NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D2.5.2 Pattern based ontology design: methodology and software support

Deliverable Co-ordinator: **Enrico Daga**

Deliverable Co-ordinating Institution: **CNR**

Other Authors: **Eva Blomqvist (CNR); Aldo Gangemi (CNR); Elena Montiel (UPM); Nadejda Nikitina (AIFB); Valentina Presutti (CNR); Boris Villazón-Terrazas (UPM);**

Ontology design patterns (ODPs) is an emerging techinque based on the reuse of encoded good practices in the ontology development process. In this deliverable we present the latest results about the exploitation of ontology design patterns in ontology design, with particular focus on software support.

Document Identifier:	NEON/2009/D2.5.2/v1.0	Date due:	December 31st, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	January 31st, 2010
Project start date	March 1, 2006	Version:	v1.0
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.ump.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:

- CNR
- UPM
- AIFB

Change Log

Version	Date	Amended by	Changes
0.1	17-11-2009	Enrico Daga	Outline
0.2	27-11-2009	Enrico Daga	Outline updated; Added sections “eXtreme Design, an introduction“ and “XD Tools for the NeOn Toolkit”;
0.3	3-12-2009	Boris Villazón-Terrazas	Added section “The role of reengineering patterns“;
0.4	07-12-2009	Elena Montiel-Ponsoda	Added sections “The role of Lexico-Syntactic Patterns” and “Lexico-Syntactic Patterns: reusable JAPE code for NLP applications”
0.5	7-12-2009	Boris Villazón-Terrazas	Section “The role of reengineering patterns” updated
0.6	7-12-2009	Boris Villazón-Terrazas	Section “Re-engineering patterns: a software library” updated;
0.7	10-12-2009	Eva Blomqvist	Updated Sections “eXtreme Design (XD): an introduction", “The eXtreme design iteration" and “The role of Content Ontology Design Patterns" . Added section heading “Pattern-based ontology evaluation".
0.8	13-12-2009	Nadejda Nikitina	Added section "Pattern-Based Ontology Refinement", prelimenary version (contains additional images and references).
0.9	17-12-2009	Enrico Daga	Updated section “XD Tools for the NeOn Toolkit”; Outline updated.
1.0	18-12-2009	Enrico Daga	Added section “ontologydesignpatterns.org”.
1.1	18-12-2009	Elena Montiel-Ponsoda	Reviewed sections “The role of Lexico-Syntactic Patterns” and “Lexico-Syntactic Patterns: reusable JAPE code for NLP applications”
1.2	19-12-2009	Boris Villazón-Terrazas	Reviewed sections “The role of Re-engineering Patterns” and “Re-engineering Patterns: a software library”
1.3	20-12-2009	Enrico Daga	Added executive summary and Abstract.
1.4	20-12-2009	Nadejda Nikitina	Updated section "Pattern-Based Ontology Refinement".
1.5	05-01-2010	Eva Blomqvist	Language editing of complete document. Adding explanations how the methods fit together (in introduction to 1.2 and 1.3). Aligning terminology of all sections. Introducing appendix A and B that were previously parts of section 1.3 and 2.4. Removed related work in 2.5, and clarified connection to XD.
1.6	07-01-2010	Enrico Daga	Added Chapter Introduction. Moved 2.5 to the end of Chapter 1

Version	Date	Amended by	Changes
1.7	13-01-2010	Eva Blomqvist	In Chapter 1, eXtreme Design: revised according to QA advises
1.8	13-01-2010	Nadejda Nikitina	Updated Section about Ontology Refinement: changed the figure according to the review comments; rewritten the section Matching criteria; fixed the broken table reference; corrected 'Section' and 'Figure' all over the Section
1.9	13-01-2010	Elena Montiel-Ponsoda	Updated the sections “The role of Lexico-Syntactic Patterns” and “Lexico-Syntactic Patterns: reusable JAPE code for NLP applications” according to QA advises
2.0	13-01-2010	Boris Villazón-Terrazas	Updated the sections “The role of Re-engineering Patterns” and “Re-engineering ODPs: a software library” according to QA advises
2.1	14-01-2010	Enrico Daga	Executive summary rewritten; Introduction chapter updated; Added future work section for “XD Tools“ and “ODP Portal“; Conclusions added.
2.2	15-01-2010	Eva Blomqvist	Language review of executive summary, introduction and conclusions. Minor layout improvements.

Executive Summary

In this deliverable we refer to the experiences regarding the role of patterns in ontology design that have matured within WP2 (with input also from experiments in WP5), with focus on how to concretely deal with patterns in ontology development, starting from methodological aspects in Chapter 2 and then presenting end-user tools and software components in Chapter 3. Section 2.1 presents a general methodology for pattern based design - i.e. eXtreme Design. Such a methodology has been designed to cover all kinds of patterns, but the actual realization presented here is mainly focused on Content Ontology Design Patterns (Content ODPs).

Since the work on eXtreme Design is continuously evolving, we present also two emerging methods for the reuse of Re-engineering ODPs and Lexico-Syntactic ODPs. Within the framework of the NeOn methodology, Re-engineering ODPs define methods for transforming non-ontological resources to ontological resources. This method is summarized in Section 2.2. The method based on Lexico-syntactic ODPs, described in Section 2.3, targets novice users. Exploiting Lexico-syntactic ODPs we can identify associated ODPs with the aim of allowing a semi-automatic reuse of other ODPs, where a formulation of the modelling problem in Natural Language (NL) is the starting point. Both Re-engineering ODPs and Lexico-syntactic ODPs are candidate methods to be incorporated into, and experimented on, within the XD methodology in the future. Chapter 2 is concluded by Section 2.4 that presents a method for ontology refinement by the means of highly axiomatized Content ODPs.

Chapter 3 is entirely dedicated to software, exploiting Ontology Design Patterns (ODPs). The XD Tools for the NeOn Toolkit - Section 3.1 - provides support for some of the main tasks of the eXtreme Design workflow, e.g. reusing ontology pattern registries, Content ODP specialization, annotation, pattern selection and ontology analysis with respect to good practices in pattern based ontology design. Section 3.2 presents the ODP portal[1], a semantic web portal targeted to users interested in this topic, exposing registries of reusable patterns, with support for a discussion and evaluation process. Section 3.3 presents a software library that exploits the reuse of non-ontological resources by the means of Re-engineering ODPs. The library is implemented in a modular way and is easily extendable to support new patterns. Lexico-Syntactic ODPs have been implemented as a rule library to be used in NLP applications, in Section 3.4 we present their usage within the GATE framework.

[1] http://ontologydesignpatterns.org

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The notion of *patterns* has proved useful in many areas of design, such as architecture and software engineering. Ontology Design Patterns (ODPs) have been presented and formally defined in the context of NeOn in D2.5.1 [PGD$^+$08] (and in D5.1.1 [SFBG$^+$07]), where catalogues of patterns have already been presented . Within C-ODO, the model of collaborative ontology design presented in D2.1.2 [GP09a], ODPs play a crucial role, since they summarize the good practices to be applied within design solutions, and the design rationales that have motivated their adoption. Informally we can define an ODP as a modeling solution to solve a recurrent ontology design problem. Methodological aspects involving ODPs were the focus of D5.4.2 [SBD$^+$09] and D3.8,1[VB08]. In this deliverable we refer to the experiences regarding the role of patterns in ontology design that have matured within WP2 (with input also from experiments in WP5), with particular focus on pattern-enabled software and tool support for end-users.

The eXtreme Design (XD) is probably the only methodology proposed so far that is entirely focused on ODPs. Section 2.1 presents background and related work concerning design patterns in ontology development, and gives an example of an ontology development workflow reusing Content ODPs. A software tool has been developed as a NeOn Toolkit plugin for supporting the XD methodology: XD Tools (see Section 3.1). We present here version 1.0 that provides support for some of the main tasks of the workflow.

Within this perspective the ODP portal[1] - presented in Section 3.2 - offers a space to the community where users, who are interested in best practices for ontology design and ontology engineering, can discuss and collect design solutions, provide resources for training and perform ODP evaluation (see Section 3.2.4). The portal has been recently extended for supporting additional pattern types, e.g. Re-engineering ODPs.

Within the framework of the NeOn methodology, Re-engineering ODPs define methods for transforming non-ontological resources to ontological resources. This method, as summarized in Section 2.2, relies on a repository of patterns for re-engineering non-ontological resources (PR-NORs), also known as Re-engineering ODPs. In Section 3.3 we describe the Re-engineering ODP software library, a Java library that implements the transformation process suggested by the Re-engineering ODPs.

The method based on Lexico-syntactic ODPs, described in Section 2.3, targets novice users. Exploiting Lexico-syntactic ODPs we can identify associated ODPs with the aim of allowing a semi-automatic reuse of other ODPs, where a formulation of the modelling problem in Natural Language (NL) is the starting point. In this deliverable we release a new version of the Lexico-syntactic ODP repository (a very preliminary version was included in D2.5.1 [PGD$^+$08]), and also offer the JAPE code, that can be reused in NLP applications (see Section 3.4). Both Re-engineering ODPs and Lexico-syntactic ODPs are candidate methods to be incorporated into, and experimented on, within the XD methodology in the future.

Content ODPs are a resource that can also be exploited for ontology refinement. In Section 2.4 we present a pattern-based ontology refinement algorithm. The proposed approach searches for components within an ontology, that already partially instantiate a given ODP. In this way, potential missing ontology elements, present in the ODP, can be identified and reused to enrich the ontology.

