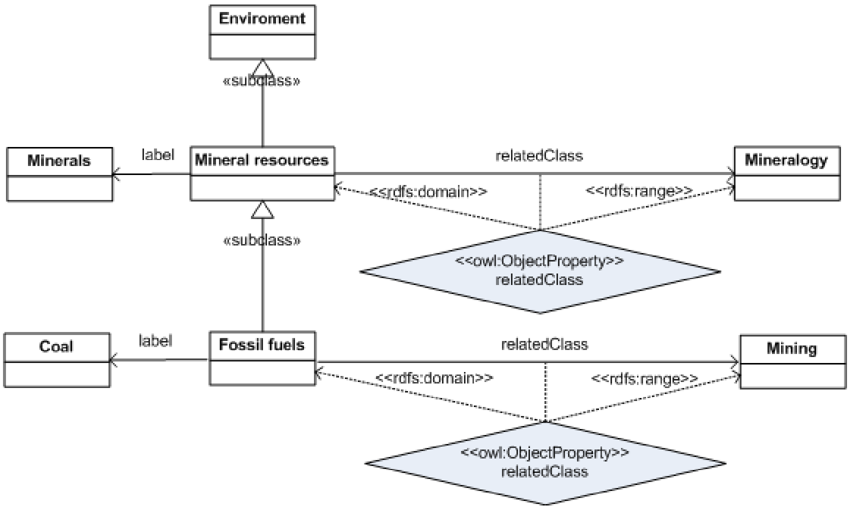
Slot	Value
(UML) General Solution Ontology	 Concept3 «subclass» Term3 label Concept1 relatedClass RConcept1 <<rdfs:domain>> <<rdfs:range>> «subclass» <<owl:ObjectProperty>> relatedClass Term4 label Concept2 relatedClass RConcept2 <<rdfs:domain>> <<rdfs:range>> <<owl:ObjectProperty>> relatedClass
(UML) Example Solution Ontology	 Enviroment «subclass» Minerals label Mineral resources relatedClass Mineralogy <<rdfs:domain>> <<rdfs:range>> «subclass» <<owl:ObjectProperty>> relatedClass Coal label Fossil fuels relatedClass Mining <<rdfs:domain>> <<rdfs:range>> <<owl:ObjectProperty>> relatedClass
PROCESS: How to Re-engineer	
General	1. Identify the records which contain thesaurus concepts without a *broader concept*. 2. For each one of the above identified thesaurus concepts t_i: 2.1. Create the corresponding ontology class, C_i class, if it is not created yet. 2.2. Identify the thesaurus concept, t_j, which are narrower concepts of t_i. They are referenced in the same record which contains t_i. 2.3. For each one of the above identified thesaurus concept t_j: 2.3.1. Create the corresponding ontology class, C_j class, if it is not created yet. 2.3.2. Set up the *subClassOf* relation between C_j and C_i 2.3.3. Repeat from step 2.2 for c_j as a new c_i 2.4. Identify the thesaurus concept, t_r, which are related concepts of t_i. They are referenced in the same record which contains t_i. 2.5. For each one of the above identified thesaurus concept t_r: 2.5.1. Create the corresponding ontology class, C_r class, if it is not created yet. 2.5.2. Set up the *relatedClass* relation between C_r and C_i 2.5.3. Repeat from step 2.4 for c_r as a new c_i 2.6. Identify the non-preferred terms for the concept t_i. They are referenced in the same record which contains t_i. 2.7. For each one of the above identified thesaurus term t_q: 2.7.1. Create the corresponding label l_q for the concept t_i.

Table 5.3: Pattern for Re-engineering a Thesaurus(continued)

Slot	Value
Example	1. Create the `Environment` class. 2. Create the `Mineral resources` class and assert that `Mineral resources` is *subClassOf* `Environment`. 2.1. Create the `Fossil fuels` class and assert that `Fossil fuels` is *subClassOf* `Mineral resources`. 2.2. Create the `Mineralogy` class and assert that `Mineralogy` is *relatedClass* `Mineral resources`. 2.3. Create the `Minerals` label and assert that `Minerals` is *label* of `Mineral resources`. 2.4. Create the `Mining` class and assert that `Mining` is *subClassOf* `Fossil fuels`. 2.5. Create the `Coal` label and assert that `Coal` is *label* of `Fossiel fuels`.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG^{+}07]

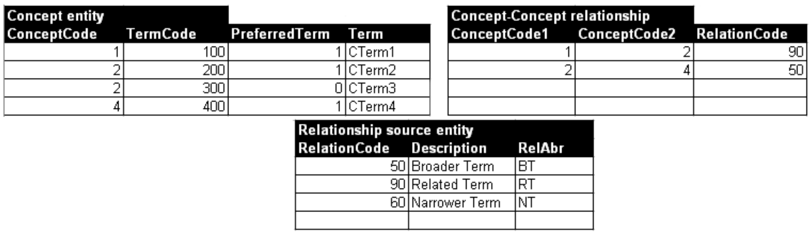
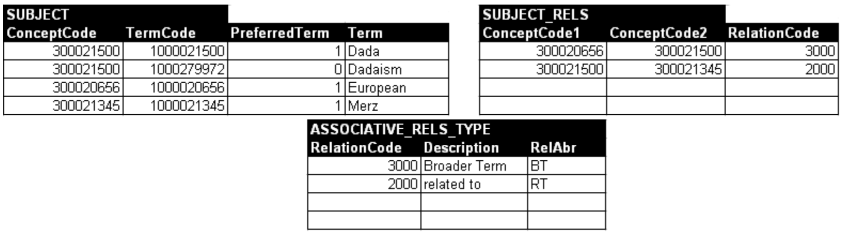
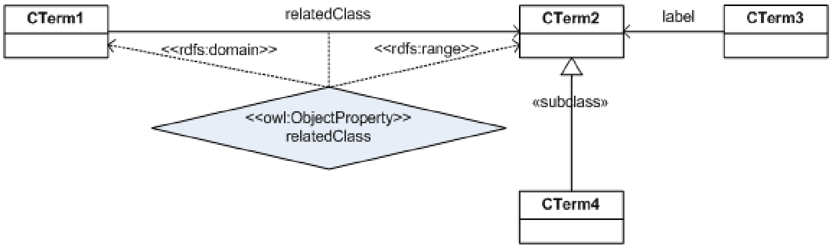
PR-NOR-TSLO-04. Pattern for re-engineering a concept-based thesaurus which follows the relation-based data model

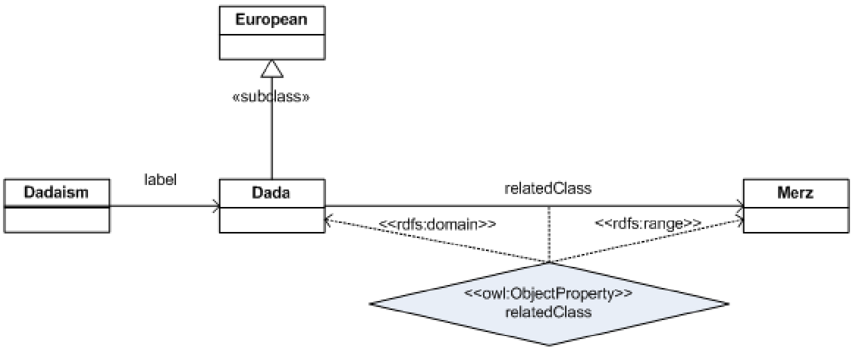
The pattern for re-engineering thesaurus shown in Table 5.4 suggests a guide to transform a thesaurus into a lightweight ontology. The thesaurus is a concept-based one and it is modeled with a relation-based data model. This pattern aims at creating a lightweight ontology from the thesaurus, being the semantics of the BT/NT relations between terms the *subClassOf* relationship.

Table 5.4: Pattern for Re-engineering a concept-based thesaurus which follows the relation-based data model

Slot	Value
General Information	
Name	Concept-based Thesaurus to Lightweight Ontology (relation-based model)
Identifier	PR-NOR-TSLO-04
Type of Component	Pattern for Re-engineering Non-Ontological Resources (PR-NOR)
Use Case	
General	Re-engineering a concept-based thesaurus which follows the relation-based model to design a Lightweight Ontology.
Example	Suppose that someone wants to build a lightweight ontology based on the Art and Architecture Thesaurus (AAT), which is used to describe art, material culture, and archival materials.
Pattern for Re-engineering Non-Ontological Resources	
INPUT: Resource to be Re-engineered	
General	A non-ontological resource holds a concept-based thesaurus which follows the relation-based model. A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them. The relation-based data model [Soe95] is a normalized structure, in which relationship types are not defined as fields in a record, but they are simply data values in a relationship record, thus new relationship types can be introduced with ease.
Example	The Art and Architecture Thesaurus (AAT) is used to describe art, material culture, and archival materials. The relation semantics between the sub-ordinate and the super-ordinate concepts is *subClassOf*.
Graphical Representation	

Table 5.4: Pattern for Re-engineering a Thesaurus(continued)

<table>
<tr><th>Slot</th><th>Value</th></tr>
<tr><td><b>General</b></td><td>

<table>
<tr><th colspan="4">Concept entity</th></tr>
<tr><th>ConceptCode</th><th>TermCode</th><th>PreferredTerm</th><th>Term</th></tr>
<tr><td>1</td><td>100</td><td>1</td><td>CTerm1</td></tr>
<tr><td>2</td><td>200</td><td>1</td><td>CTerm2</td></tr>
<tr><td>2</td><td>300</td><td>0</td><td>CTerm3</td></tr>
<tr><td>4</td><td>400</td><td>1</td><td>CTerm4</td></tr>
</table>
<table>
<tr><th colspan="3">Concept-Concept relationship</th></tr>
<tr><th>ConceptCode1</th><th>ConceptCode2</th><th>RelationCode</th></tr>
<tr><td>1</td><td>2</td><td>90</td></tr>
<tr><td>2</td><td>4</td><td>50</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
<table>
<tr><th colspan="3">Relationship source entity</th></tr>
<tr><th>RelationCode</th><th>Description</th><th>RelAbr</th></tr>
<tr><td>50</td><td>Broader Term</td><td>BT</td></tr>
<tr><td>90</td><td>Related Term</td><td>RT</td></tr>
<tr><td>60</td><td>Narrower Term</td><td>NT</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td><b>Example</b></td><td>

<table>
<tr><th colspan="4">SUBJECT</th></tr>
<tr><th>ConceptCode</th><th>TermCode</th><th>PreferredTerm</th><th>Term</th></tr>
<tr><td>300021500</td><td>1000021500</td><td>1</td><td>Dada</td></tr>
<tr><td>300021500</td><td>1000279972</td><td>0</td><td>Dadaism</td></tr>
<tr><td>300020656</td><td>1000020656</td><td>1</td><td>European</td></tr>
<tr><td>300021345</td><td>1000021345</td><td>1</td><td>Merz</td></tr>
</table>
<table>
<tr><th colspan="3">SUBJECT_RELS</th></tr>
<tr><th>ConceptCode1</th><th>ConceptCode2</th><th>RelationCode</th></tr>
<tr><td>300020656</td><td>300021500</td><td>3000</td></tr>
<tr><td>300021500</td><td>300021345</td><td>2000</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
<table>
<tr><th colspan="3">ASSOCIATIVE_RELS_TYPE</th></tr>
<tr><th>RelationCode</th><th>Description</th><th>RelAbr</th></tr>
<tr><td>3000</td><td>Broader Term</td><td>BT</td></tr>
<tr><td>2000</td><td>related to</td><td>RT</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><th colspan="2">OUTPUT: Designed Ontology</th></tr>
<tr><td><b>General</b></td><td>The generated ontology will be based on the lightweight ontology architectural pattern (AP-LW-01)[SFBG+07]. Each thesaurus concept is mapped to a class. A <i>subClassOf</i> relation is defined between the new classes for the BC (Broader Concept)/NC (Narrower Concept) relation. A <i>relatedClass</i> relation is defined between the new classes for the RT relation. A label is created for every non-preferred term of a concept.</td></tr>
<tr><th colspan="2">Graphical Representation</th></tr>
<tr><td><b>(UML) General Solution Ontology</b></td><td>


</td></tr>
<tr><td><b>(UML) Example Solution Ontology</b></td><td>


</td></tr>
<tr><th colspan="2">PROCESS: How to Re-engineer</th></tr>
</table>

Table 5.4: Pattern for Re-engineering a Thesaurus(continued)