[1] http://ontologydesignpatterns.org

Chapter 2 is dedicated to methodological aspects. Even though this deliverable is mainly focused on software support, this chapter gives an overview of methods involving ODPs and provides a background for Chapter 3, which is dedicated to software components and end-user tools.

Chapter 2

Developing ontologies using design patterns

2.1 eXtreme Design (XD): pattern based ontology design

Ontology Design Patterns (ODPs) is an emerging technique for the reuse of encoded experience and best practices. In the context of the NeOn project catalogues of ODPs have already been presented in D5.1.1 [SFBG+07] and D2.5.1 [PGD+08], and an initial version of the XD methodology, focusing mainly on Content ODPs, was presented in D5.4.2 [SBD+09]. In this deliverable a new version of the methodology is described, since it has been extended and detailed based on experimental results from WP5 (see D5.6.2 [DSFGP+09]) and its application in WP7 use-cases (such as described in D7.6.2 [EBC+09]).

With the name **eXtreme Design (XD)** we identify a family of methods based on the application, exploitation, and detection of ontology design patterns (ODPs) [PGD+08], for solving problems related to an ontology development project. The methods are in general characterized by exploiting two sets: the *problem space*, which is composed of the actual problems that have to be addressed during an ontology development project, and the *solution space*, which is made up of successful reusable solutions, e.g. ODPs. From the project's *solution space*, a number of ODPs are selected by matching the actual (local) problems (Local Use Case - LUC), from the ontology project's *problem space*, typically expressed in terms of competency questions (CQs)[GF94], to the General Use Cases (GUC) associated with the ODPs in the solution space. The general approach is schematized in Figure 2.1. The selected ODPs are then used according to specific guidelines, such guidelines for Content ODP reuse were presented in D5.4.2[SBD+09].

Below, we first present a brief introduction to existing work in the area of pattern-based ontology design, or rather point at the lack of such existing work, then we describe the principles of the XD methodology and give an example of a workflow with Content Ontology Design Patterns (Content ODPs).

2.1.1 State-of-the-art and related work

The notion of pattern has proved useful in the context of design within many areas, such as architecture, software engineering, etc. Within C-ODO (the collaborative ontology design model presented in D2.1.2 [GP09a]) ODPs play a crucial role, since they summarize good practices to be applied within design solutions, and keep track of the design rationales that have motivated their adoption. In this context ODPs are the basis for ontology design, starting from ontology requirements and resulting in ODPs integrated into the ontology network to be built.

Even though the use of patterns is widespread in computer science in general, for instance in areas such as software engineering, there are very few methodologies that explicitly mention the use of patterns, and if mentioned they are usually proposed as a kind of additional support that may guide developers within any methodology. So far, very few purely pattern-based methodologies have been proposed, in ontology engineering or elsewhere. In ontology engineering pattern-based methods are present primarily at the logical

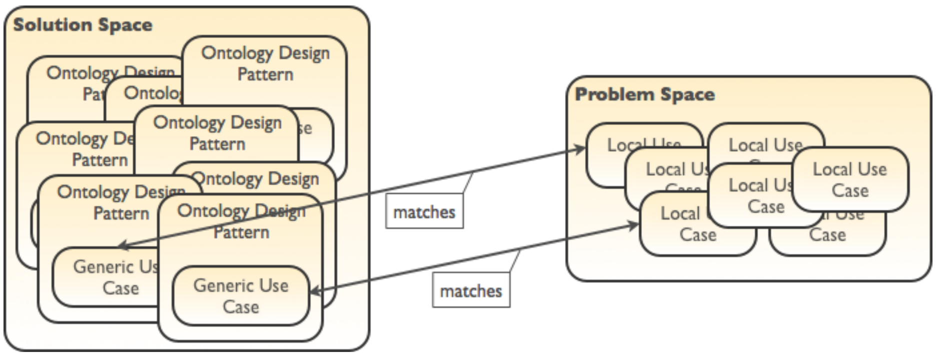


Figure 2.1: The general approach of eXtreme Design. ODPs represent the ontology project's *solution space*, which is used as the main knowledge source for addressing ontology design problems, e.g. reengineering, evaluation, design, etc.

level, where patterns support methods for ontology learning, enrichment and similar tasks. In these methods patterns are used more or less automatically, e.g. lexico-syntactic patterns to for example identify ontological elements in a natural language text or to extract relations between ontology concepts (see Section 2.3). Such methods have been proposed in several NeOn deliverables, e.g. [VB08], and in work such as [MFP09]. In the first chapter of this deliverable, on the other hand, the focus is on methodological support, rather than tool support.

As already mentioned, within this methodological focus, there is even less prior work within the ontology engineering field, whereas NeOn is at the forefront of this development. Some operations on ODPs were, however, already presented in D2.5.1 [PGD$^+$08] and a first version of the XD method was introduced in D5.4.2 [SBD$^+$09]. An existing methodology for pattern-based ontology design was presented by Clark and Porter in [CP97]. The difference between this method and XD is that the former do not consider collaboration, and that the patterns were assumed to be a non-evolving set, mostly defined with a top-down approach. One recent methodology that explicitly mentions the use of patterns is presented by Maas et al. in [MJ09], which is a methodology specifically tailored to ontologies for ambient intelligence. However, the methodology is quite general and resembles the XD in its outline, although there is no mention of general principles (see the XD principles later in this chapter). Additionally, this methodology is quite focused on the identification of user requirements and the matching of those requirements to the ODPs, e.g. introducing a notion of informal patterns (called prototypical ODPs) as a step between user requirements and formally described ODPs.

For ontology evaluation and debugging, Content ODPs have been used during the evaluation of ontologies in the Fisheries use-case in WP7. Related work, such the one by Corcho et al. in [CRBP09], has also proposed the use of Logical ODPs, and even anti-patterns, for ontology debugging tasks.

2.1.2 Background and principles

XD is partly inspired by software engineering's eXtreme Programming (XP) (see Shore et al. in [SW07]), and experience factory (von Wangenheim et al. in [vWAB99] and Basili et al. in [BCR94]). The former is an agile software development methodology aiming to minimize the impact of changes at any stage of the development, and producing incremental releases based on customer requirements and their prioritization. The latter is an organizational and process-oriented approach for improving life cycles and products based on the exploitation of past experience and know-how. The methodology of XP (also see Beck in [Bec99] for a comprehensible summary of the XP method and its practices) has evolved over the last decade, as part of the agile software development movement. As mentioned above, the main idea of agile software development is to be able to incorporate changes easily, at any stage of the development. Instead of using a waterfall-like method where you first do all the analysis, then the design, the implementation and finally the testing, the idea is to cut this process into small pieces, each containing all those elements but only for a very small subset of the problem. The solution will grow almost organically and there is no grand plan that can be ruined by a big change request from the customer.

The XD methodology, especially the version for reusing Content ODPs, is a result of the observation and consequent description of the way we (in our research lab and in the NeOn project in general) have developed ontologies with Content ODPs. Since 2005, we have been developing Content ODPs, teaching pattern-based ontology design in conference tutorials and PhD courses, and for much longer we have been using and refining this approach for our professional work. In order to teach pattern-based design to PhD students and practitioners, we needed to provide trainees with guidelines to follow. This requirement provided us with a good occasion for defining the XD method, primarily for Content ODPs, and also with a context for running the method with different teams, and applying possible refinements and adjustments. During the development of the overall NeOn methodology in WP5 we have then formalized the guidelines and the methodology outline, as well as performed experiments (see D5.6.2 [DSFGP+09]), mainly on the specific methodology for Content ODP reuse. The analysis of those experiments in combination with our previous experience has led up to the current version of the methodology.

Principles

The XD method is inspired by XP in many ways, but its focus is different: where XP diminishes the value of careful design, this is exactly where XD has its main focus. Of course, designing software and designing ontologies is inherently different, but still there are many lessons to be learnt from programming. XD is test-driven, and applies the divide-and-conquer approach, similarly to XP. Also, XD adopts pair design, in analogy to pair programming. Although we did not yet perform a formal evaluation of the effectiveness of pair design, we have collected feedback of trainees and developers through informal discussions and questionnaires after the execution of XD with different teams. Most of them feel that they benefit from on-the-fly brainstorming, and perceive to improve the effect of learning-by-doing within the pair design setting. We have planned to conduct more rigorous evaluations which also involve the analysis of this aspect.

The intensive use of ODPs, modular design, and collaboration are the main principles of the methodology. The effectiveness of ODPs, in particular Content ODPs, in ontology design has been rigorously evaluated in D5.6.2 [DSFGP+09], where XD has been used as development guidelines. The main principles of the XD methodology can be summarized as follows:

- **Customer involvement and feedback.** The development of an ontology is part of a bigger picture, where typically a software project is under development. Ideally, the customer should be involved in the ontology development and the customer representation should be a team, whose members are aware of all parts and needs of the project. For example, the roles that should be represented include: domain experts, i.e. persons with deep knowledge of the domain to be described by the ontology, those who are in charge of maintaining the knowledge and data sets, i.e. persons who know the views over the data that are usually required by users, those who control and coordinate organizational processes, i.e. persons who have an overall view on the entire flow of data. Depending on the project characteristics, and on the complexity of the organization, the customer representative can be one person or a team. It is important that the team of designers is able to easily interact with the customer representative in order to minimize the possible number of assumptions that they have to make on the incomplete requirement descriptions, i.e. assumptions on the implicit knowledge without discussing and validating them first. Interaction with the customer representative is a key success factor for favoring the explicit expression of knowledge that is usually implicit in requirement documents, including competency questions (CQs). Furthermore, the customer representative should be able to describe what tasks the application involving the ontology is expected to solve.

- **Customer stories, Competency Questions (CQs), and contextual statements.** The ontology requirements and its tasks are described in terms of small stories by the customer representative. Designers work on those small stories and, together with the customer, transform them into the form of CQs and contextual statements. Contextual statements are accompanying assertions that explicitly state knowledge that is typically implicit in CQs, e.g. restrictions such as relational cardinality. CQs and contextual statements will be used throughout the whole development process, and their definition is a key phase as the designers have the challenge to help the customer in making explicit as much implicit knowledge as possible.

- **ODP reuse and modular design.** If there is a GUC of an ODP that matches a LUC it should be reused. Otherwise a new ODP, with its GUC defined based on the LUC in question, should be developed and shared with the team (and ideally on the Web[1]). In the creation of ODPs certain principles apply (see Gangemi in [GP09b]) concerning the characteristics of the constructed ODP, e.g. level of generality, size, and cognitive comprehensiveness. For the specific case of Content ODPs the reuse of the ODPs as OWL building blocks introduce a natural way of modularizing the ontology, i.e. based on modelling issues. The solution for each CQ, or a group of related CQs, will become a module of the resulting ontology. Such modules, if general and reusable, could also be considered as Content ODP candidates.

- **Collaboration and Integration.** Integration is a key aspect of XD as the ontology is developed in a modular way, using a divide-and-conquer paradigm. Collaboration and constant sharing of knowledge is needed in an XD setting, in fact similar or even the same CQs and sentences can be defined for different stories. When this happens, it is for instance important that the same ODP is reused, or else the ontology might require some refactoring when the modules are integrated.

- **Task-oriented design.** The focus of the design is on a specific part of the domain of knowledge under investigation that is needed in order to address the user stories, and more generally, the tasks that the ontology is expected to address. This is opposed to the more philosophical approach of formal ontology design where the aim is to be comprehensive with respect to a certain domain. Similarly to XP, XD proposes to provide solutions to the exact requirements stated, nothing less but also nothing more; in the sense that unnecessary knowledge should be left out, and the concepts should be defined exactly according to the intended task of the ontology rather than in some common sense notion of their “true" nature.

- **Test-driven design.** Stories, CQs, and contextual statements are used in order to develop unit tests. A new story has been realized only when all unit tests associated with it have been passed. This aspect enforces the task-oriented approach of the method. It has to be noticed that in this context, unit tests have a completely different meaning with respect to software engineering unit tests. An ontology module developed for addressing a certain user story associated to a certain competency question, is tested e.g. (i) by encoding in the ontology a sample set of facts based on the user story, (ii) defining one or a set of SPARQL queries that formally encode the competency question, (iii) associating each SPARQL query with the expected result, and (iv) running the SPARQL queries against the ontology and comparing actual with expected results. Unit tests for ontologies have already been analyzed by Vrandecic et al. in [VG06], however, the focus was more on purely logical structures.

- **Pair design.** The team of designers is organized in pairs. This principle is analogous to the pair programming of XP, but while pair programming has proven efficient in software development it still remains to formally prove the efficiency for ontology engineering. Currently this has to be considered a hypothesis, based on experience and observations made so far. At least one pair is in charge of integrating ontology modules, and if necessary refactoring the resulting model.

2.1.3 The eXtreme design iteration

The XD ODP reuse method is illustrated in Figure 2.2. The process starts by identifying the set of requirements to be addressed, possibly not the complete set of requirements in the ontology requirements specifi-

[1] For example on http://www.ontologydesignpatterns.org

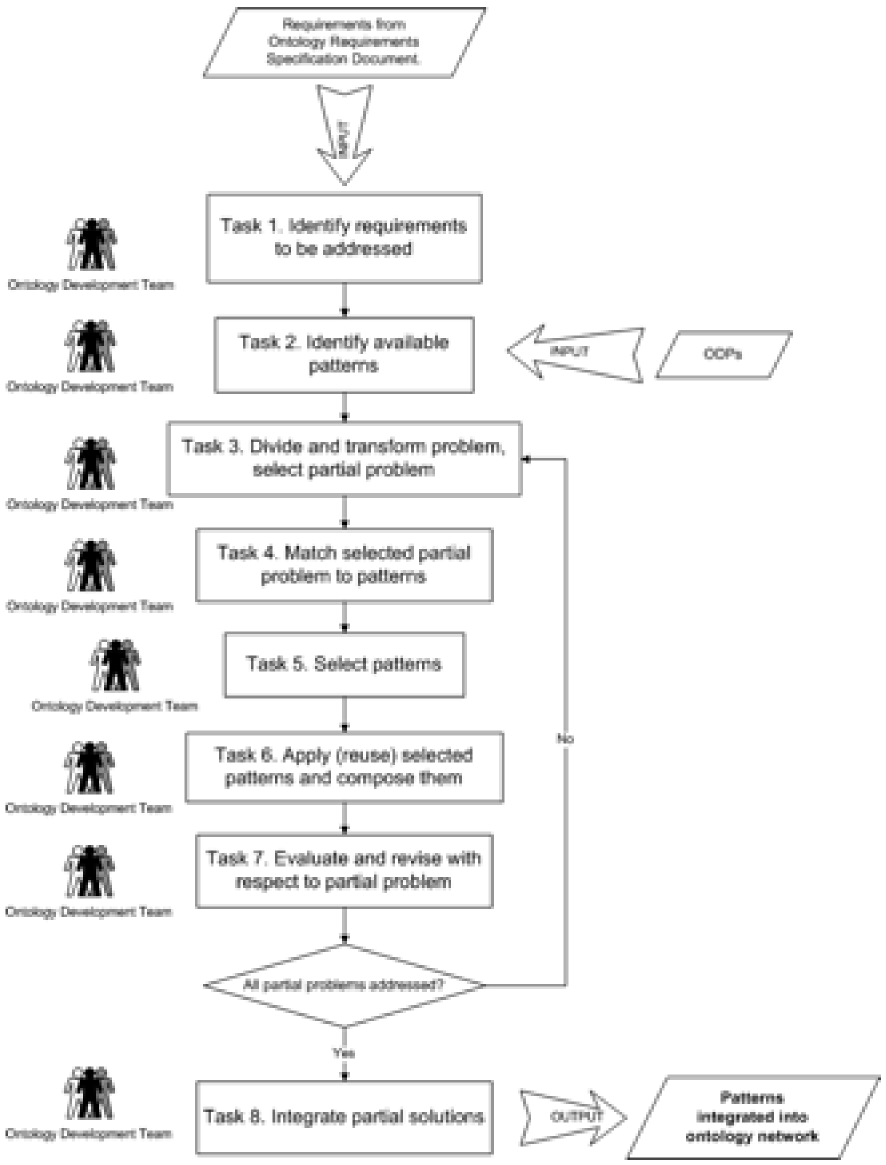

Requirements from Ontology Requirements Specification Document
INPUT
Task 1. Identify requirements to be addressed
Ontology Development Team
Task 2. Identify available patterns
INPUT
ODPs
Task 3. Divide and transform problem, select partial problem
Task 4. Match selected partial problem to patterns
Task 5. Select patterns
Task 6. Apply (reuse) selected patterns and compose them
Task 7. Evaluate and revise with respect to partial problem
All partial problems addressed?
No
Yes
Task 8. Integrate partial solutions
OUTPUT
Patterns integrated into ontology network

Figure 2.2: Overview of the XD method, from D5.4.2 [SBD+09].

cation document (ORSD) will be addressed using ODPs, and a set of patterns (catalogues of patterns) that are available for reuse. Throughout the process a divide-and-conquer paradigm is used.

For each sub-problem, it is matched to the patterns (a search of the solution space) and some appropriate patterns are selected and reused. Depending on the nature of the ODPs, e.g. if they are Content ODPs or Re-engineering ODPs, the matching uses different methods, e.g. for Content ODP reuse the specific requirement CQs are matched against the abstract CQs of the Content ODPS, while for Re-engineering ODPs the matching is done based on available non-ontological resource types, syntactical representation of those resources, and the desired output of the re-egnieering process. The pattern-based methodology is test-driven in the sense that each small solution is tested against the requirements before integrating it with the rest of the solution. More detailed guidelines for each step can be found in D5.4.2 [SBD+09].