Slot	Value
General	1. Identify the records which contain thesaurus concepts without a *broader concept*, within the concept-concept relationship entity. 2. For each one of the above identified thesaurus concepts t_i: 2.1. Obtain the thesaurus concept t_i within the concept entity. 2.2. Create the corresponding ontology class, C_i class, if it is not created yet. 2.3. Identify the thesaurus concept, t_j, which are narrower concepts of t_i, within the concept-concept relationship entity. 2.4. For each one of the above identified thesaurus concept t_j: 2.4.1. Obtain the thesaurus concept t_j within the concept entity. 2.4.2. Create the corresponding ontology class, C_j class, if it is not created yet. 2.4.3. Set up the *subClassOf* relation between C_j and C_i 2.4.4. Repeat from step 2.2 for c_j as a new c_i 2.5. Identify the thesaurus concept, t_r, which are related concepts of t_i, within the concept-concept relationship entity. 2.6. For each one of the above identified thesaurus concept t_r: 2.6.1. Obtain the thesaurus concept t_r within the concept entity. 2.6.2. Create the corresponding ontology class, C_r class, if it is not created yet. 2.6.3. Set up the *relatedClass* relation between C_r and C_i 2.6.4. Repeat from step 2.4 for c_r as a new c_i 2.7. Identify the non-preferred terms for the concept t_i. They are referenced in the same records which contain t_i, within the concept entity. 2.8. For each one of the above identified thesaurus term t_q: 2.8.1. Create the corresponding label l_q for the concept t_i.
Example	1. Create the `European` class. 2. Create the `Dada` class and assert that `Dada` is *subClassOf* `European`. 2.1. Create the `Merz` class and assert that `Dada` is *relatedClass* of `Dada`. 2.2. Create the `Dadaism` label and assert that `Dadaism` is *label* of `Dada`.
Relationships	
Relations to other modelling components	Use the Architectural Pattern: AP-LW-01 [SFBG+07]

5.6 NeOn Method for Re-engineering Thesauri

The goal of the thesaurus re-engineering process is to transform a thesaurus into an ontology. The activities of this process are based on the ones presented in 3.1.4: (1) non-ontological resource reverse engineering, which includes tasks 1 to 3, (2) non-ontological resource transformation, which includes tasks 4 and 5, and (3) ontology forward engineering, which includes tasks 6 and 7.

5.6.1 Thesaurus Transformation

In the following, we outline the specialization of the non-ontological resource transformation activity for thesauri, and consequently how to carry out tasks 4 and 5 included in Figure 3.2. The thesaurus transformation consists of the following tasks:

- **Task 4. Search for a suitable pattern for re-engineering thesaurus.** The goal of this task is to find out if there is any applicable re-engineering pattern for re-engineering thesaurus useful to transform the thesaurus into a conceptual model. The search for a suitable pattern for re-engineering thesaurus should be done into the NeOn library of patterns[2]. For this search, we have to define the following search criteria:

- Type of non-ontological resource: thesaurus.
- Type of thesaurus: term-based thesaurus or concept-based thesaurus.
- Data model: record-based, or relation-based.
- Target ontology: lightweight ontology.
- The semantics of the BT/NT relations between thesaurus terms: *subClassOf*, *parOf*, or other *ad-hoc* relation.

- **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 thesaurus into a conceptual model. If a suitable pattern for re-engineering thesaurus is found then the conceptual model is created from the thesaurus following the procedure established in the re-engineering pattern.

- **Task 5.b. Perform and ad-hoc transformation.** The goal of this task is to set up an *ad-hoc* procedure to transform the thesaurus 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 thesaurus.

- **Task 5.c. Manual refinement.** We add this task for dealing with thesauri transformation. Within this task, software developers and ontology practitioners with the domain experts support can perform the following refinements:

  - Some *related term* relations can be transformed into more specific kinds of relations.
  - If there will be the case where some *BT* ambiguous relations are present. These *BT* relations can be disambiguated. They may mean either *subClassOf*, *partOf* or *instanceOf*.

[2] http://www.ontologydesignpatterns.org

Chapter 6

Method and tool for re-engineering folksonomies

The re-engineering of resources on a specific domain aims at lowering the cost and effort of creating domain ontologies from scratch. The goal of this work is to create an ontological structure for a folksonomy of a specific domain by utilising automatically selected knowledge from online available ontologies.

In the following we describe the process of transforming a folksonomy tagset over a specific domain into an ontology covering this domain. Our folksonomy semantic enrichment tool, FLOR, described in Section 6.1, can operate on any tagspace independently thus the selection of folksonomy tagspaces depends on the domain of interest. For example many organisations have introduced folksonomies as part of their content management procedures. The tags of these close-world corporate folksonomies are provided by the employees of these organisations thus describe relevant domain concepts. The re-engineering of these corporate folksonomies can result to corporate ontologies for the related organisation.

Alternatively, in cases where the domain of interest is more generic or independent of specific organisations e.g., Flora or Fauna, the large scale open folksonomies freely available on the web i.e. Flickr or delicious provide social network information such as user groups of interest on specific domains such as Flowers[1] or Animals[2]. The extraction of these groups' tags provide a domain related folksonomy tagspace to re-engineer and acquire the relevant ontology.

6.1 Semantic Enrichment of Tags with FLOR

We introduce FLOR, a tool for automatic folksonomy enrichment by combining knowledge from WordNet and online ontologies. The goal of FLOR is to transform a flat folksonomy tagspace into a rich semantic representation by assigning relevant Semantic Web Entities (SWEs) to each tag. A SWE is an ontological entity (class, relation, instance) defined in an online available ontology. While here we describe the process of enriching a set of tags with SWEs, the ultimate goal of our FLOR is not just to connect to SWE's but also to bring in other knowledge related to these SWE's. An example of the inputs and expected outcomes of FLOR is demonstrated in Fig. 6.1. The input consists a set of tags and the output is a set of semantically enriched tags, connected with each other to an ontological structure. Note that FLOR is agnostic to the way in which this tagset was obtained. It can either be the set of all tags associated to a domain specific resource, or a cluster of related tags obtained through co-occurrence based clustering over the total domain tagspace. The experiments reported in this work used sets of tags associated with a given resource.

Intuitively, FLOR performs three basic steps (see Fig. 6.1 and 6.2). First, during the **Lexical Processing** the input tagset is cleaned and all potentially meaningless tags are excluded. We rely on a set of heuristics to decide which tags are likely to be meaningless. Second, during the **Sense Definition and Semantic**

[1] http://www.flickr.com/groups/florus/

[2] http://www.flickr.com/groups/animal_planet/

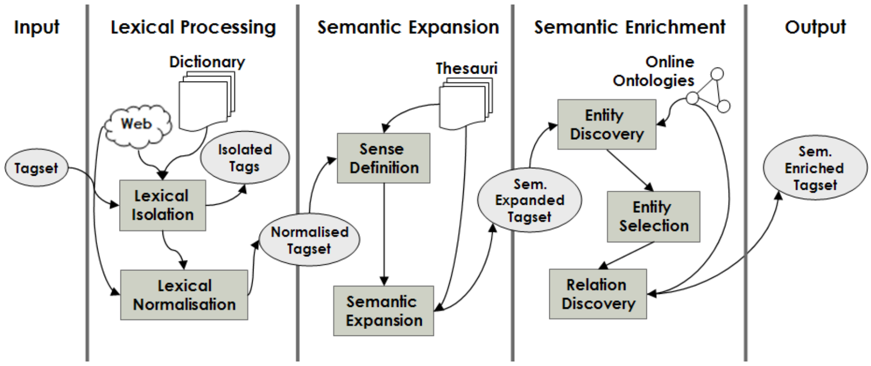


Figure 6.1: FLOR Phases

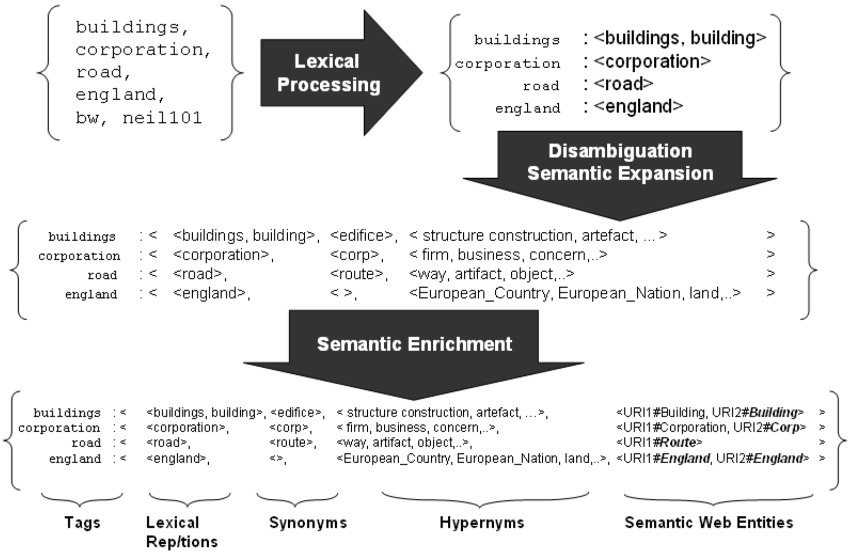

buildings, corporation, road, england, bw, neil101
Lexical Processing
buildings : <buildings, building>
corporation : <corporation>
road : <road>
england : <england>
Disambiguation Semantic Expansion
buildings : < <buildings, building>, <edifice>, < structure construction, artefact, ... > >
corporation : < <corporation>, <corp>, < firm, business, concern,..> >
road : < <road>, <route>, <way, artifact, object,..> >
england : < <england>, < >, <European_Country, European_Nation, land,..> >
Semantic Enrichment
buildings : < <buildings, building>, <edifice>, < structure construction, artefact, ...>, <URI1#Building, URI2#Building> >
corporation : < <corporation>, <corp>, < firm, business, concern,..>, <URI1#Corporation, URI2#Corp> >
road : < <road>, <route>, <way, artifact, object,..>, <URI1#Route> >
england : < <england>, <>, <European_Country, European_Nation, land,..>, <URI1#England, URI2#England> >
Tags
Lexical Rep/tions
Synonyms
Hypernyms
Semantic Web Entities

Figure 6.2: FLOR Steps Example

Expansion we attempt to assign a WordNet sense to each tag based on its context (i.e., the other tags in its cluster) and to extract all relevant synonyms and hypernyms so that we migrate to a richer representation of the tag. Finally, during the **Semantic Enrichment** step each tag is associated to the appropriate SWE.

The first step results in the **Lexical Representations** which is a list of lexical forms for the tag, such as plural and singular forms for nouns, or various delimited types of compound tags (sanFrancisco, san.Francisco, etc). The second step identifies **Synonyms** and **Hypernyms** for each tag. The last step generates the list of **Entities** containing the associated SWE's. Note that a tag can be associated to several relevant SWE's.

6.1.1 STEP 1: Lexical Processing

Due to the freedom of tagging as a basic rule of folksonomies, a wide variety of different tag types are in use. Understanding the types of tags used is the first step in deciding which of them are meaningful and should be taken into account as a basis of a semantic enrichment process. Previous work ([ASSM07, GH06, MDA07b]) has identified different conceptual categories of tags (event, location, person), as well as tag categories that can be described by syntactic characteristics. For example, there are many tags containing special characters (e.g., `:P`), numbers (e.g., `aug07`), plurals as well as singular forms of the same word (e.g., `building`, `buildings`), concatenated tags (e.g., `littlegirl`) or tags with spaces (e.g., `little girl`) and a big number of non-English tags (e.g., `sillon`). The role of the lexical processing step is to identify these different categories of tags and exclude those that are meaningless and should not be further included in the semantic enrichment process. This is done in two sub-steps.

SUB-STEP 1.1: The Lexical Isolation

This sub-step identifies sets of tags that should be excluded as well as those that can be further processed. Currently we isolate and exclude all tags with numbers, special characters and non English tags. The reason for excluding non-English tags is that our method explores various external knowledge sources (WordNet, Semantic Web ontologies) that are primarily in English. As future work, we will extend FLOR to isolate additional types of tags as well and deal with non-English tags.

SUB-STEP 1.2: The Lexical Normalisation

This sub-step aims to solve the incompatibility between different naming conventions used in folksonomies, ontologies and other lexical databases, such as WordNet. This phase produces a list of possible **Lexical Representations** for each tag aiming to maximise the coverage of this tag by different resources. For example, the compound tag `santabarbara` in folksonomies appears as *Santa-Barbara* or *Santa+Barbara* in various ontologies and as ***Santa Barbara*** in WordNet. However, as the lexical anchoring to these resources is a quite complex problem, we try to address it by producing all the possible lexical representations for each tag such as: {santaBarbara, santa.barbara, santa_barbara, santa barbara, santa-barbara, santa+barbara, ...}.