2.1.4 The XD Workflow with Content Ontology Design Patterns

A previous version of this methodology, including detailed guidelines and examples, was presented in D5.4.2 [SBD+09]. The methodology presented here is an improved version, based on recent experiences and the

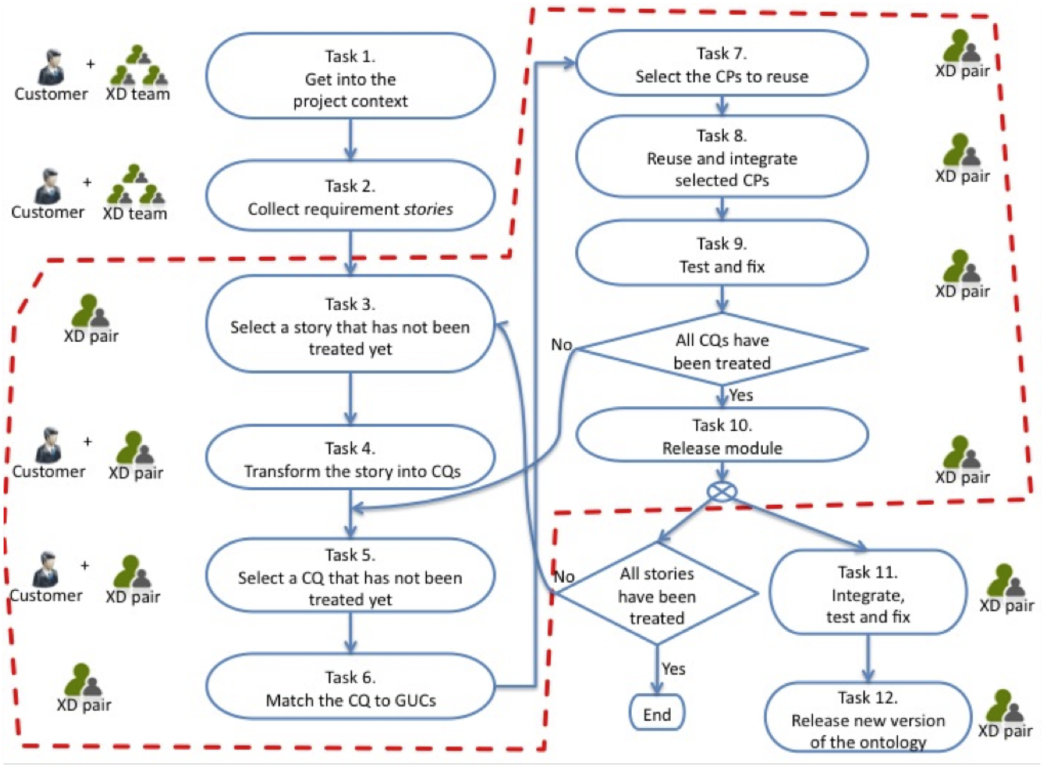

Customer + XD team
Task 1. Get into the project context
Customer + XD team
Task 2. Collect requirement stories
XD pair
Task 3. Select a story that has not been treated yet
Customer + XD pair
Task 4. Transform the story into CQs
Customer + XD pair
Task 5. Select a CQ that has not been treated yet
XD pair
Task 6. Match the CQ to GUCs
Task 7. Select the CPs to reuse
XD pair
Task 8. Reuse and integrate selected CPs
XD pair
Task 9. Test and fix
XD pair
No
All CQs have been treated
Yes
Task 10. Release module
XD pair
No
All stories have been treated
Yes
End
Task 11. Integrate, test and fix
XD pair
Task 12. Release new version of the ontology
XD pair

Figure 2.3: XD workflow for Content ODP reuse.

experiments performed in WP5 (reported in D5.6.2 [DSFGP+09]). Main improvements are that the initial and the final parts of the methodology have been detailed further (in the previous version the focus was mainly on the iterations by the design pairs), and the order of the steps slightly modified in order to better support the integration and collaboration of the design pairs.

Figure 2.3 shows the workflow of XD with Content ODPs. In this section we will describe the single tasks with the help of a simplified scenario coming from a real case study in the Fishery domain. XD is an incremental, iterative method for pattern-based ontology development. So far detailed guidelines and partial tool support exist for the tasks within the dashed line in Figure 2.3. Guidelines for the initial steps, and the integration and release of new ontology versions, has so far not been properly established through experimental validation of our hypotheses. The guidelines presented here should be read with this in mind.

Before considering the details of each single task, it is worth noting a few assumptions. The team of designers is organized in pairs that work in parallel. At least one pair is in charge of integrating the modules produced by the other pairs, in order to obtain incremental releases of the ontology. A wiki for the project is set up with a basic structure able to collect customer stories and their associated modeling choices, testing documentation, and contextual statements. The wiki will be used in order to incrementally build the project documentation. During the development, and in particular for testing purposes, an ontology module containing instances according to the customer stories is created and shared. This module is used in order to run unit tests against the ontology.

Task 1. Get into the project context. The development is kicked-off by a group of ontology developers and a group of domain experts i.e. the customer representative. In principle they do not know much about each other; they do not have a precise idea of what will be the result of the project, they are used to different terminology, and have different backgrounds. This first task has a twofold objective: (i) make the customer representative aware of the methods and tools that will be applied during the development project, (ii) provide the ontology designer team with an overview of the problem from a domain expert perspective, its scope, and agree on initial terminology. The result of this task is the setup of a collaborative environment where the customer representative and ontology designers will share documentation and collect arguments and motivations about modeling issues, including terminology, e.g. through deploying a wiki for the project.

Task 2. Collect requirement stories. The customer representative is invited to write stories, possibly from real, documented scenarios, that sample the typical facts that should be stored in the resulting ontology. All stories are organized in terms of priority, and possible dependencies between them are identified and made explicit. Each story is described by means of a small card, like the one depicted in Table 2.1, which includes the story's title, a list of other stories which it depends on, a description in natural language, and a priority value. It is important to notice that this task is not intended to be performed only once during the project. Stories can be added by the customer during the whole project life cycle. For example, if a new requirement emerges new stories can be written. In case the project already has an ontology requirements description document (c.f. D5.4.2 [SBD$^+$09]) with clear CQs, these can instead be enriched by attaching them to such stories, otherwise this process can be seen as an alternative method for collecting CQs (see next steps below).

Task 3. Select a story that has not been treated yet. Each pair of designers selects a story that will be the focus of their work for the next iteration. The selection is based on the experience and competencies of the design pair, but also on the priority of the story, i.e. what the customer wants to be realized in this iteration. A new wiki page for the story is created: the name of the page is the title of the story, and its content is set up based on the information that is present on the card. By performing this task a pair enters a development iteration. To illustrate the process we assume that a pair has selected the story described by the card in Table 2.1.

Table 2.1: A requirement story card. It includes the story's title, a list of other stories which the story depends on, a natural language description, and a priority value.

Title	Tuna observation
Depends on	Exploitation values, Tuna areas
Description	In 2004 the resource of species 'Tuna' in water area 24 was observed to be fully exploited in the tropical zone at pelagic depth.
Priority	High

Task 4. Transform the story into CQs. The pair processes the story and derives a set of CQs from it. In order to do that, designers could involve the customer for having feedback and clarifications. In case CQs are already present (c.f. D5.4.2 [SBD$^+$09]) this step will only involve to check the CQs for coverage, against the stories collected earlier. However, in case CQs need to be developed, first the story is split into simple sentences, meaning that complex example sentences may be broken up into shorter sentences to increase clarity. The sentences are abstracted so that they describe a class of facts instead of a specific one. The sentences are then transformed into CQs. For example, the story *Tuna observation* is transformed to the following CQs, which are added to the story's wiki page:

- CQ_1 : What are the exploitation state and vertical distance observed in a given climatic zone for a certain resource?
- CQ_2 : What resources have been observed during a certain period in a certain water area?

Additionally, the following contextual statement is derived from the discussion with the customer representative:

- A resource contains one or more species.
- Species are associated to vertical distances. As a consequence, the vertical distance of a resource is inferred through the vertical distance of the species.

Contextual statements are listed in a dedicated wiki page, and are mainly handled by the pair in charge of the integration task.

Task 5. Select a CQ that has not been treated yet. The iteration continues by selecting one of the CQs, or a small set that constitutes a coherent modelling task and should be treated together. In our example this could mean to select CQ_1. Sometimes it is more intuitive for the developers to select a set of coherent CQs, treating the same concepts, rather than treating each CQ individually. However, the main idea is that the problem now should be broken down into its smallest components, i.e. the individual modelling issues. The term *modelling issue* here denotes a primitive modelling problem, such as modelling an n-ary relation, e.g. the observation relation in CQ_1.

Task 6. Match the CQ to GUCs. This task has the aim of identifying a set of candidate Content ODPs based on the CQ(s), which express part of the LUC, i.e. the output Content ODP set is the basis for the Content ODP selection performed in Task 7. The matching procedure can be done either with some tool support e.g., keyword based search, or manually e.g., if the designers have a good knowledge of available Content ODPs. We here assume that designers manually perform the matching against the ontologydesign-patterns.org repository of Content ODPs. In our example, candidate Content ODPs are (for CQ_1 and CQ_2 above): situation, and time interval. All of which are available on http://ontologydesignpatterns.org (see Section 3.2). The competency question of situation - “What entities are in the setting of a certain situation?” - can be said to match the observation, the resource, and the parameters that are in the setting of that observation. Additionally, the time interval Content ODP may be seen as partially matching the question of what period a certain observation was made, although this could also be solved with just a simple datatype property in case, for example, a year is sufficient to denote the period. The Content ODP contains CQs such as: “What is the end time of this interval?”, “What is the starting time of this interval?”, “What is the date of this time interval?”. The result of this task is then two matching Content ODPs.

Task 7. Select the Content ODPs to reuse. The goal of this task is to select which of those patterns should be used for solving the modeling problem. We may decide that time interval adds too much extra effort, besides the needed year of observation (decided in cooperation with the customer representative), in which case we will only select situation.

Task 8. Reuse and integrate selected Content ODPs. The term ’reuse’ here refers to the application of typical operations that can be applied to Content ODPs, i.e. import, specialization, and composition (see D2.5.1 [PGD$^+$08]). In our example, we specialize the *situation* Content ODP in order to address CQ_1. The particular situation is in our case the observation, and the thing observed is the resource. Additionally, the exploitation state, climatic zone, and vertical distance of the observation, are also involved in the setting. Thereby, we add a subclass of situation:Situation named AquaticResourceObservation, and add the other entities as subclasses of owl:Thing. In addition, we construct subproperties of the situation:isSettingFor and its inverse situation:hasSetting, for connecting the observations to the resources and the different parameters. In this case we have discussed a simplified example where only one Content ODP has been reused and specialized. In other cases, we might reuse more Content ODPs. Each of them would first be specialized then integrated with the others. The process that is typically performed during this task consists of the steps:

1. Select a Content ODP that has not been integrated yet.
2. Specialize the pattern.
3. Compose: identify elements and axioms of the specialized pattern that should be aligned with the current module being produced, and align them.

After iterating over all the selected Content ODPs, and integrating them into the current module, the module also has to be extended to cover the complete CQ(s). In our example, no pattern was selected to solve the time period issue in CQ_2, hence a datatype property has to be added to the module in order to cover the complete CQ.

Task 9. Test and fix. The goal of this task is to validate the resulting module with respect to the CQ(s) just modeled. To this aim, the task is executed through the following steps: (i) the CQ(s) are elaborated in order to derive a unit test e.g., one or more SPARQL queries; (ii) the instance module is fed with sample facts based on the story; (iii) the unit test is ran against the ontology module (importing the instance module). If the result is not the expected one, i.e. the test is not passed, the module is revised in order to fix the problem, and the unit test ran again until the test is passed; (iv) run all other unit tests associated with the story so far until they all pass. Notice that all unit tests are described in dedicated wiki pages that are properly linked to the associated story. If all CQs associated to the story have been addressed, the process continues with Task 10, otherwise the pair goes back to Task 5. In our example, the unit test associated to CQ_1 is the following:

```
SELECT ?exp ?dist ?resource ?zone
WHERE {
?obs a :AquaticResourceObservation .
?obs observedResource ?resource .
?obs inClimaticZone ?zone
?obs inState ?exp .
?obs atVerticalDistance ?dist
}
```

Task 10. Release module. This task identifies the end of an iteration for a pair and the result is an ontology module. Once the whole story has been addressed, and the resulting module has been successfully tested, the new module can be released. The module is assigned a URI and published in order to be shared by the whole team. If the module can be publicly shared, it can be published in open Web registries such as ontologydesignpatterns.org. The module is then passed to the pair in charge of the integration. The pair of designers selects a new story if there are still some unaddressed (see Task 3).

Task 11. Integrate, test and fix. Once a new module is released, it has to be integrated with all the others that constitute the current version of the ontology. At least one pair is in charge of performing integration and related tests; new unit tests are defined for the integration, and all existing ones are again executed as regression tests before moving to next task. In this task, all contextual statements are taken into account and all necessary alignment axioms are defined. The module is now under the complete control and editing of the pair in charge of the integration, and refactoring of the ontology modules may be performed in case inconsistent modelling choices are discovered. The products of this task are new unit tests and alignment axioms, and possibly a set of changes to the ontology modules (results of refactoring), all properly documented in the wiki.

Task 12. Release new version of the ontology. Once all unit tests have been passed, a new version of the ontology can be released. The ontology is given a new version number, and it is associated to its own version of the wiki documentation.

2.2 The role of Re-engineering Ontology Design Patterns

In [SFDMP+08, VTAGS+08], we already introduced a method for re-engineering non-ontological resources. We improved and refined the method in [ALVT09]. The method is included in the framework of the NeOn Methodology. It is related to the XD method family in the sense that it heavily makes use of ODPs, namely Re-engineering ODPs, however, it has so far not been properly aligned to the XD workflow presented previously in Section 2.1.3. It is included in this deliverable as a separate method, showing the role of Re-engineering ODPs in the overall NeOn methodology.

The method relies on a repository of Patterns for Re-engineering Non-Ontological Resources (PR-NORs). We can consider these patterns as Re-engineering Ontology Design Patterns, Re-engineering ODPs. In this research work, Re-engineering ODPs 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's underlying data model. The data model defines how the different components of the non-ontological resource are represented [GSGPSFVT08].

We will present a summary of our method for re-engineering non-ontological resources into ontologies in Section 2.2.1 of this deliverable. Then we briefly sketch the Re-engineering Ontology Design Patterns in Section 2.2.2. Finally, in Section 3.3 we will present a software library that implements the transformation process suggested by the Re-engineering ODPs.

2.2.1 Method for re-engineering non-ontological resources

Non-ontological resources (NORs), which were defined in D5.4.1 [SFDMP+08], are knowledge-aware resources whose semantics have not been formalized yet by an ontology. There is a large 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. For example, 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.

With the aim of speeding up the ontology development process, by re-engineering non-ontological resources, in [SFDMP+08, VTAGS+08, ALVT09] we proposed a method that consists of three activities:

1. ***Non-Ontological Resource Reverse Engineering***. The goal of this activity 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) [ALVT09].

2. ***Non-Ontological Resource Transformation***. The goal of this activity is to generate a conceptual model from the non-ontological resource. We propose the use of Re-engineering ODPs to guide the transformation process. To this end, the ontology development team has to find out if there is any applicable Re-engineering ODP(s) to transform the non-ontological resource into a conceptual model. 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 transformation approach has to be selected, e.g. from the ones described in [ALVT09].

3. ***Ontology Forward Engineering***. The goal of this activity is to generate the ontology. We use the ontology levels of abstraction, described in [ALVT09], to depict this activity because they are directly related to the ontology development process.

In [ALVT09] we described a user based evaluation on using the method for re-engineering non-ontological resources into ontologies. The goal of this evaluation is to gain evidence on whether the usage of the method leads to users being able to design ontologies faster and/or better quality standards.

2.2.2 Re-engineering ODPs

In this Section we briefly present our *Re-engineering ODPs*. The patterns are described in detail in NeOn deliverables D2.2.2 [VTAGS+08] and D2.2.4 [ALVT09]. These patterns come from the experience of ontology engineers in developing ontologies using non-ontological resources in several projects (e.g., SEEMP[2], and Knowledge Web[3] as well as the NeOn project itself). The patterns describe how to generate the ontologies at a conceptualization level, independent of the ontology implementation language.

The use of Re-engineering ODPs for transforming non-ontological resources into ontologies has several advantages, for instance, they:

[2]`http://www.seemp.org`
[3]`http://knowledgeweb.semanticweb.org`

- embody expertise about how to guide a re-engineering process,
- improve the efficiency of the re-engineering process,
- make the transformation process easier for both ontology engineers and domain experts,
- improve the reusability of non-ontological resources.

These patterns 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's 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 D2.2.2 [VTAGS+08], 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 Re-engineering ODP.

The Re-engineering ODPs must disambiguate/discover the semantics of the relations among the non-ontological resource elements. To perform this disambiguation, patterns rely on external resources. In cases where the external resource gives an empty result, i.e. it does not provide any relation between two concepts, the patterns take advantage of the use of Logical or Content ODPs for asserting relations such as *partOf* or *subClassOf*.

Currently, as external resources, patterns are relying on (1) Scarlet[4], which is a technique for discovering relations between two concepts by making use of ontologies available online, and (2) WordNet[5], which is an electronic lexical database created at Princeton University. The relation disambiguation phase consists in

1. Take two related terms from the non-ontological resource.
2. Search for a relation between those terms by using Scarlet.
3. Assert the relation returned by Scarlet for the two terms.
4. If Scarlet gives an empty result
   - 4.1. Search for a relation between the two terms by using WordNet. The patterns deal with the following WordNet relations:
     - hyponym relation, interpreted as *subClassOf*.
     - hypernym relation, interpreted as *superClassOf*.
     - member meronym relation, interpreted as *partOf*.
     - member holonym relation, interpreted as *hasPart*.
   - 4.2. Assert the relation returned by WordNet for the two terms.
   - 4.3. if WordNet gives an empty result, assert the *partOf* or *subClassOf* relation as appropriate.

Next, we present the proposed template used to describe the Re-engineering ODPs. To present the patterns we have modified the tabular template used in [VTAGS+08]. The adapted template and the meaning of each field is shown in Table 2.2.

Table 2.2: Re-engineering ODP Template

Slot	Value
General Information	

[4] http://kmi.open.ac.uk/technologies/name/Scarlet
[5] http://wordnet.princeton.edu/

Table 2.2: Re-engineering ODP Template (Continued)

Slot	Value
Name	Name of the pattern
Identifier	An acronym composed of component type + abbreviated name of the component + number
Component Type	Re-engineering ODP
Use Case	
General	Description in natural language of the re-engineering problem addressed by the Re-engineering ODP.
Example	Description in natural language of an example of the re-engineering problem.
Re-engineering ODP	
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.
Formal Transformation	
General	Formal description of the transformation by using the formal definitions of the resources.
Relationships (Optional)	
Relations to other modelling components	Description of any relation to other Re-engineering ODPs or other ontology design patterns.