6.1.2 STEP 2: Sense Definition and Semantic Expansion

Due to polysemy, the same tag can have different meanings in different contexts. For example, the tag `jaguar` can describe either a car or an animal depending on the context in which it appears. Before connecting a tag with a relevant SWE, it is important to determine its intended sense in the given context. This task is performed in the first sub-step, which is the **Sense Definition and Disambiguation** sub-step.

Another issue to take into account is that, despite its significant growth, the Semantic Web is still sparse. A direct implication is that while online ontologies might not contain concepts that are syntactically equivalent to a given tag, they might contain concepts that are labeled with one of its synonyms. To overcome this limitation, we perform a semantic expansion for each tag, based on its previously identified sense, in the second sub-step.

SUB-STEP 2.1: The Sense Definition and Disambiguation

This sub-step discovers the intended sense of a tag in the context it appears. As context we consider the set of tags with which the given tag co-occurs when describing a resource. For example, in the tagset: {`panther`, `jaguar`, `jungle`, `wild`} the context of `jaguar` is {`panther`, `jungle`, `wild`}. We use WordNet as a sense repository and rely on its hierarchy of senses to compute the similarities between the senses of all tags in the tagset and thus achieve their disambiguation. WordNet also provides rich sense definitions which facilitate the semantic expansion in the next sub-step.

To define the senses of the tags in a tagset, we identify all the lexical representations for each tag in WordNet. In the cases that a tag has more than one senses in WordNet (synsets) we exploit the contextual information of the tagset to identify the most relevant sense. For this, we calculate the similarity between all the combinations of tags in the tagset using the Wu and Palmer similarity formula ([WP94]) on the WordNet graph. The similarity degree between two senses is calculated based on the number of common ancestors between them in the WordNet hierarchy and the length of their connecting path. The result for each calculation is a couple of senses and a similarity degree for these senses. We select the two senses of the tags that return the highest similarity degree provided that this is higher than a specified threshold. If a tag has low similarities when compared to all the other tags in its cluster, then it is assigned to the most popular WordNet sense.

We currently use a threshold value of 0.8 which we observed to correctly indicate relatedness in most of the cases. Indeed, as high values as 0.7 are often assigned to unrelated tags. For example, in the tagset: {`girl`, `eating`, `red`, `apple`} the similarity between `red` and `girl` is 0.7 for the senses:

Bolshevik, Marxist, Pinko, Red, Bolshie (emotionally charged terms used to refer to extreme radicals or revolutionaries)

Girlfriend, Girl, Lady_friend (a girl or young woman with whom a man is romantically involved)

These two senses are connected through the concept ***Person*** in the WordNet hierarchy, however the two tags are unrelated in the context of this tag cluster. While this empirically established 0.8 value lead to reasonable results and was sufficient for this proof of concept prototype, we plan to establish an optimal value through systematic experiments.

Thanks to the modular architecture of FLOR, the disambiguation and sense selection method can be replaced by other methods (e.g., such as those used in [TGEM07] and [YGS07]). Or our current method could be modified to exploit a different similarity measure between two concepts such as the Google Similarity Distance [CV07].

Another possible improvement could be achieved by further expanding the resource tagset with more related tags. These can be discovered with statistical measures based on tag co-occurrence as described in [SM07]. For example, the expanded tagset of {`apple`, `mac`} could be {`apple`, `mac`, `computer`, `macOs`}. So instead of trying to disambiguate with two tags we increase the possibilities of finding the correct sense by disambiguating with a more specific context.

SUB-STEP 2.2: The Semantic Expansion

expands the tag with its synonyms and hypernyms (see Fig. 6.2). For the purpose of this work we used WordNet to extract the synonyms of the correct sense and the synonyms of this sense's hypernym in WordNet. For example, if in the specific context the tag `jaguar` refers to an animal then the semantic expansion would include a list of synonyms: {***Panther***, ***Panthera onca***, ***Felis onca***} and a list of hypernyms: {***Big cat***, ***Feline***, ***Carnivore***}.

6.1.3 STEP 3: Semantic Enrichment

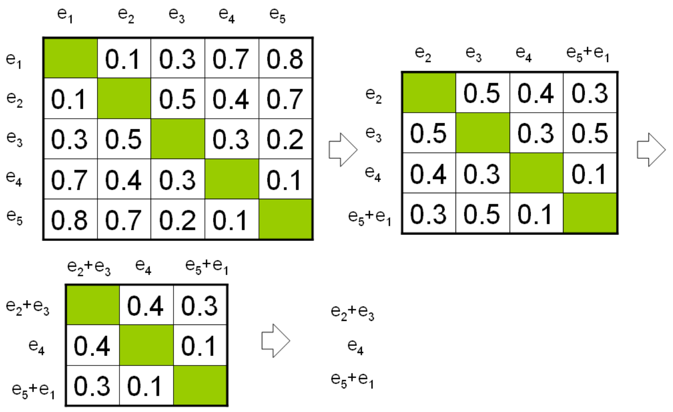
The final phase of FLOR identifies the SWEs that are relevant for each tag by leveraging the results of lexical cleaning and semantic expansion performed in the previous two phases. The final output of FLOR is produced by this phase (see Fig. 6.1) and it is an enriched tagset with relevant SWEs and their semantic neighbourhood (e.g., parents, children, relations).

SUB-STEP 3.1 and 3.2: Entity Discovery and Selection

The relevant SWEs are discovered by querying the WATSON semantic web gateway [M. 07], which gives access to all online ontologies. We search for all ontological entities (Classes, Properties, Individuals) that contain in their local name or in their label(s) one of the lexical representations or the synonyms of a tag.

Such queries often result in several SWEs some of which are very similar (or the same when they appear in ontologies that are versions of each other). To reduce the number of SWEs, we perform an entity integration process similar to the one described in [TGEM07]. The goal of this process is to "collapse" entities that have a high similarity into a single semantic object, thus reducing redundancy. To compute similarity between two entities we compare their semantic neighbourhoods (superclasses, subclasses, disjoint classes for classes; domain, range, superproperties, subproperties for properties) and their localnames and labels. The similarity $simDgr$ for two SWEs e_1 and e_2 is calculated as:

$$simDgr = W_l * simLexical(e_1, e_2) + W_g * simGraph(e_1, e_2)$$



	e_1	e_2	e_3	e_4	e_5
e_1	🟩	0.1	0.3	0.7	0.8
e_2	0.1	🟩	0.5	0.4	0.7
e_3	0.3	0.5	🟩	0.3	0.2
e_4	0.7	0.4	0.3	🟩	0.1
e_5	0.8	0.7	0.2	0.1	🟩

	e_2	e_3	e_4	e_5+e_1
e_2	🟩	0.5	0.4	0.3
e_3	0.5	🟩	0.3	0.5
e_4	0.4	0.3	🟩	0.1
e_5+e_1	0.3	0.5	0.1	🟩

	e_2+e_3	e_4	e_5+e_1
e_2+e_3	🟩	0.4	0.3
e_4	0.4	🟩	0.1
e_5+e_1	0.3	0.1	🟩

e_2+e_3
e_4
e_5+e_1

Figure 6.3: Merging Strategy with threshold 0.5

$simLexical(e_1, e_2)$ is the similarity between the lexical information of two entities, i.e., their labels and localnames, computed with the Levenshtein distance metric. $simGraph(e_1, e_2)$ is the similarity of the entities' neighbourhoods, where the similarity of each neighbourhood element is computed based on string similarity. Because we consider the similarity of the semantic neighbourhoods more important than the similarity of the labels, we set the weights as $W_l = 0.3$ and $W_g = 0.7$. Note that these weights will be fine-tuned through systematic experiments.

If the similarity between two entities is higher than a threshold we merge them in one entity by integrating their neighbourhoods into one. Then we repeat the process until all entities are sufficiently different from each other, i.e., their similarity falls under a chosen threshold.

Consider for example Fig. 6.3 where five SWEs $e_{1,5}$ are compared against a threshold value of 0.5. We start by performing their pair-wise comparison and observe that the pairs (e_1, e_4), (e_1, e_5), (e_2, e_3) and (e_2, e_5) have a similarity equal or above the set threshold. We proceed by merging the first two entities with the highest similarity, e_1 and e_5, to one entity e_1+e_5 and compute the similarities between the new entity and the remaining ones. This process continues until all similarities are lower than the set threshold, which implies that the obtained entities are sufficiently different.

Once the merged entities are created we enrich the tag with the relevant entities. This is done by comparing the ontological parents of the merged entity with the hypernyms retrieved from WordNet. The ontological parents are the superclasses of classes, the superproperties of properties and the classes of individuals. For example, as shown in Fig. 6.4, the tag `moon` is enriched with two entities. The superclasses of both the entities have as localname one of the hypernyms extracted from the WordNet sense of `moon`. Also, apart from the semantic definition of the tag with the respective entity, we further enrich the tag with the information carried by the entity, *EarthsMoon* `TypeOf` *Moon*.

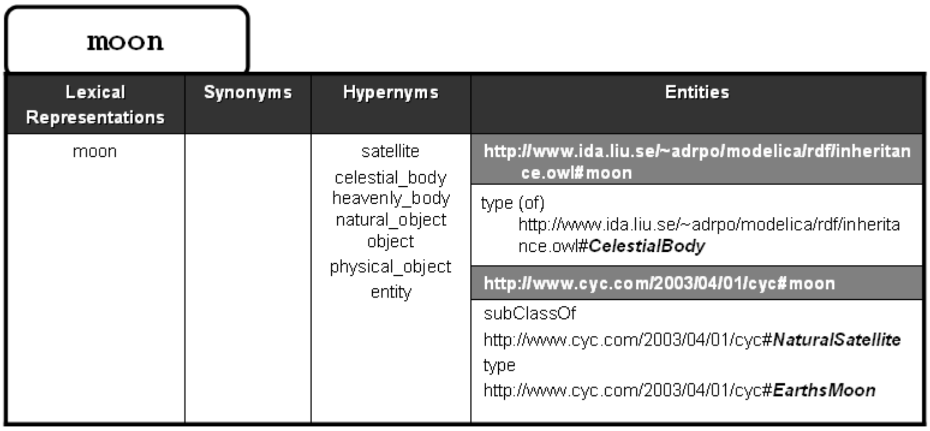
As depicted in Fig. 6.1 Phase 3 of FLOR includes also the Relation Discovery Step. This is done by the SCARLET [3] Relation Discovery tool. SCARLET is able to take as input any two SWEs and identify the relations between them by searching in online ontologies. At the moment SCARLET and FLOR haven't been integrated yet in order to provide an interconnected semantically enriched tagset as a proper ontological structure. This is part of our future work. Currently the output of FLOR is a tagset enriched with a set of SWEs describing it as depicted in Fig. 6.1.

6.1.4 Example: FLOR Enrichment

In this section we present a full cycle of the FLOR semantic enrichment method for the tag `lake`, which was found in the following five tagsets:

- {`rush`, `lake`, `pakistan`, `rakaposhi`, `mountain`, `asia`, `kashmir`, `snow`, `glacier`, `green`,

[3] `http://scarlet.open.ac.uk/`



moon

Lexical Representations	Synonyms	Hypernyms	Entities
moon		satellite celestial_body heavenly_body natural_object object physical_object entity	**http://www.ida.liu.se/~adrpo/modelica/rdf/inheritance.owl#moon** type (of) http://www.ida.liu.se/~adrpo/modelica/rdf/inheritance.owl#***CelestialBody*** **http://www.cyc.com/2003/04/01/cyc#moon** subClassOf http://www.cyc.com/2003/04/01/cyc#***NaturalSatellite*** type http://www.cyc.com/2003/04/01/cyc#***EarthsMoon***

Figure 6.4: Enriched FlorTag `moon`

`white`, `sky`, `blue`, `clouds`, `water`},

- {`moraine`, `alberta`, `banff`, `canada`, `lake`, `lac`, `rockies`, `scan`},
- {`rising`, `sunlight`, `lake`, `quality`, `bravo`}, {`lake`, `nature`, `landscape`, `sunset`, `water`, `organisms`}
- {`lake`, `finland`, `suomi`, `beach`, `bubbles`, `blue`, `sunlight`, `kids`, `natural`}

Note that these tagsets contain the tags that remain after the lexical processing performed in the first phase. Fig. 6.5 shows the information contained in the automatically obtained FlorTag.