So far, we have the following patterns:

- Patterns for Re-engineering Classification Schemes into Ontologies
  - PR-NOR-CLTX-01. Pattern for re-engineering a classification scheme, which follows the path enumeration data model, into an ontology schema.
  - PR-NOR-CLTX-02. Pattern for re-engineering a classification scheme, which follows the adjacency list data model, into an ontology schema.
  - PR-NOR-CLTX-03. Pattern for re-engineering a classification scheme, which follows the snowflake data model, into an ontology schema.
  - PR-NOR-CLTX-04. Pattern for re-engineering a classification scheme, which follows the flattened data model, into an ontology schema.
  - PR-NOR-CLLO-10. Pattern for re-engineering a classification scheme, which follows the path enumeration data model, into an ontology.

  - PR-NOR-CLLO-11. Pattern for re-engineering a classification scheme, which follows the adjacency list data model, into an ontology.
  - PR-NOR-CLLO-12. Pattern for re-engineering a classification scheme, which follows the snowflake data model, into an ontology.
  - PR-NOR-CLLO-13. Pattern for re-engineering a classification scheme, which follows the flattened data model, into an ontology.

- Patterns for Re-engineering Thesauri into Ontologies
  - PR-NOR-TSLO-01. Pattern for re-engineering a term-based thesaurus, which follows the record-based data model, into an ontology schema.
  - PR-NOR-TSLO-02. Pattern for re-engineering a term-based thesaurus, which follows the relation-based data model, into an ontology schema.
  - PR-NOR-TSLO-11. Pattern for re-engineering a term-based thesaurus, which follows the record-based data model, into an ontology.
  - PR-NOR-TSLO-12. Pattern for re-engineering a term-based thesaurus, which follows the relation-based data model, into an ontology.

Additionally, in the context of NeOn *D2.4.4 An integrated model for linguistic/terminological resources and ontologies*, we are proposing patterns for re-engineering lexica into ontologies. This set of patterns will soon be included at the Ontology Design Pattern Portal[6], in the Re-engineering Catalogue[7] Section. The patterns already present were described in detail in NeOn deliverables D2.2.2 [VTAGS$^+$08] and D2.2.4 [ALVT09]

2.3 The role of Lexico-syntactic Patterns

In [SFDMP$^+$08], we already introduced a method for the reuse of Ontology Design Patterns (ODPs) aimed at newcomers to Ontology Engineering. This method is based on a repository of Lexico-syntactic patterns, Lexico-syntactic ODPs, associated to other ODPs with the aim of allowing a semi-automatic reuse of ODPs taking as a starting point the formulation of the modelling problem in Natural Language (NL). This method can be used to support for example the selection and reuse of Content ODPs, as was already described in D5.4.1 [SFDMP$^+$08]. In this deliverable the Lexico-syntactic ODPs and the associated methods are described as separate methods, although they can be used to support tasks within the XD workflow (as described in 2.1.3) and hence describe the role of Lexico-syntactic ODPs in this context.

In this research work, Lexico-syntactic ODPs are understood as *formalized linguistic schemas or constructions derived from regular expressions in NL that consist of certain linguistic and paralinguistic elements, following a specific syntactic order, and that permit to extract some conclusions about the meaning they express* [CGPMPSF08]. Our approach to derive Lexico-syntactic ODPs from NL expressions is based on the assumption that any language has a number of lexical and/or syntactical mechanisms to reliably convey a relation of interest, which in this case is the one represented by the ODP.
At this stage of the research we have only considered verb-oriented Lexico-syntactic ODPs that are mainly composed by tuples of the form subject-verb-object, although we also consider compound sentences, i.e., sentences in which more than one verb is involved. We assume that for expressing how concepts are related in ontologies, we make use of verbs in affirmative or declarative sentences in the simple present tense. In this kind of pattern, verbs are mostly the ones that carry the semantics of the relation. It is also worth mentioning that Lexico-syntactic ODPs are language dependent, since each NL relies on different linguistic and morphosyntactical mechanisms to convey semantics. Our aim thus is to provide Lexico-syntactic ODPs for different NLs (currently English and Spanish), since the purpose of the approach is to assist novice users in their own languages.

[6] http://ontologydesignpatterns.org
[7] http://ontologydesignpatterns.org/wiki/Submissions:ReengineeringODPs

The original method proposed in D5.4.1 [SFDMP+08] has undergone some minor modifications that are reported in Section 2.3.1 of this deliverable. Then, we briefly describe the approach within the wider framework of the NeOn Methodology. A very preliminary version of the repository of Lexico-syntactic ODPs was published in D2.5.1[PGD+08]. That first Lexico-syntactic ODP repository contained Lexico-syntactic ODPs in English associated to the other ODPs (e.g. Content ODPs) they corresponded to. A second version of the Lexico-syntactic ODP respository that contains Lexico-syntactic ODPs in English and Spanish is presented in appendix A. Note that the repository is in a more advanced stage for English and in a more initial one for Spanish. Finally, in Section 3.4, we describe how the set of English patterns contained in the Lexico-syntactic ODP repository have been implemented in the GATE Architecture [8] [CMB+09] to enable a semi-automatic identification of corresponding ODPs.

2.3.1 Method for the reuse of ODPs starting from NL formulations

With the aim of helping untrained ontology designers to reuse ODPs, e.g. Content ODPs, in [SFDMP+08] we proposed a method that consisted of three tasks:

1. **Task 1. ODP Formulation.** The goal of this task is to formulate, in full NL, the domain aspect to be modeled: the user has difficulties in solving a particular modelling issue within a certain domain, and expresses that knowledge in NL.

2. **Task 2. ODP Refinement.** The goal of this task is to refine the input from Task 1. This task is only carried out when there is no direct correspondence to one ODP or a combination of ODPs. Different strategies may be adopted with this aim, such as, user interaction with the system, search in external ontologies or lexicons, etc. The reasons for refinement may be related to ontology enrichment needs or lexical ambiguities.

3. **Task 3. ODP Validation.** The goal of this task is to confirm that the resulting ODP, or ODPs, meet user expectations.

Since this is an approach involving NL, the method needs to rely on NLP tools, as explained below. Besides, this method has to be understood within the wider framework of the NeOn Methodology. In this sense, we assume that users have been following the NeOn Methodology for the development of their ontologies, and that they have already performed the Specification Activity (see [SBD+09]). As a result of this activity, users obtain the Ontology Requirements Specification Document (ORSD) in which the requirements to be satisfied by the ontology are expressed. Of particular interest are here the so-called Competency Questions (CQs). CQs are the starting point of this method, as also suggested by the method for Ontology Design Patterns Reuse aimed at Ontology Engineers in [SBD+09] (the initial version of the XD Content ODP methods described previously in this deliverable). Figure 2.4 illustrates the interaction of the main methodological and technological components of this approach.

Relying on the set of CQs novice users are asked to formulate in NL the domain aspect (Step 1), i.e. the modelling issue in terms of domain knowledge, they aim at modelling in the ontology (as proposed in Task 1 of the method). Initially, the idea behind this method was to allow users to freely introduce sentences in NL without restrictions, with the aim of palliating the difficulties imposed by controlled languages (for more on this see [CGPMPSF08, dCMPSF09]). However, after some initial tests with real users, we realized that some advice or recommendations were needed to guide users in the formulation task. Note that the user is assumed to introduce correct information from the content viewpoint, moreover taking into account that CQs have been previously validated. Therefore, with the aim of helping or guiding users in the formulation of sentences in NL, we have provided some recommendations accompanied with examples in NL. See table 2.3.

[8] http://gate.ac.uk/

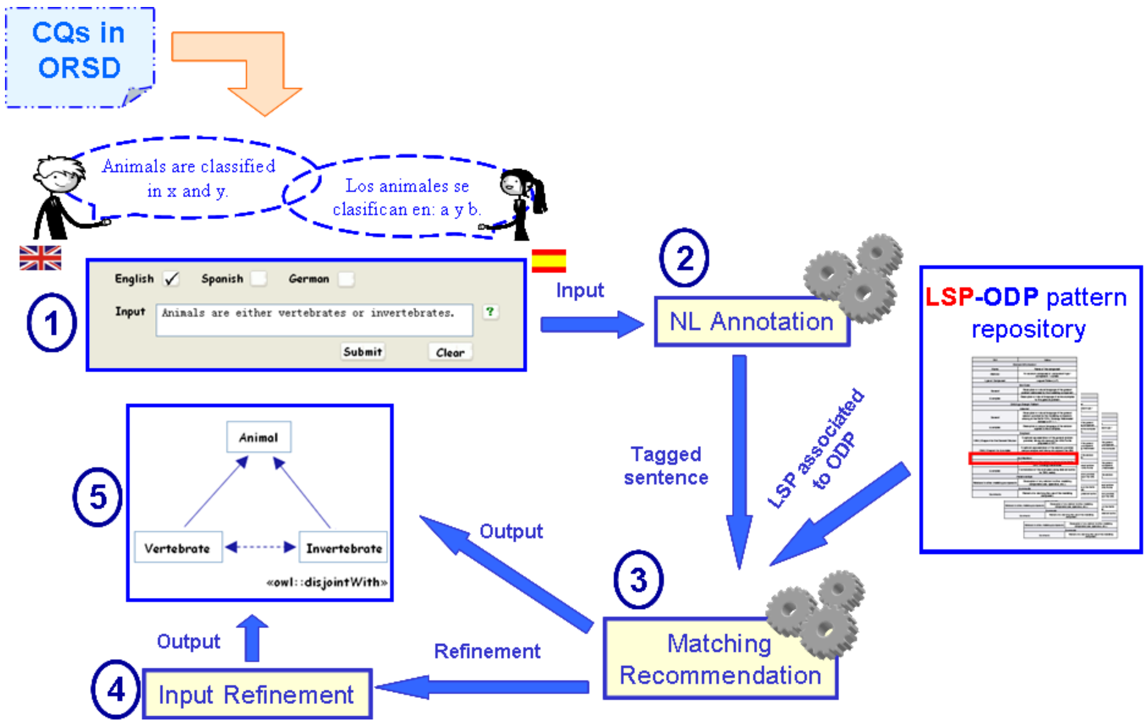


Figure 2.4: Approach for the reuse of ODPs, aimed at novice users, by exploiting Lexico-syntactic ODPs.