For the second phase of FLOR, Sense Definition and Semantic Expansion using WordNet, the available WordNet senses for ***Lake*** are considered. These are the following:

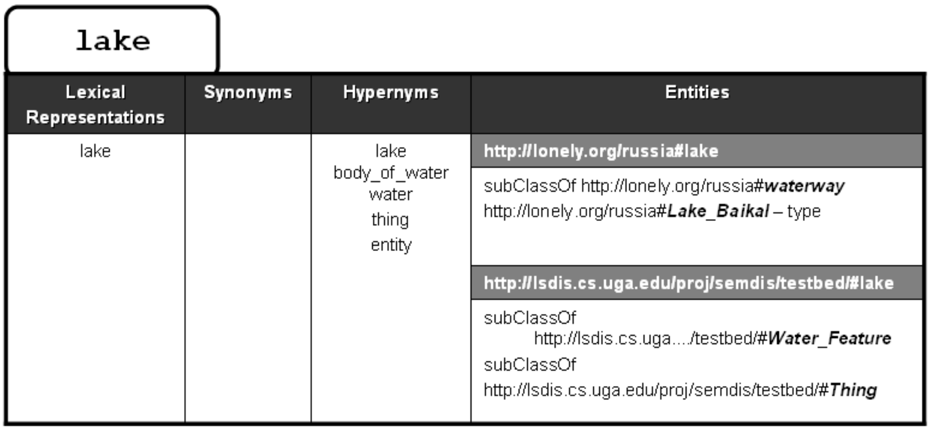
WordNet 1: ***Lake→Body of water***, ***Water→Thing→Entity***
(a body of (usually fresh) water surrounded by land)

WordNet 2: ***Lake→Pigment→Coloring material→Material***
→ Substance→Entity
(a purplish red pigment prepared from lac or cochineal)

WordNet 3: ***Lake→Pigment→Coloring material→Material***
→Substance→Entity
(any of numerous bright translucent organic pigments)

Applying the Wu and Palmer formula for the senses of `lake` and the senses of the rest of the tags in these tagsets we obtained variable similarities from 0 to 0.86. The zero similarities were obtained for location names such as `banf`, `pakistan`, `suomi` and for generally unrelated tags such as `quality`, `scan`, `sunlight`, `sunset`. Interestingly, `lake` returned zero similarity for the tags `glacier` and `mountain` while they should be related. This is due to the fact that, in WordNet, ***Glacier*** and ***Mountain*** are hyponyms of ***Geological formation*** which is a hyponym of ***Natural object*** while ***Lake*** is a hyponym of ***Body of water*** which is a direct hyponym of ***Thing***. Furthermore ***Glacier*** is a hyponym of ***Ice mass*** but there is no subsumption relation between ***Ice mass*** and ***Ice*** or ***Water*** that would allow for a connecting path between ***Lake*** and ***Glacier***. This fact motivates further research on how to identify similarities between tags of a tagset beyond the subsumption relations provided by WordNet.

The highest similarity, 0.86, for `lake` was obtained with the tag `water`, because Sense 1 of ***Lake*** is related to ***Body of water*** (Sense 2 of ***Water***) with a direct hyponymy relation. Note that, in most of tagsets the first sense of ***Water***, ***Liquid***, is selected as this is the most common sense in which the tag is used. Therefore, this is a nice example of phase 2 identifying a non-trivial correlation.



lake

Lexical Representations	Synonyms	Hypernyms	Entities
lake		lake body_of_water water thing entity	**http://lonely.org/russia#lake** subClassOf http://lonely.org/russia#***waterway*** http://lonely.org/russia#***Lake_Baikal*** – type **http://lsdis.cs.uga.edu/proj/semdis/testbed/#lake** subClassOf http://lsdis.cs.uga..../testbed/#***Water_Feature*** subClassOf http://lsdis.cs.uga.edu/proj/semdis/testbed/#***Thing***

Figure 6.5: Enriched FlorTag `lake`

Sense 1. ***Water***, ***H2O***: (binary compound that occurs at room temperature as a clear colorless odorless tasteless liquid) → ***Binary Compound*** AND → ***Liquid***

Sense 2. ***Body of water***, ***Water***: (the part of the earth's surface covered with water) → ***Thing***

Once the correct sense is selected and the tag is semantically expanded with hypernyms (there are no synonyms for this sense of ***Lake***) then the third phase of FLOR queries the online ontologies through WATSON and selects the SWEs that correspond to this sense. As shown in Fig. 6.5 both selected entities have the term *Lake* in their localname and their superclass in the ontology contains one or more of the hypernyms returned by WordNet, *Water* and *Thing*, as a whole or as a compound. This example shows that our anchoring to ontologies is strict for the tags to be defined (their lexical representations and synonyms) and the localnames and labels of the entities and flexible for the ontological parents and hypernyms. Note also that the selected SWEs carry additional information about two superclasses of *Lake* (*Waterway*, *Waterfeature*) and an instance of *Lake* (*Lake Baikal*) thus further enriching the tag.

6.1.5 Experiments: Applying FLOR on a Flickr dataset

To assess the correctness of FLOR enrichment (i.e., whether tags were linked to relevant SWEs) we applied FLOR on a Flickr data set comprised of 250 randomly selected photos with a total of 2819 individual tags. During the Lexical Isolation we removed 59% of the initial tags resulting to 1146 tags in total. We isolated 45 tags with two characters (e.g., `pb`, `ak`), 333 tags with numbers (e.g., `356days`, `tag1`), 86 tags with special characters (e.g., `:P`, `(raw → jpg)`), and 818 non English tags (e.g., `turdus`, `arbol`). Then we filtered out the photos that exclusively contained the isolated tags (24 photos) and obtained a dataset of 226 photos with a total of 1146 tags. After running the FLOR enrichment algorithm for these 226 photos, one of the authors manually checked all the assignments between tags and SWE's.

The assignment of a SWE to a tag is considered correct if the concept described by the SWE is the same as the concept of the tag in the context of its tagset. To decide that the evaluator was given a tagset and the SWEs linked to its tags. She evaluated each tag enrichment as CORRECT if the tag was linked to the appropriate SWE and INCORRECT otherwise. In cases when she was not sure about the intended meaning of the tag, she rated the enrichment as UNDETERMINED. Finally, a NON ENRICHED value was assigned to tags that were not associated to any SWE. The results are displayed in Table 6.1.

Out of the individual 1146 lexically processed tags, FLOR correctly enriched 281 tags and incorrectly enriched 20 tags thus leading to precision results of 93%. An example of incorrect enrichment is that of `square` in the context {`street`, `square`, `film`, `color`, `documentary`}. While its intended meaning is ***Geographical area***, because during the disambiguation phase `square` did not return high similarity

Enrichment Result	# of Tags	Percentage
CORRECT	281	24.5%
INCORRECT	20	1.7%
UNDETERMINED	4	0.3%
NON ENRICHED	841	73.4%
Total	1146	100%

Table 6.1: Evaluation of semantic enrichment for individual tags.

with any of the rest of the tags, the WordNet sense assigned to it was the most popular one, ***Geometrical shape***. This lead to the assignment of non-relevant SWE's namely, *Square* `SubClassOf` *Rectangle* and *Square* `SubClassOf` *RegularPolygonShaped*. Despite this error, the rest of the tags in this tagset were correctly enriched.

FLOR failed to enrich 841 tags, i.e., 73.4% of the tags (see Table 6.1). Because this is a significant amount of tags, we wished to understand whether the enrichment failed because of FLOR's recall or because most of the tags have no equivalent coverage in online ontologies. To that end we selected a random 10% of the 841 tags (85 tags) and manually identified appropriate SWE(s) using WATSON and taking into account the context(s) of the tags in the tagset(s) they appear. Out of the 85 tags we manually enriched 29. We therefore estimate that the number of tags that could have been enriched by FLOR (i.e., those for which an appropriate SWE exists) is approximately 287. Thus, taking into account that the overall number of tags that should be correctly enriched was 568 (281+287) but only 281 were enriched by FLOR this leads to an approximate recall rate of 49%. While this is quite a low recall, these results are highly superior to the ones we have obtained in previous experiments where phase 2 was not part of FLOR, i.e., we directly searched for SWEs for the tags without relying on WordNet as an intermediary step. Indeed, the WordNet sense definition and expansion of the tags with synonyms and hypernyms (FLOR phase 2) increased the tag discovery in the Semantic Web thus having a positive effect on recall.

FLOR failed to enrich the above 29 tags due to the following reasons. The majority of the failures (55%) was due to **different definition** in terms of superclasses in WordNet and in online ontologies For example, the definition of `love` in WordNet and the relevant entity found in the Semantic Web are:

WordNet: ***Love→Emotion→Feeling→Psychological feature***
(a strong positive emotion of regard and affection)

Semantic Web: *Love* `SubClassOf` *Affection*

Although both these definitions refer to the same sense, and additionally the superclass *Affection* belongs to the gloss of ***Love*** in WordNet, they were not matched because *Affection* does not appear as a hypernym of *Love*. Current work investigates alternative ways of Semantic Expansion.

A further 24% of the tags not connected to any SWE were assigned to the **wrong sense** during phase 2. For example, `bulb` referring to `light bulb` in its tagset is assigned the incorrect sense ***Bulb*** → ***Stalk*** → ***Stem*** → ***Plant organ***. The rest of the unenriched tags are due to failures in anchoring them into appropriate SWE's. For example, the sense of `butterfly` was correctly identified, but non of its lexical forms matched the label of the appropriate SWE (*Butterfly_Insect*):

WordNet: ***Butterfly→Lepidopterous insect*** → ***Lepidopteron*** → ***Lepidopteran*** → ***Insect***

Semantic Web: Identified entity with localname *Butterfly_Insect*

In the case of 4 tags the evaluator could not determine whether the enrichment was correct or incorrect (Table 6.1). This is because the meaning of the tag was unclear even when considering its context and the actual photo.

Enrichment Result	**# of Photos**	**Percentage**
CORRECT	179	79.2%
INCORRECT	3	1.3%
MIXED	17	7.5%
UNDETERMINED	4	1.8%
NON ENRICHED	23	10.2%
Total	226	100%

Table 6.2: Evaluation of SWE assignment to photos.

After evaluating the individual tag enrichments the evaluator was able to draw conclusions on the overall enrichment of the tagset i.e., by photo. The evaluation output is displayed in Table 6.2. This would result to {CORRECT, INCORRECT, MIXED, UNDETERMINED, NON ENRICHED}. According to this table, 179 enrichments (about 80%) were {CORRECT}, i.e., all the enriched tags of the photo are enriched correctly. Note that the {CORRECT} enrichment results are much higher from a photo-centric perspective as many tags may appear in many photos. For the total of 20 {INCORRECT} and {MIXED} enrichments, 3 of the photos had all enriched tags incorrect and 17 had at least one tag incorrectly enriched. Finally the above 4 {UNDETERMINED} tags resulted to 4 {UNDETERMINED} enrichments. Finally if no enriched tag appears in the photo then the result for the photo is {NON ENRICHED}.

Chapter 7

Methods and tools for extracting entities from unstructured text

7.1 Introduction

Named entity recognition (NE) from textual sources has been the subject of research for the last two decades, and is crucial for many Natural Language Processing (NLP) tasks as well as other applications. NE is the cornerstone of tasks such as information extraction, which rely on the initial extraction of entities before identifying relations, co-reference etc. Traditional methods and tools for NE have been described and discussed widely in the past, so we shall not reiterate here – the interested reader can see for example [MLP08]. Both rule-based methods such as [MBC, MTB$^+$03, AM99, Chi98] and statistical methods such as [BON03, Col02, IK02] have proved very successful. For a closed and concise set of named entities, both approaches are much easier, not only because ambiguity is less and the task is more constrained, but also because for rule-based approaches, only a relatively small number of rules needs to be written, while for machine learning approaches, it is relatively easy to create sufficient annotated data for training.

In recent years, research has turned towards a more fine grained classification of named entities, generally in the form of Ontology-based Information Extraction (OBIE), where the task requires annotating text on the basis of a much wider ranger of concepts, according to an ontology. So instead of just identifying entitled that fall into the category "Organisation", the system must distinguish between different types of Organisation such as Religious Organisation, Charity, Company, Financial Institution, and so on. Early work on this began with an attempt simply to subcategorise the standard named entities further [FH02] or to involve a more semantically-based decision process [MBC03], and progressed towards a full ontology-based approach - either with standard types of named entities [PEK03] or in a more confined scenario [MYKK05, MSY$^+$07].

The work described here combines aspects from both traditional named entity recognition and ontology-based information extraction, in order to identify patterns for the extraction of a variety of entity types and relations between them, and to re-engineer them into concepts and instances via ontology creation and population. The entities extracted include both traditional named entities such as Person, Location and Organisation, and single or multi-word terms such as "Pacific flounder" and "shark". In the following chapter, we investigate the identification of patterns in text for entity and relation extraction. We then look at the process of ontology population and creation using a modified version of the CLONE technology in GATE [CMBT02]. The system is embodied in an application called SPRAT (Semantic Pattern Recognition and Annotation Tool) which takes as input a set of texts and optionally an existing ontology, and outputs an annotated version of the texts and either a new ontology or a modified version of the existing ontology, as appropriate. SPRAT will be available in the Neon Toolkit as part of the SAFE web services plugin.