The following step (step 2 in Figure 2.3) consists of processing the input sentences with NLP tools. This is performed with GATE, the General Architecture for Text Engineering developed by our colleagues from the University of Sheffield. With this aim, the Lexico-syntactic ODPs identified so far for the English language have been implemented in the JAPE language, as detailed in Section 3.4. Once the input sentence has been annotated with GATE, the result is compared against the repository of Lexico-syntactic ODPs to find correspondences (step 3 in Figure 2.3). As can be seen in appendix A, correspondences between Lexico-syntactic ODPs and other ODPs are not always direct or 1:1 correspondences. Sometimes a Lexico-syntactic ODP corresponds to a combination of other ODPs, because the same linguistic structure conveys the information encoded in various other ontological structures. There is still a third possibility, namely, the correspondence of one Lexico-syntactic ODP to pairwise disjoint ODPs. This means that the information conveyed by the same linguistic structure can have various modelling alternatives, which are not necessarily compatible. This can be summarized as follows:

- 1:1 correspondence: 1 Lexico-syntactic ODP - 1 ODP
- 1:N correspondence: 1 Lexico-syntactic ODP - combination of 2 or more ODPs
- 1:1/2 correspondence: 1 Lexico-syntactic ODP - pairwise disjoint ODPs

When the matching between the annotated sentence and the Lexico-syntactic ODP repository is 1:1 or 1:N, the system will identify the appropriate ODP, or ODPs, that solve the user's modelling issue and show the results, which is either a UML diagram instantiated with the information from the input sentece or the corresponding OWL code (step 5 in Figure 2.3). If, on the other hand, the third correspondence modality occurs, or no match at all is found, then a disambiguation or refinement task has to be performed (step 4 in Figure 2.3), concerning Task 2 of the proposed method. This situation may be due to an ambiguous or polysemic linguistic structure, or to the fact that different modelling decisions can be made to solve a certain modelling issue. In any case, a process of refinement is needed in order to obtain the ODP, or ODPs, that correspond to the input sentence. For this purpose different refinement strategies have been investigated (see [CGPMPSF08, MPdCGPSF08]).

Recommendations
1. Express one topic or idea per sentence. *Falls are types of incidents, which can happen in hospitals.* WRONG *Falls are types of incidents. Falls can happen in hospitals.* CORRECT
2. Include in each sentence subject, verb and object (SVO). (Do not use pronouns instead of nouns!) *They receive assistance.* WRONG *Patients receive assistance.* CORRECT
3. Avoid using neither interrogative nor negative sentences. *Chairs are not considered mobility aids.* WRONG *Mobility aids are walking sticks, walking frames, crutches, wheelchairs, walking tripods, callipers, orthotics, and prosthetic devices.* CORRECT
4. Avoid coordination of phrases, use only when necessary. *Falls are types of incidents, and can be caused by different factors or hazards.* WRONG *Falls are types of incidents. Falls are caused by different factors or hazards.* CORRECT
5. Avoid including redundant or unnecessary information that does not add new content to the idea. *According to many people, medications can cause falls.* WRONG *Medications can cause falls.* CORRECT
6. End each sentence with a full stop.
7. In enumerations, use commas to separate elements. *Examples of Fall Minimation Strategies are restraints, safety devices, protocols, intervention, and procedures.* CORRECT

Table 2.3: Recommendations for Task 1.

Finally, the user is asked to validate the obtained result. Since this approach is aimed at novice users, the returned ODP, or OWL code, should be accompanied by an explanation in NL of the modelling possibilities offered by the matched ODP.

2.3.2 Second version of the Lexico-syntactic ODP repository

At this stage of the research, we have identified a set of Lexico-syntactic ODPs from NL expressions in English and Spanish. Lexico-syntactic ODPs are considered to be language dependent and not interchangeable among different NLs, despite some overlap. In order to describe Lexico-syntactic ODPs in a systematic way, we have designed a template that consists of four slots, as shown in Table 2.4. The information contained in each table refers to:

- **Identifier of the Lexico-syntactic ODP.** This mandatory slot contains an acronym composed of: LSP (Lexico-syntactic Pattern), plus the acronym of the relation captured by the ODP, plus the ISO-639 code for representing the name of the language for which the Lexico-syntactic ODP is valid.
- **NeOn ODP Identifier.** This mandatory slot inherits the ODP identifier used in the NeOn ODP repository within D5.1.1 [SFBG$^+$]. If the pattern was not contained in that repository, an acronym is created following the same rules. Identifiers are composed of the component type (e.g. LP standing for Logical Pattern, or CP for Content Pattern), component (e.g. SC standing for SubClassOf), and number of the pattern (e.g. 01).
- **Formalization.** This mandatory slot includes the various Lexico-syntactic ODPs that express the relation contained in the corresponding ODP(s). Lexico-syntactic ODPs have been formalized according

to a BNF extension (see Table 2.5 for Symbols and Abbreviations created for this purpose).

- **Examples.** This optional slot shows some examples of sentences in NL that match the Lexico-syntactic ODP in question.

Table 2.4: Lexico-syntactic ODP template.

Lexico-syntactic ODP Identifier	An acronym composed of LSP + ODP component + ISO code for language
NeOn ODP Identifier	An acronym composed of component type + component + number
Formalization	Lexico-syntactic ODPs formalized according to BNF extension
Examples	Sentences in NL that exemplify corresponding Lexico-syntactic ODPs

Following the above described template, we present a number of Lexico-syntactic ODPs for English and Spanish in appendix A. As already described in [PGD$^+$08], the elements represented in the formalized patterns are considered to be necessary for identifying the relation of interest expressed by the pattern. The main elements of the formalized patterns are described in the following paragraphs. The rest of them have been described in Table 2.5.
A Noun Phrase (NP) is a phrase whose main word is a noun or a pronoun, and that is optionally accompanied by a set of modifiers, as for example, determiners, adjectives, etc. NPs represent the arguments of a predicate, which in ontologies can be classes or properties. The semantic role attached to each NP in the patterns has been made explicit in angle brackets < >. Verbs expressing the conceptual relation in question are represented by its lemma, or base form. The elements represented in the pattern are the ones considered to be necessary for the pattern to express a certain relation. Optional elements, i.e. the ones that may appear or not without modifying the basic meaning of the pattern, have been indicated by the use of [ ], as included in Table 2.5. Any additional element should in principle be ignored, because it does neither provide any information nor does it affect or modify the semantics of the sentence in NL.

SYMBOLS & ABBREVIATIONS	DESCRIPTION
AP<…>	Adjectival Phrase. It is defined as a phrase whose head is an adjective accompanied optionally by adverbs or other complements as prepositional phrases. AP is followed by the semantic role played by the concept it represents in the conceptual relation (for instance, *property*) in angle brackets.
CATV	Verbs of Classification. Set of verbs of classification plus the preposition that normally follows them. Some of the most representative verbs in this group are: *classify in/into, categorize in/into, subcategorize in/into, group in/into, fall into*.
CD	Cardinal Number.
CN	Class Name. Generic names for semantic roles usually accompanied by preposition. Two main groups have been identified: CN conveying classification (CN-CATV) (*class, group, type, subtype, subclass, category*, *species, family, order*) and CN conveying mereological relations (CN-PART) (*part, set, example, member, constituent, component, element, piece, item, layer*). Other CN can include generic names such as *period* or *phase*.
PART	Verbs of Mereology. Set of verbs conveying the relation existing between a whole and its parts. Some of the most representative ones are: *contain*, *hold*, *form part of, consist of, comprise, be composed of, be made up of, be formed of/by, be part of, be constituted of/by.*
NP<…>	Noun Phrase. It is defined as a phrase whose head is a noun or a pronoun, optionally accompanied by a set of modifiers, and that functions as the subject or object of a verb. NP is followed by the semantic role played by the concept it represents in the conceptual relation in question in <…>, e.g., *class*, *subclass, part.*
PARA	Paralinguistic symbols like *colon*, or more complex structures such as *as follows*, etc., that introduce a list.
PREP	Prepositions
QUAN	Quantifiers such as *all*, *some*, *most*, *many*, *several*, *every*, etc.
REPRO	Relative pronouns such as *that, which, whose*.
( )	Parentheses group two or more elements.
*****	Asterisk indicates repetition.
[ ]	Elements in brackets are meant to be optional, which means that they can be present either at that stage of the sentence or not. By default of appearance, the semantic of the pattern remains unmodified.
¬	Elements preceded by this symbol should not appear in the pattern.