7.1.1 NE Recognition with GATE and ANNIE

GATE, the General Architecture for Text Engineering, is a framework providing support for a variety of language engineering tasks. It includes a vanilla information extraction system, ANNIE, and a large number of plugins for various tasks and applications, such as ontology support, information retrieval, support for different languages, WordNet, machine learning algorithms, and so on. There are many publications about GATE and ANNIE – see for example [MTC$^+$02]. We summarise briefly below the components and method used for rule-based information extraction in GATE.

ANNIE consists of the following set of processing resources: tokeniser, sentence splitter, POS tagger, gazetteer, finite state transduction grammar and orthomatcher. The resources communicate via GATE's annotation API, which is a directed graph of arcs bearing arbitrary feature/value data, and nodes rooting this data into document content (in this case text).

The **tokeniser** 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, which adds a part-of-speech 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, organisations, 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 [CMB$^+$02], 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 tokeniser, gazetteer, document format analysis, etc.).

The **orthomatcher** performs coreference, or entity tracking, by recognising 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.

ANNIE has been adapted to many different uses and applications: see [MC03, MTBC03, May03] for some examples. 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 tokeniser, sentence splitter, POS tagger and orthographic coreference modules fall into the former category, while resources such as gazetteers and JAPE grammars will need to be modified according to the application. Similarly, some resources, such as the tokeniser 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 [PMC$^+$02]; 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.

7.1.2 Ontology population

Ontology population is a crucial part of knowledge base construction and maintenance that enables us to relate text to ontologies, providing on the one hand a customised ontology related to the data and domain with which we are concerned, and on the other hand a richer ontology which can be used for a variety of semantic web-related tasks such as knowledge management, information retrieval, question answering, semantic desktop applications, and so on.

Ontology population is generally performed by means of some kind of ontology-based information extraction (OBIE). This consists of identifying the key terms in the text (such as named entities and technical terms) and then relating them to concepts in the ontology. Typically, the core information extraction is carried out by linguistic pre-processing (tokenisation, POS tagging etc.), followed by a named entity recognition component, such as a gazetteer and rule-based grammar or machine learning techniques. Named entity recognition (using such approaches) and automatic term recognition are generally performed in a mutually exclusive way: i.e. one or other technique is used depending on the ultimate goal. However, it makes sense to use a combination of the two techniques in order to maximise the benefits of both. For example, term extraction generally makes use of frequency-based information, whereas typically named entity recognition uses a more linguistic basis. Note also that a "term" refers to a specific concept characteristic of a domain, so while a named entity such as Person or Location is generic across all domains, a technical term such as "myocardial infarction" is only considered a relevant term when it occurs in a medical domain: if we were interested in sporting terms then it would probably not be considered a relevant term, even if it occurred in a sports article. As with named entities, however, terms are generally formed from noun phrases (though in some contexts, verbs may also be considered as terms).

7.2 Patterns for entity recognition

Traditional NE 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 [MMG99] for a discussion of 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 recognised as a proper name by the POS tagger). Most patterns include some combination of proper noun or word with an initial capital letter (for English) 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 superclass of the concept may exist in the ontology

We therefore need to make more use of linguistic patterns and also contextual clues, rather than relying on gazetteer lists as with traditional recognition. Lexico-syntactic pattern-based ontology population has proven to be reasonably successful for a variety of tasks [ECD$^+$04]. The idea of acquiring semantic infomration from texts dates back to the early 1960s with Harris' *distributional hypothesis* [Har68] and Hirschman and Sager's work in the 1970s [HGS75], which focused on determining sets of sublanguage-specific word classes using syntactic patterns from domain-specific corpora. A detailed description and comparison of lexical and syntactic pattern matching can be found in [May00], In particular, research in this area has been used in specific domains such as medicine, where a relatively small number of syntactic structures is often found, for example in patient reports. Here the structures are also quite simple, with short and relatively unambiguous sentences typically found: this makes syntactic pattern matching much easier.

Text2Onto [CV05] performs synonym extraction on the basis of patterns. It combines machine learning approaches with basic linguistic processing such as tokenisation or lemmatisation and shallow parsing. Since like SPRAT it is based on the GATE framework, it offers flexibility in the choice of algorithms to be applied. Compared with our work, it has a smaller number of lexico-syntactic patterns. On the other hand, it applies additional statistical clustering and parsing for relation extraction All in all, this leads to more data, but not necessarily to an improvement in the resulting ontology in terms of precision.

We have identified three sets of patterns which can help us identify concepts, instances and properties to extend the ontology: the well-known Hearst patterns (Section 7.2.1), the Lexico-Syntactic Patterns developed in NeOn corresponding to Ontology Design Patterns (Section 7.2.2), and some new contextual patterns defined by us which take into account contextual information (Section 7.2.3).

7.2.1 Hearst patterns

The Hearst patterns are a set of lexico-syntactic patterns that indicate hyponymic relations [Hea92], and have been widely used by other researchers. Typically they 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. The patterns can be described by the following rules, where NP stands for a Noun Phrase and the regular expression symbols have their usual meanings[1]:

- `such NP as (NP,)* (or|and) NP`
  Example: ... works by such *authors* as *Herrick*, *Goldsmith*, and *Shakespeare*.
- `NP (,NP)* (,)? (or|and) (other|another) NP`
  Example: *Bruises*, *wounds*, or other *injuries*...
- `NP (,)? (including|especially) (NP,)* (or|and) NP`
  Example: All *common-law countries*, including *Canada* and *England*...

Hearst actually defined five different patterns, but we have condensed some of them into a single rule. Also, where Hearst defines the relations as hyponym-hypernym, we need to be more specific when translating this to an ontology, as they could represent either instance-class or subclass-superclass relations. To make this distinction, we tested various methods. In principle, POS-tagging should be sufficient, since proper nouns generally indicate instances, but our tagger mistags capitalised common nouns (at the beginning of sentences) as proper nouns frequently enough that we cannot rely on it for this purpose. We also looked at the presence or absence of a determiner preceding the noun (since proper nouns in English rarely have determiners) and whether the noun is singular or plural, but this still left the problem of the sentence-initial nouns. Finally, we decided to pre-process the text with the named entity recognition application ANNIE, and only consider certain types of named entities (*Person*, *Location*, *Organization*, and potentially some unknown entity types) as candidates for instances; all other NPs are considered to be classes. This gave much better results, occasionally missing an instance but almost never overgenerating.

7.2.2 Lexico-Syntactic Patterns

The second type of patterns investigated was the set of Lexico-Syntactic Patterns (LSPs) corresponding to Ontology Design Patterns [dCGPMPSF08]. We implemented a number of these patterns in our application (for the time being, we ignored some of the more complex relation types because we were not able to implement them easily). For each relation, there are several possible patterns: mostly these are all combined into a single rule in our grammars, but we separate them here for ease of comprehension. The grammars are written in JAPE [CMT00]: further details are discussed in Section 7.3.1.

[1] `() for grouping;` `|` for disjunction; `*`, `+`, and `?` for iteration.

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 verb[2]; `PUNCT` is punctuation; `NPlist` is a conjoined list of NPs ("X, Y and Z").

1. Subclass rules

   - `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>`

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

2. Equivalence rules

   - `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>`

   Example: *Poison dart frogs* are also called *poison arrow frogs*.

3. Properties

   - `NP<class> have NP<property>`
   - `NP<instance> have NP <property>`

   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 overgeneration. For example, in the following sentence:

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

the patterns produced the inference that *writers have penguins*, recognising *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. To start with, we took the simple patterns which generated new basic instances, subclasses and properties, described below.

7.2.3 Contextual patterns

We also defined 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 plugin 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:

[2] E.g., *classify in/into*, *comprise*, *contain*, *compose (of)*, *group in/into*, *divide in/into*, *fall in/into*, *belong (to)*.

1. **Add a new subclass**: `(Adj|N) NP<class>` → `NP<subclass>`.

   This matches a class name aready 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 amongst 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 superclass 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. Currently 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.

   Example: *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, so we can recognise *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:

   Example: *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.

7.3 SPRAT application

SPRAT (Semantic Pattern Recognition and Annotation Tool) is composed of a number of GATE components: some linguistic pre-processing followed by a set of gazetteer lists and the JAPE grammars described above. The components are as follows:

- Tokeniser: divides the text into tokens
- Sentence Splitter: divides the text into sentences
- POS-Tagger: adds part-of-speech information to tokens
- Morphological Analyser: adds morphological information (root, lemma etc.) to tokens
- NP chunker: divides the text into noun phrase chunks
- Gazetteers: looks up various items in lists
- OntoRootGazetteer (optional): looks up items from the ontology and matches them with the text, based on root forms
- JAPE transducers: annotates text and adds new items to the ontology

The application can either create an ontology from scratch, or modify an existing ontology. The ontology must be loaded with the application (in the former case, a blank ontology is loaded; in the latter, the ontology to be modified) and referenced by the grammar via the runtime parameter. The ontology used is the same one for the whole corpus: this means that if a number of documents are to be processed, the same ontology will be modified. If this is not the desired behaviour, then there are two options:

1. A separate corpus is created for each document or group of documents corresponding to a single output ontology. The application must be run separately for each corpus.
2. A processing resource can be added to the application that clears the ontology before re-running on the next document. This of course requires that the ontology is saved at the end of the application, after processing each document.

7.3.1 Implementation of patterns

The patterns themselves are implemented as JAPE rules. On the LHS of the rule is the pattern to be annotated. This consists of a number of pre-existing annotations which have been created as a result of pre-processing components (such as POS tagging, gazetteer lookup and so on) and earlier JAPE rules. The example below shows a pattern for matching a subclass relation, such as "Frogs are a kind of amphibian" where "frog" is a subclass of "amphibian".

```
Rule:Subclass1

 (
  ({CloneNP}):sub
  (
   {Lookup.minorType == be}
   {Token.category == DT}
   {Lookup.majorType == kind}
  )
  ({CloneNP}):super
) -->
```

This pattern matches a noun phrase (identified by a previous JAPE grammar), followed by the verb "to be" in some format (identified via the gazetteer lookup), a determiner (identified via the POS tagger), some word indicating a "kind of" relation (identified via the gazetteer lookup) followed by another noun phrase (identified by a previous JAPE grammar). The two noun phrases (corresponding ultimately to the subclass and superclass) are given labels ("sub" and "super" which will used in the second part of the rule.

The right hand side of the rule invokes NEBOnE and creates the new items in the ontology, as well as adding annotations to the document itself. NEBOnE is responsible also for ensuring that the resulting changes to the ontology are wellformed: this is described in more detail in Section 7.3.2.

```
{
// get the relevant annotations
gate.AnnotationSet subSet = (gate.AnnotationSet)bindings.get("sub");
gate.Annotation subAnn = (gate.Annotation)subSet.iterator().next();
AnnotationSet superSet = (gate.AnnotationSet)bindings.get("super");
gate.Annotation superAnn = (gate.Annotation)superSet.iterator().next();

// add a new class just below the root class
try {
neon.nebone.Nebone.createOrLinkClass(ontology, doc, superAnn, null);
```

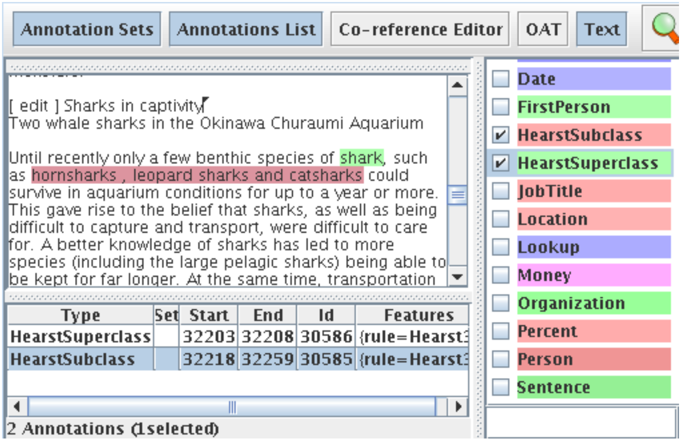
```
   }
   catch (ClassCastException e) {
System.err.println("Warning: ClassCastException in createOrLinkClass!");
        superAnn.getFeatures().put("debugInfo", "CCE in cOLC");
   }

// add a new subclass of the class just added
   try {
     neon.nebone.Nebone.createOrLinkClass(ontology, doc, subAnn, superAnn);
   }
   catch (ClassCastException e) {
System.err.println("Warning: ClassCastException in createOrLinkClass!");
        superAnn.getFeatures().put("debugInfo", "CCE in cOLC");
   }
},

// create the annotations on the documents
:sub.Subclass = {rule=Subclass1},
:super.Superclass = {rule=Subclass1}
```

The RHS of the rule first gets the relevant information from the annotations (using the labels assigned on the LHS of the rule), then adds a new class below the root class for the superconcept (labelled "amphibian" in our example), a new subclass of this (labelled "frog" in our example), and finally adds annotations to the entities in the document.

Figure 7.1 shows a screenshot from GATE of annotations added, while Figure 7.2 shows a screenshot from GATE of an ontology created.



[edit] Sharks in captivity
Two whale sharks in the Okinawa Churaumi Aquarium

Until recently only a few benthic species of shark, such as hornsharks , leopard sharks and catsharks could survive in aquarium conditions for up to a year or more. This gave rise to the belief that sharks, as well as being difficult to capture and transport, were difficult to care for. A better knowledge of sharks has led to more species (including the large pelagic sharks) being able to be kept for far longer. At the same time, transportation

Type	Set	Start	End	Id	Features
HearstSuperclass		32203	32208	30586	{rule=Hearst3
HearstSubclass		32218	32259	30585	{rule=Hearst3

2 Annotations (1selected)

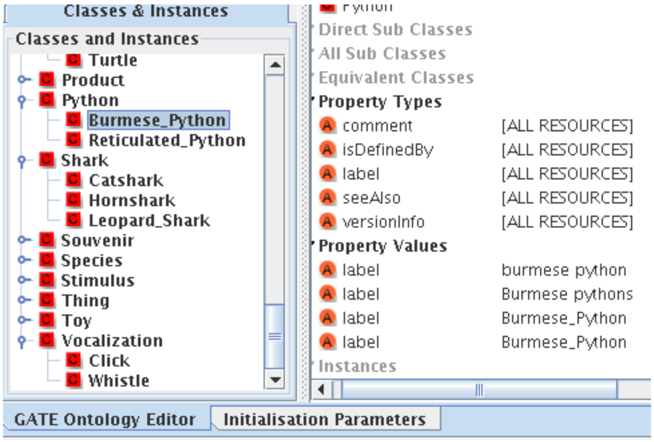
☐ Date
☐ FirstPerson
☑ HearstSubclass
☑ HearstSuperclass
☐ JobTitle
☐ Location
☐ Lookup
☐ Money
☐ Organization
☐ Percent
☐ Person
☐ Sentence

Figure 7.1: Annotation in GATE

7.3.2 NEBOnE

The SPRAT application uses the specially developed NEBOnE plugin for GATE in order to generate the changes to the ontology. NEBOnE (Named Entity Based ONtology Editing) is an implementation for processing natural language text and manipulating an ontology. It is derived from the CLOnE plugin [FTB+07] for GATE.

In CLOnE, input sentences are analysed deterministically and compositionally with respect to a given ontology, which the software consults in order to interpret the input semantics. CLOnE allows users to design,



Property Types

comment	[ALL RESOURCES]
isDefinedBy	[ALL RESOURCES]
label	[ALL RESOURCES]
seeAlso	[ALL RESOURCES]
versionInfo	[ALL RESOURCES]

Property Values

label	burmese python
label	Burmese pythons
label	Burmese_Python
label	Burmese_Python

Figure 7.2: Generated ontology in GATE

create, and manage information without knowledge of complicated standards (such as XML, RDF and OWL) or ontology engineering tools. It is implemented as a simplified natural language processor that allows the specification of logical data for semantic knowledge technology purposes in normal language, but with high accuracy and reliability. The components are based on GATE's existing tools for IE (information extraction) and NLP (natural language processing). Because the parsing process is deterministic, accuracy is not an issue: as long as the user specifies their input in correct controlled language, the system always produces correct output.

Because CLOnE was designed to be used with Controlled Language textual input, it is quite restricted in the patterns it can process and in how it generates the ontological data from them. We therefore developed NEBOnE in order to deal specifically with free text input. As its name suggests, is based on named entity recognition rather than a restricted set of keywords and noun phrases. For example, in CLOnE, the only way to derive a new subclass is by using a specific pattern containing the restricted keywords `type of` followed by the name of the class, e.g., "a dog is a type of animal". In free text, however, there are many ways in which a subclass could be stated, e.g., "animals such as dogs" or "dogs are animals", and so on (as described in Section 7.2); the use of a controlled language avoids ambiguity between syntactic structures. CLOnE also imposed strict rules on the order of input sentences and the creation of resources in the ontology. For example, the function to create a subclass required the superclass to exist already, and the function to create an instance made the same stipulation about the new instance's class. We cannot avoid this problem in NEBOnE: this is the sacrifice made for the gain in flexibility of input, which is essential with real world texts.

NEBOnE is based on the same underlying principles as CLOnE and is realised as another GATE plugin. The idea behind NEBOnE is that once a text has been annotated using Named Entity recognition techniques, these annotations can be used to generate new concepts, instances and properties in the ontology. CLOnE uses so-called *chunks* from the input sentences as candidates for inclusion in the ontology as classes, instances and properties: these are noun phrases previously created by a chunker in GATE. In NEBOnE, however, a *chunk* can be any annotation previously created, and does not need to correspond to a noun phrase, thereby ensuring a great deal more flexibility. When the NEBOnE plugin is installed, actions concerning the ontology are implemented on the RHS of JAPE rules, such as adding or deleting new classes, instances, subclasses, properties and so on.

If an item is selected for addition to the ontology as a new class, NEBOnE first checks to see whether the item in question already exists in the ontology: if it already exists in the place where it is scheduled to be added, NEBOnE will do nothing. If the item exists as a class elsewhere in the ontology, NEBOnE will add the new class (because it supports multiple inheritance). If the requested parent class and subclass both exist and are class names, NEBOnE will make the second a subclass of the first and print a message. If either is already an instance, or the parent class does not exist yet, NEBOnE will print a warning message.

Similarly for an instance, if it exists elsewhere as an instance, NEBOnE will add the new instance but generate a notification message. If the item already exists as a class, and an instance of the same name is to be added, or vice versa, then NEBOnE will not generate the new instance/class and will produce a warning message. Thus NEBOnE ensures consistency in the ontology, avoiding the need to run a checker after the ontology has been modified. A user can of course choose to ignore any potential inconsistencies, by checking the generated messages and then manually adding any offending items or making other changes to the ontology.

7.3.3 Implementation of NEBOnE

Once the text has been pre-processed, a JAPE transducer processes each sentence in the input text and manipulates the ontology appropriately. This PR refers to the contents of the ontology in order to analyse the input sentences and check for errors; some syntactically identical sentences may have different results if they refer to existing classes, existing instances, or non-existent names, for example.

The *canonical* feature—from which the name of a new class or instance is derived—is the concatenation of the string values of the tokens and underscores for the space-tokens (which can represent literal spaces, tabs or newlines), or the token lemmas.

The Java code that tests chunks in the input text against existing classes and instances in the ontology returns a match if any of the three features of the chunk (canonical, root or string) is case-insensitively equal to any of those features of an existing class or instance; for example, the chunks `'multiword expressions'` and `Multiword expressions` match each other, although the class name in the ontology varies according to which one is first used to create the class.

7.3.4 NEBOnE functions

In this section, we describe the main functions from NEBOnE that we use in our application. These are based on the functions developed for CLOnE, but with some differences (described below).

Create a new class

e.g., `There are universities.`
For each chunk, create a new class (`University`) immediately under the top class.

Create a new instance

e.g., `'University of Sheffield' is a university.`
For each chunk, create an instance (`University_of_Sheffield` of the class (`University`). If the class `University` does not exist in the ontology, create it spontaneously (immediately under the top class). If the instance and class already exist, make the instance a member of the specified class (in addition to other classes it already belongs to—both NEBOnE and the GATE Ontology API allow this).

Create a new subclass

e.g., `Dogs are a type of mammal.`
Make each chunk (`Dog`) a subclass of the superclass chunk (`Mammal`); create any required classes that do not already exist in the ontology. This function creates subclass-superclass relations between existing classes too (both NEBOnE and the GATE Ontology API support multiple inheritance).

Create a new property

CLASSES/INSTANCES have CLASSES/INSTANCES.
e.g., `Dolphins have acute eyesight and few natural enemies.`
Iterate through the cross-product of the chunks in the two chunk-lists. For each pair, if both are classes, create a property of the form *Domain_has_Range*. If both are instances, find a suitable property and instantiate it with those instances; if there is a class-instance mismatch or a suitable property does not exist, generate an error. (Unlike the others above, this function requires the classes or instances to exist already—it would be impossible to determine what to create otherwise.)

Changes from CLOnE

Because CLOnE was designed to interpret without ambiguity a controlled language that users wrote deliberately for the purposes of creating and editing ontologies, it imposed strict rules on the order of input sentences and the creation of resources in the ontology. For example, the function to create a subclass required the superclass to exist already, and the function to create an instance made the same stipulation about the new instance's class. NEBOnE, however, processes uncontrolled natural language so we cannot impose such restrictions. The NEBOnE library does, however, reject function calls that would otherwise try to create an instance with the same name as an existing class, or the other way round.

7.4 Evaluation

We evaluated the accuracy of the lexical patterns using a corpus of 25 randomly selected wikipedia articles about animals, such as the entries for *rabbit*, *sheep* etc. We ran SPRAT and examined the results in some detail. In total, SPRAT generated 201 classes, 21 instances, and 98 synonyms, and 107 other properties. Table 7.1 shows the results for each type. Note that, unlike in traditional named entity recognition evaluation, we use a strict method of scoring where a partially correct response, i.e. one where the span of the extracted entity is too short or too long, is considered as incorrect. This is because for ontology population, having an incorrect span is generally a more serious error than in named entity recognition.

7.4.1 Subclass Relations

In total, 163 subclass relations were generated, of which 79 were correct (48%). However, 15 of these were not really useful classifications: e.g., *turtle* as subclass of *local creature* makes sense only in a very specific context. Of the subclass relations, 69 were found by the Hearst patterns, of which 58 were correct (84%). Of the incorrect relations generated, most contained at least one correct subclass out of a list. For example, in the phrase "by disturbing the natural state of pasture, sheep and other livestock. . . " both *natural state of pasture* and *sheep* were recognised as subclasses of *livestock*, of which the former is incorrect but the latter is correct. Some refinement of the rules (for example, avoiding NPs containing *of*) could help improve the results. The remaining patterns accounted for 94 of the subclass relations, of which 21 were correct (22%).

Relation	Total Extracted	Correct	Precision
Subclass	163	79	48.5%
Instance	21	10	47.6%
Synonym	98	47	48.0%
Property	107	24	22.4%

Table 7.1: Results of relation extraction on 25 wikipedia documents

We experimented also with restricting possible classes and subclasses to terms found by TermRaider, a term selection algorithm based on linguistic filtering and tf-idf scoring. If we narrow the results to match only subclasses which are also terms, this improves the precision but lower the recall a little. Adjusting TermRaider's parameters to be a little more flexible with patterns should improve the recall, however. For subclass relations which are also terms, we find 50 occurrences, of which 31 are correct (62%). Of these, the Hearst patterns are almost entirely correct, though lacking a little in recall (95% correct, but only 20 are found in total), while the patterns found by the other rules are 40% correct, which is double the previous score, although only 30 are found in total.

7.4.2 Instances

The recognition of instances, on the other hand, was quite low in recall though with the same precision as that of subclasses. In total, 21 instances were found, of which 10 were correct. Many of the incorrect results were either as a result of erroneous named entity recognition (e.g. *Harmonia axyridis Pupal stage eggs Coccinellidae* was wrongly extracted as a named entity) or due to instances which are somewhat irrelevant (e.g. *San Francisco Bay* was extracted as an instance of *principal area* in the phrase "some of the principal areas are San Francisco Bay, Richardson Bay, Tomales Bay and Humboldt", which is factually correct but not useful to extract). The main reason for missing instances is that they were wrongly extracted either as subclasses or synonyms.

7.4.3 Synonyms

98 synonyms were found, of which 47 were correct (again 48%). Of the incorrect responses, some of the so-called synonyms were actually instances. For example, "A talking dolphin called Howard", where *Howard* was identified as a synonym of *talking dolphin* rather than as an instance. This was found to be simply due to a bug in the rule which prevented items identified as Entity from being classed as synonyms. Many of the incorrect synonyms were due to the relationship between the two items only holding in a very particular context. For example, in the sentence "A modified slit called a spiracle is located just behind the eye", the system identifies *modified slit* as a synonym of *spiracle*. Clearly not all modified slits are spiracles: in fact, *spiracle* should be extracted as a subclass of *modified slit*. Since there are many of these examples, we need to look more closely at the rules governing these.

7.4.4 Properties

Aside from synonyms, the system found 107 class and instance properties. Of these, we analysed the first 50, of which 17 were correct, 13 were incorrect, 5 correct but irrelevant, and 15 were correct but had the wrong span of either the domain or range. An improvement to the results could be made by creating a stop list of certain nouns which should not be included as possible ranges, perhaps restricting such nouns to terms, as with the subclasses.

7.5 Discussion

We can see that the patterns implemented are far from foolproof, since unlike with a controlled language such as CLOnE, we cannot rely on a one-to-one correspondence between a simple syntactic structure and its semantics. First we have the problem of overgeneration. Already, we have discarded some potential patterns (such as some of the LSPs) that we consider to generate too many errors. Further refinement is still necessary here, either to remove other patterns or to reimplement them in a different way.

One of the main causes of overgeneration is caused by the span of the noun phrase describing the concept to be added to the ontology. We have experimented with different possibilities. A larger span provides finer distinctions and thus better classes, but overgenerates considerably, while a smaller span produces more general classes but better accuracy (does not overgenerate so much). For example, in the following sentence:

> The individuals communicate using a variety of clicks, whistles and other vocalizations.

variety of click is recognised as a subclass of *vocalization*. While this is technically correct, a better interpretation would be simply the subclass *click*.

Similarly in the sentence:

> A minor class of sheep are the dairy breeds.

minor class of sheep is recognised as a superclass, with subclass *dairy breeds*. Better would be to recognise the superclass simply as *sheep*, since the subclass relation indicates the *minor class* relationship.

On the other hand, if we reduce the span of the noun phrase, we risk losing some important information. For example, in the sentence:

> Mygalomorph and Mesothelae spiders have two pairs of book lungs filled with haemolymph

if we do not identify the full noun phrase *two pairs of book lungs*, we can end up with the rather uninformative property *two pairs* of the class *Mesothelae spider*. A closer analysis of the spans is needed, which may help identify which patterns require longer spans than others, for example. Currently, the NP chunker decides the span for all noun phrases, but it may be more appropriate to alter the functionality of the chunker depending on the pattern, or to restrict the NPs to terms using TermRaider for some cases.

Second, lexical patterns tend to be quite ambiguous as to which relations they indicate. For example, `NP have NP` could indicate an object property or a datatype property relationship. Also, English word order can lead to inverse relations. For example, in the sentence “A traditional Cornish pilchard dish is the stargazy pie”, *stargazy pie* is a kind of Cornish pilchard dish, but the sentence can equally be written “The stargazy pie is a traditional Cornish pilchard dish”. Here, the use of the definite and indefinite determiner helps to identify the correct relationship, but this is not always the case. Often, further context is also crucial. For example, in the sentence “Both African males and females have external tusks”, it is not very useful to extract the concept *females* with the property *have external tusks* unless you know that *females* actually refers to female African elephants. To extract this information would require also coreference matching.

Care also needs to be taken to avoid actual erroneous (rather than simply spurious) results. For example, if negatives are not taken into acount, the consequences can be disastrous. From the phrase “DAT is a legitimate therapy”, we could easily deduce that *DAT* could be classified as an instance of *therapy*. However, further inspection of the wider context reveals that the opposite is true, as the sentence actually reads “Reviews of this and other published dolphin-assisted therapy (DAT) studies have found important methodological flaws and have concluded that there is no compelling scientific evidence that DAT is a legitimate therapy.” Of course, this is a common problem with shallow NLP systems.

To summarise, we are currently investigating a number of options to improve the recognition. First, we are looking at the incorporation of deeper semantic relations using semantic classes from VerbNet[Sch05] and WordNet[Fel98] in order to look for verbal patterns connecting terms in a sentence. We make use of the ANNIC plugin in GATE [ATBC05] to search for frequently occurring annotation patterns. We are also investigating the use of TermRaider for restricting the number of candidates for extraction. Also, we plan to incorporate combinations of Hearst patterns and statistically derived collocational information, because its combination with lexico-syntactic patterns has proven to improve precision and recall [CW03].

Integration of a full parser has also been investigated, but discarded on the grounds of speed (full parsing is extremely computationally expensive in this situation). In particular, we found that the sentences in Wikipedia articles, which we have used for training and testing, are quite hard to parse well, because they frequently exhibit a long and complex sentence structure which is highly ambiguous to a parser. This causes not only speed but also accuracy problems.

7.6 Further Work

Because the innovative character of this work lies in engineering rather than in research, we need to emphasise that, in this phase, the strength of the approach lies in its fundamental approach to linguistically motivated ontology engineering. An increasing number of atomic ontology editing operations are associated with lexico-syntactic patterns, according to the ontological information these patterns and the participating entities contribute. The flexibility of this association enables us to approach the transformation of linguistic structures into lightweight ontological knowledge in an incremental fashion. Also, the opportunity to incorporate any kind of additional knowledge into the system allows us to experiment with different settings, and use SPRAT as a research platform rather than a black box product. This sets it apart from partial approaches such as Hearst, because it offers a platform to dynamically include new algorithms.

In summary, the SPRAT tool assists the user in the generation and/or population of ontologies from text, using linguistic patterns. We have developed a number of new GATE plugins, including NEBOnE for editing the ontology, and TermRaider for finding new terms. In general, the rules give good recall, but precision is a little low. The idea behind this work is to investigate the extent to which such patterns can be used either on their own or in conjunction with the user to generate or populate a more detailed ontology from text. Currently, there is a good basis, but some work to go in improving the rules, and we have put forward a number of suggestions for ways in which this might be done.

Chapter 8

Conclusions and future work

During performing the work that is reported in this deliverable, the involved partners have drawn the following major conclusions which will influence the further evolution of this task.

First, we have updated the three level categorization of NORs, introduced in [SFdCB$^+$08], according to three different features: type of NOR, data model and implementation. Moreover, we presented a pattern based approach for re-engineering NORs into ontologies. We take advantage of the NOR data model to define patterns for re-engineering NORs. This approach is covered in the following Chapters: 3, 4, and 5. This approach will be extended to cover glossaries and lexicons, and create richer and more complex ontologies. We plan to include the algorithms and implementations later on in a framework which will implement the transformation process. Also we will include a section on the PR-NORs to generate ontologies following the Linking Open Data[1] recommendations. We also need to evaluate how much effort do we save re-engineering NORs using patterns compared with re-engineering NORs without them.

Second, we presented FLOR, a tool for automatic enrichment of folksonomy tagspaces with Semantic Entities automatically discovered from online ontologies. The experiments reported in Section 6.1.5 we demonstrated FLOR's functionalities and performance and provided additional insights for future work on the enhancement of the FLOR enrichment algorithm. Finally as described in Section 6.1.3 the final step of FLOR, which is the SCARLET Semantic Relation Discovery hasn't yet been integrated with FLOR and thus is part of our future work.

Finally, the SPRAT tool described in Chapter 7 uses a method for automatically extracting entities from unstructured text in any domain, and generating and/or populating an ontology with the resulting information. The method relies on the identification of syntactic and semantic linguistic patterns in unstructured text, in combination with methods for generic named entity recognition and term extraction. The tool is deployed as a plugin for the Neon Toolkit, and a domain-specific version of it, SARDINE, is used in WP8 for the augmentation of fisheries ontologies. Currently, a number of patterns have been deployed, but ongoing work is investigating the incorporation of deeper semantic information and further refinement of the patterns, in order to improve precision.

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

Bibliography

[ABM05] Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings Between XML Data and Ontologies. In *International Semantic Web Conference*, pages 6–20, 2005.

[AM99] D. Appelt and D. Martin. Named entity extraction from speech: Approach and results using the TextPro system. In *Proceedings of DARPA Broadcast News Workshop*, pages 51–54, 1999.

[AM05] Y. An and J. Mylopoulos. Translating XML Web Data into Ontologies. In *OTM Workshops*, pages 967–976, 2005.

[ANS05] ANSI/NISO. *Documentation – Guidelines for the construction, format, and management of monolingual controlled vocabularies.*, 2005. Report ANSINISO Z3919.

[ASC07] R. Abbasi, S. Staab, and P. Cimiano. Organizing resources on tagging systems using t-org. In *In proceedings of Workshop on Bridging the Gap between Semantic Web and Web 2.0 at ESWC 2007*, June 2007.

[ASSM07] S. Angeletou, M. Sabou, L. Specia, and E. Motta. Bridging the gap between folksonomies and the semantic web: An experience report. In *4th European Semantic Web Conference*, pages 30–43, Innsbruck, Austria, 2007.

[ATBC05] N. Aswani, V. Tablan, K. Bontcheva, and H. Cunningham. Indexing and Querying Linguistic Metadata and Document Content. In *Proceedings of Fifth International Conference on Recent Advances in Natural Language Processing (RANLP2005)*, Borovets, Bulgaria, 2005.

[Bar07] J. Barrasa. *Modelo para la definición automática de correspondencias semánticas entre ontologías y modelos relacionales*. PhD thesis, Facultad de Informatica, Universidad Politecnica de Madrid, Madrid, Spain, March 2007.

[BCGP04] J. Barrasa, O. Corcho, and A. Gómez-Pérez. R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. In *Second Workshop on Semantic Web and Databases (SWDB2004)*, 2004.

[Ber94] J. Berge. *The EDIFACT Standards*. Blackwell Publishers, Inc., Cambridge, MA, USA, 1994.

[BH06] S. Brockmans and P. Haase. A Metamodel and UML Profile for Networked Ontologies. A Complete Reference. Technical report, Universität Karlsruhe„ 2006.

[BON03] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 148–151. Edmonton, Canada, 2003.

[Bra05] D. Brandon. Recursive database structures. *Journal of Computing Sciences in Colleges*, 2005.

[BS 05a] British Standards Institution, BSI. *Documentation – Structured vocabularies for information retrieval - Guide - Part 1: Definitions, symbols and abbreviations.*, 2005. Report BS 8723-1.

[BS 05b] British Standards Institution, BSI. *Documentation – Structured vocabularies for information retrieval - Guide - Part 2: Thesauri.*, 2005. Report BS 8723-2.

[BS 05c] British Standards Institution, BSI. *Documentation – Structured vocabularies for information retrieval - Guide - Part 5: Exchange formats and protocols for interoperability.*, 2005. Report BS 8723-5.

[Car02] B. Carkenord. Why Build a Logical Data Model. http://www.embarcadero.com/resources/tech_papers/datamodel.pdf, 2002.

[Chi98] Nancy A. Chinchor. Proceedings of the Seventh Message Understanding Conference (MUC-7) named entity task definition. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, page 21 pages, Fairfax, VA, April 1998. version 3.5, http://www.itl.nist.gov/iaui/894.02/related_projects/muc/.

[CMB^{+}02] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. *The GATE User Guide*. `http://gate.ac.uk/`, 2002.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[CMT00] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield, November 2000.

[Col02] M. Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia,PA, 2002.

[Cor05] O. Corcho, editor. *A Layered Declarative Approach to Ontology Translation with Knowledge Preservation*. IOS Press, 2005.

[CV05] P. Cimiano and J. Voelker. Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, Alicante, Spain, 2005.

[CV07] R. Cilibrasi and P. Vitanyi. The google similarity distance. *Transactions on Knowledge and Data Engineering, IEEE*, 19(3):370–383, 2007.

[CW03] S. Cederberg and D. Widdows. Using lsa and noun coordination information to improve the precision and recall of automatic hyponymy extraction. In *Proceedings of the 7th conference on Natural language learning at HLT-NAACL*, pages 111–118, Morristown, NJ, 2003.

[CXH04] I. F. Cruz, H. Xiao, and F. Hsu. An ontology-based framework for xml semantic integration. In *IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium*, pages 217–226, Washington, DC, USA, 2004. IEEE Computer Society.

[dCGPMPSF08] G. Aguado de Cea, A. Gómez-Pérez, E. Montiel-Ponsoda, and M-C. Suárez-Figueroa. Natural language-based approach for helping in the reuse of ontology design patterns. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns (EKAW 2008)*, pages 32–47, Acitrezza, Italy, September 2008.

[ECD$^+$04] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale Information Extraction in KnowItAll. In *Proceedings of WWW-2004*, 2004. http://www.cs.washington.edu/research/knowitall/papers/www-paper.pdf.

[FB06] D. Foxvog and C. Bussler. Ontologizing EDI Semantics. In *ER (Workshops)*, pages 301–311, 2006.

[Fel98] Christiane Fellbaum, editor. *WordNet - An Electronic Lexical Database*. MIT Press, 1998.

[FH02] M. Fleischman and E. Hovy. Fine grained classification of named entities. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, Taipei, Taiwan, 2002.

[FTB$^+$07] A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Handschuh. Clone: Controlled language for ontology editing. In *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, November 2007.

[GC05] R. García and O. Celma. Semantic Integration and Retrieval of Multimedia Metadata. In *Proceedings of the ISWC 2005 Workshop on Knowledge Markup and Semantic Annotation (Semannot'2005)*, 2005.

[GGMO03] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WORDNET with DOLCE. *AI Mag.*, 24(3):13–24, 2003.

[GH06] S. Golder and B. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006.

[GNV03] A. Gangemi, R. Navigli, and P. Velardi. The OntoWordNet Project: Extension and Axiomatization of Conceptual Relations in WordNet. In *CoopIS/DOA/ODBASE*, 2003.

[GPC08] Jose Manuel Gómez-Pérez and Oscar Corcho. Problem-solving methods for understanding process executions. *Computing in Science and Engg.*, 10(3):47–52, 2008.

[GPS98] A. Gangemi, D. Pisanelli, and G. Steve. Ontology integration: Experiences with medical terminologies. *Ontology in Information Systems*, pages 163–178, 1998.

[Hah03] V. Hahn. Turning informal thesauri into formal ontologies: a feasibility study on biomedical knowledge re-use. *Comparative and Functional Genomics*, 4:94–97(4), January/February 2003.

[Har68] Z.S. Harris. *Mathematical Structures of Language*. Wiley (Interscience), New York, 1968.

[HdB07] M. Hepp and J. de Brujin. GenTax: A generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies. In *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*. Springer-Verlag, 2007.

[Hea92] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Conference on Computational Linguistics (COLING'92)*, Nantes, France, 1992. Association for Computational Linguistics.

[HFP$^+$06] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi. RDF123: a mechanism to transform spreadsheets to RDF. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[HGS75] L. Hirschman, R. Grishman, and N. Sager. Grammatically based automatic word class formation. *Information Processing and Retrieval*, 11:39–57, 1975.

[HHST06] S. Hakkarainen, L. Hella, D. Strasunskas, and S. Tuxen. A Semantic Transformation Approach for ISO 15926. In *Proceedings of the OIS 2006 First International Workshop on Ontologizing Industrial Standards*, 2006.

[Hir04] G. Hirst. Ontology and the lexicon. In *Handbook on Ontologies in Information Systems*, pages 209–230. Springer, 2004.

[Hod00] G. Hodge. Systems of Knowledge Organization for Digital Libraries: Beyond Traditional Authority Files. http://www.clir.org/pubs/reports/pub91/contents.html, 2000.

[HS03] U. Hahn and S. Schulz. Towards a broad-coverage biomedical ontology based on description logics. pac symp biocomput. pages 577–588, 2003.

[HVTS08] E. Hyvönen, K. Viljanen, J. Tuominen, and K. Seppälä. Building a national semantic web ontology and ontology service infrastructure -the finnonto approach. In *ESWC*, pages 95–109, 2008.

[IK02] H. Isozaki and H. Kazawa. Efficient Support Vector Classifiers for Named Entity Recognition. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, pages 390–396, Taipei, Taiwan, 2002.

[ISO85] International Standard Organization (ISO). *Documentation – Guidelines for the establishment and development of multilingual thesauri*, 1985. Report ISO 5964.

[ISO86] International Standard Organization (ISO). *Documentation – Guidelines for the establishment and development of monolingual thesaurus*, 1986. Report ISO 2788.

[ISO04] International Standard Organization (ISO). *Information technology - Metadata registries - Part 1: Framework*, 2004. Report ISO/IEC FDIS 11179-1.

[KBH$^+$97] T. Koch, A. Bummer, D. Hiom, M. Peereboom, A. Poulter, and E. Worsfold. Specification for resource description methods Part 3. the role of classification schemes in Internet resource description and discovery. Technical report, DESIRE project deliverable D3.2, 1997.

[Lab07] Lawrence Berkeley National Laboratory. eXtended MetaData Registry (XMDR) Project. http://www.xmdr.org/standards/cmaps/Thesaurus Standards Relationships.html, 2007.

[LS06] B. Lauser and M. Sini. From agrovoc to the agricultural ontology service/concept server: an owl model for creating ontologies in the agricultural domain. In *DCMI '06: Proceedings of the 2006 international conference on Dublin Core and Metadata Applications*, pages 76–88. Dublin Core Metadata Initiative, 2006.

[M. 07] M. d Aquin and M. Sabou and M. Dzbor and C. Baldassarre and L. Gridinoc and S. Angeletou and E. Motta. Watson: A gateway for the semantic web. In *4th European Semantic Web Conference*, Innsbruck, Austria, 2007.

[May00] D. G. Maynard. *Term Recognition Using Combined Knowledge Sources*. PhD thesis, Manchester Metropolitan University, UK, 2000.

[May03] D. Maynard. Multi-source and multilingual information extraction. *Expert Update*, 2003.

[MB05] A. Miles and D. Brickley. SKOS Core Vocabulary Specification. Technical report, World Wide Web Consortium (W3C), November 2005. http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/.

[MBC] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of named entities. pages 255–261. `http://gate.ac.uk/sale/ranlp03/ranlp03.pdf`.

[MBC03] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of Named Entities. In *Recent Advances in Natural Language Processing*, Bulgaria, 2003.

[MC03] D. Maynard and H. Cunningham. Multilingual Adaptations of a Reusable Information Extraction Tool. In *Proceedings of the Demo Sessions of EACL'03*, Budapest, Hungary, 2003. ACL.

[MDA07a] M. Zied Maala, A. Delteil, and A. Azough. A conversion process from flickr tags to rdf descriptions. In *SAW*, 2007.

[MDA07b] M. Zied Maala, A. Delteil, and A. Azough. A conversion process from flickr tags to rdf descriptions. In *10th International Conference on Business Information Systems*, Poznan, Poland, 2007.

[Mil05] A. Miles. Quick Guide to Publishing a Thesaurus on the Semantic Web. Technical report, World Wide Web Consortium (W3C), May 2005. http://www.w3.org/TR/2005/WD-swbp-thesaurus-pubguide-20050510/.

[MLP08] Diana Maynard, Yaoyong Li, and Wim Peters. Nlp techniques for term extraction and ontology population. In P. Buitelaar and P. Cimiano, editors, *Bridging the Gap between Text and Knowledge - Selected Contributions to Ontology Learning and Population from Text*. IOS Press, 2008.

[MMG99] A. Mikheev, M. Moens, and C. Grover. Named Entity recognition without gazetteers. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 1–8, 1999.

[MS01] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 2001.

[MSY$^+$07] D. Maynard, H. Saggion, M. Yankova, K. Bontcheva, and W. Peters. Natural Language Technology for Information Integration in Business Intelligence. In *10th International Conference on Business Information Systems (BIS-07)*, Poznan, Poland, 25-27 April 2007.

[MTB$^+$03] D. Maynard, V. Tablan, K. Bontcheva, H. Cunningham, and Y. Wilks. Muse: a multi-source entity recognition system. *Submitted to Computers and the Humanities*, 2003.

[MTBC03] D. Maynard, V. Tablan, K. Bontcheva, and H. Cunningham. Rapid customisation of an Information Extraction system for surprise languages. *Special issue of ACM Transactions on Asian Language Information Processing: Rapid Development of Language Capabilities: The Surprise Languages*, 2:295–300, 2003.

[MTC$^+$02] D. Maynard, V. Tablan, H. Cunningham, C. Ursu, H. Saggion, K. Bontcheva, and Y. Wilks. Architectural Elements of Language Engineering Robustness. *Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data*, 8(2/3):257–274, 2002.

[MYKK05] D. Maynard, M. Yankova, A. Kourakis, and A. Kokossis. Ontology-based information extraction for market monitoring and technology watch. In *ESWC Workshop "End User Aspects of the Semantic Web")*, Heraklion, Crete, 2005.

[MZ06] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data and Knowledge Engineering*, 2006.

[PEK03] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'03)*, 2003.

[PGD+08] V. Presutti, A. Gangemi, S. David, G. Aguado de Cea, M. C. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. NeOn Deliverable D2.5.1. A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. In *NeOn Project. http://www.neon-project.org*, 2008.

[PMC+02] K. Pastra, D. Maynard, H. Cunningham, O. Hamza, and Y. Wilks. How feasible is the reuse of grammars for named entity recognition? In *Proceedings of the 3rd Language Resources and Evaluation Conference*, 2002. `http://gate.ac.uk/sale/lrec2002/reusability.ps`.

[PS98] R. Pooley and P. Stevens. Software reengineering patterns. Technical report, 1998.

[SAd+07] M. Sabou, S. Angeletou, M. dAquin, J. Barrasa, K. Dellschaft, A. Gangemi, J. Lehman, H. Lewen, D. Maynard, D. Mladenic, M. Nissim, W. Peters, V. Presutti, and B. Villazón. Selection and integration of reusable components from formal or informal specifications. Technical report, NeOn project deliverable D2.2.1, 2007.

[Sch05] Karin Kipper Schuler. *VerbNet: A broad-coverage, comprehensive verb lexicon*. PhD thesis, University of Pennsylvania, 2005.

[SFBG+07] M. C. Suárez-Figueroa, S. Brockmans, A. Gangemi, A. Gómez-Pérez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou. Neon modelling components. Technical report, NeOn project deliverable D5.1.1, 2007.

[SFdCB+08] M.C. Suárez-Figueroa, G. Aguado de Cea, C. Buil, K. Dellschaft, M. Fernández-López, A. García-Silva, A. Gómez-Pérez, G. Herrero, E. Montiel-Ponsoda, M. Sabou, B. Villazón-Terrazas, and Z. Yufei. NeOn Methodology for Building Contextualized Ontology Networks. Technical report, NeOn project deliverable D5.4.1, 2008.

[SFGP08] M.C. Suárez-Figueroa and A. Gómez-Pérez. Towards a Glossary of Activities in the Ontology Engineering Field. In *Proceedings of the 6th Language Resources and Evaluation Conference (LREC 2008)*, 2008.

[SLL+04] D. Soergel, B. Lauser, A. Liang, F. Fisseha, J. Keizer, and S. Katz. Reengineering thesauri for new applications: The agrovoc example. *J. Digit. Inf.*, 4(4), 2004.

[SM07] L. Specia and E. Motta. Integrating folksonomies with the semantic web. In *4th European Semantic Web Conference*, pages 624–639, Innsbruck, Austria, 2007.

[Soe95] D. Soergel. Data models for an integrated thesaurus database. *Comatibility and Integration of Order Systems*, 24(3):47–57, 1995.

[SSV02] L. Stojanovic, N. Stojanovic, and R. Volz. A Reverse Engineering Approach for Migrating Data-intensive Web Sites to the Semantic Web. In *Proceedings of the Conference on Intelligent Information Processing*, 2002.

[TGEM07] R. Trillo, J. Gracia, M. Espinoza, and E. Mena. Discovering the semantics of user keywords. *Journal of Universal Computer Science*, 13(12):1908–1935, 2007.

[vAGS06] M. van Assem, A. Gangemi, and G. Schreiber. Conversion of WordNet to a standard RDF/OWL representation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006.

[vAMMS06] M. van Assem, V. Malaisé, A. Miles, and G. Schreiber. A Method to Convert Thesauri to SKOS. In *The Semantic Web: Research and Applications*, pages 95–109. 2006.

[vAMSW04] M. van Assem, M. Menken, G. Schreiber, and J. Wielemaker. A method for converting thesauri to RDF/OWL. In *Proceedings of the Third International Semantic Web Conference (ISWC)*. Springer, 2004.

[WB97] S.E. Wright and G. Budin, editors. *Handbook of terminology management, Basic aspects of terminology management*. John Benjamins Publishing Company, 1997.

[WP94] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 133 –138, New Mexico, USA, 1994.

[WSWS01] B.J. Wielinga, A. Th. Schreiber, J. Wielemaker, and J.A.C. Sandberg. From thesaurus to ontology. In *K-CAP '01: Proceedings of the 1st international conference on Knowledge capture*, pages 194–201, New York, NY, USA, 2001. ACM Press.

[YGS07] C. Yeung, N. Gibbins, and N. Shadbolt. Understanding the semantics of ambiguous tags in folksonomies. In *International Semantic Web Conference*, Busan, South Korea, 2007.