Figure 2.5: Symbols and Abbreviations used in the Formalization of Lexico-syntactic ODPs

2.4 Pattern-Based Ontology Refinement

Richly axiomatized ontologies are essential for powerful, knowledge-intensive applications, since they allow the application of advanced reasoning. As we have already seen in NeOn deliverables D3.8.1[VB08] and D3.8.2[VBBR09], ontology learning from text can exhilarate ontology engineering. However, the possible ontology extension by the means of ontology learning is limited to the information explicitly occurring in text. We consider the ODPs, and in particular Content ODPs, as a potential information source to be exploited for ontology engineering. Pattern-based refinement has already been performed by the OntoCase approach, reported in NeOn deliverables D3.8.1[VB08] and D3.8.2[VBBR09], where Content ODPs have been used to put automatically constructed ontology elements into context by extending the ontology with abstract concepts and properties. Here, we describe an extension of OntoCase which has a different focus, namely the refinement of an ontology's axiomatization by the means of highly axiomatized Content ODPs. This task is related to the XD idea of pattern-based ontology analysis also described previously in this deliverable.

The key idea of the proposed approach is to search for components within an ontology which already partially instantiate a particular Content ODP. In this way, potential missing ontology elements[9] and axioms can be identified and transferred from the underlying Content ODP into the ontology.

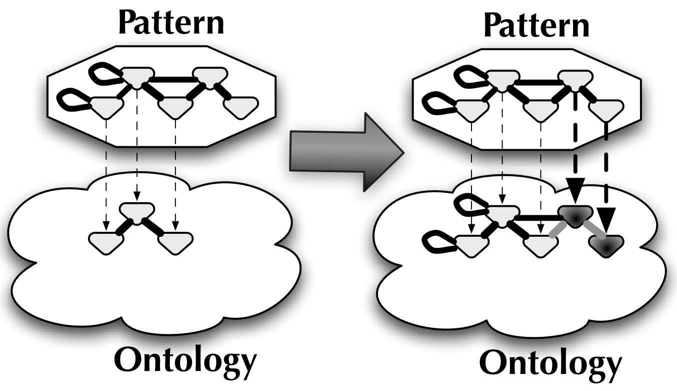


Figure 2.6: Integration of pattern elements into an ontology

If we consider the properties "before" and "after" contained in DBPedia as well as the Content ODP "precedence" introduced in D2.5.1[PGD+08] (shown in Figure 2.7) and if we assume the ontology part including these two properties to be a partial instantiation of the given Content ODP, we see that there are three axioms, that are potentially missing in DBPedia—the axioms expressing the inverseness of the properties "before" and "after" and their transitivity.

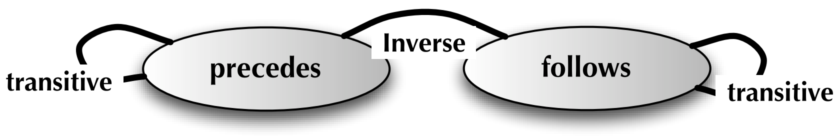


Figure 2.7: The Content ODP "precedence" containing the properties "precedes" and "follows" as well as axioms concerning these properties.

Recognizing partial instantiations of an Content ODP in an ontology would therefore allow checking an ontology for potential missing axioms and based on the output to automatically generate a list of suggestions for the further ontology refinement as demonstrated in Figure 2.6. In this way, frequently occurring and richly axiomatized Content ODPs could help to add some more axioms to a sparsely axiomatized ontology.

[9]In this deliverable section we refer to concepts and properties referenced in an ontology or an ODP as ontology or ODP elements.

In order to automatically recognize a Content ODP by the means of an algorithm, a set of indicative features of this ODP is required. While in OntoCase we basically rely on the lexical features of ontology and ODP elements, in this approach we additionally consider the structure of the ontology or ODP, formed by their axioms, in order to improve the precision of the matching. At the moment this method is not integrated into the XD framework (described previously) but future work includes extending the XD selection and refinement mechanism using this approach. The approach can also be seen as an alternative to the 'fixed rules' for analyzing ontologies as presented in Section 3.1, but has not yet been incorporated into the XD Tools plugin.

In our method, we separate the lexical element matching from the structural matching described briefly in the next subsection.The structural matching and our information integration approach will be the focus of this deliverable section and will be described in detail in subsections 2.4.2, 2.4.3, and 2.4.4.

2.4.1 Lexical element matching

The proposed ontology enrichment method is based on an optimized Ontology Matching method. Matching of two ontologies results in a *mapping* which contains correspondence relationships between the elements of the first ontology and the second one. We use Ontology Matching to obtain the correspondence relationships between the elements of the ODP and ontologies. Thereby, we consider ODPs as small-size ontologies.

General Ontology Matching methods have been widely covered in literature. An overview of the existing approaches can be found in [SE05]. The mapping can contain different types of correspondence relationships as introduced, for instance, by the author in [Euz08]. In our work, we only use the following types of correspondences:

- = (equivalent)
- < (less general)
- > (more general)

The main particularity of our matching method consists in the a priori enrichment of ODPs with external information. We expect that Content ODPs are used repeatedly for the ontology refinement. Therefore, the results of the lexical analysis of Content ODPs can be reused to improve the matching performance. We store the found information as annotations in the ODP. The storage of additional lexical information as annotations also allows a manual specification of lexical information which is the most effective way to encrease the matching recall. Since, as we mentioned before, Content ODPs are used many times for ontology enrichment, additional effort for an a-priori enrichment of ODPs can often be justified.

Another reason for an optimization is the high average level of abstractness characteristic for the concepts of a Content ODP which makes it very difficult to identify correspondences based on a lexicon only. Therefore, we also exploit the information contained in the Watson ontology repository described by the authors in [BGA$^+$08] and also store it as annotations in the ODP.

During the lexical matching of the ontology and the pattern, we rely on a list of annotations containing synonyms and hyponyms for each ODP concept and a list of synonyms for each ODP property. These annotations are matched with local parts of the URIs as well as the labels of ontology elements to obtain the correspondence relationships.

2.4.2 Matching criteria

Before presenting the algorithm, we state the underlying criteria for a high likelihood of pattern instantiation by an ontology part. Thereby, we reduce the problem of identifying partial instantiations of a pattern to the problem of identifying complete pattern instantiations. We rely on the following set of criteria:

Definition 1 *A part O of an ontology is an instantiation of the considered Content ODP P w.r.t. an alignment ω, if its structure can be matched completely with the structure of P in a way that*

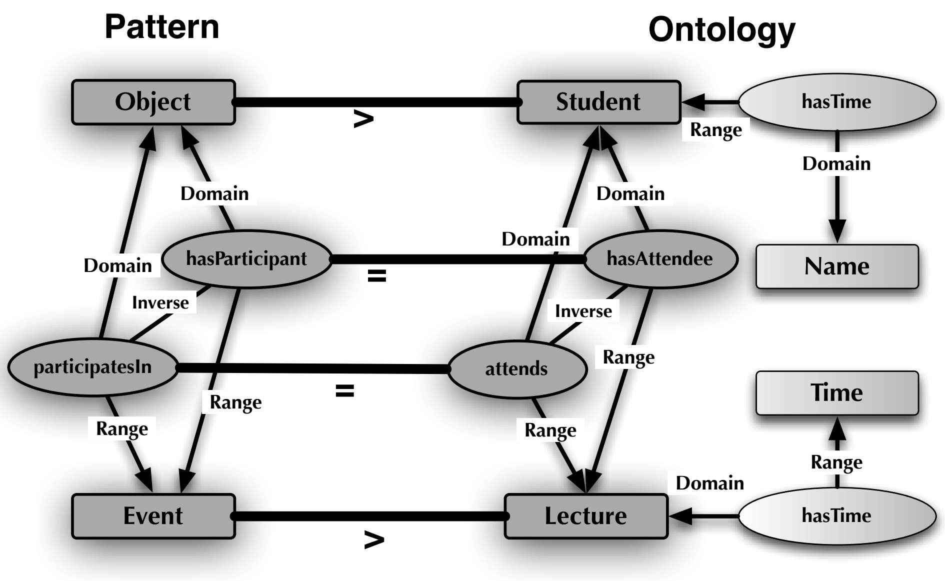


Figure 2.8: An alignment of an ontology part and a Content ODP.

1. *each concept c_P contained in P has exactly one corresponding concept c_O in O, which is equivalent to c_P or a less general according to ω,*
2. *each property r_P contained in P has exactly one corresponding property r_O in O, which is lexically equivalent to it according to ω,*
3. *each axiom a_P of P can be deduced from O when its concepts and properties are replaced by their ω-correspondents described in conditions 1 and 2.*

In our structural matching algorithm, we consider ontologies as well as ODPs as a graph with concepts and properties as nodes and axioms as edges as shown in Figure 2.8. It shows how the Content ODP "Participation" is matched with a part of an example ontology according to the stated criteria. The concept "Student" is a specific "Object", the concept "Lecture" is a specific "Event", and thus they correspond as required by condition 1. Properties "hasParticipant" and "hasAttendee" as well as "participatesIn" and "attends" are expressed by synonymous expressions (condition 2) and their domain and range concepts correspond to each other (condition 3). If the pattern also contains an axiom declaring "participatesIn" to be the inverse property of "hasParticipant", then according to condition 3 it must be possible to deduce it from the set of ontology axioms. In this case, an axiom declaring "attends" to be inverse to "hasAttendee" would suffice.

The criteria stated above provide the basis for our matching algorithm presented in the next subsection.

2.4.3 Matching algorithm

The matching algorithm is stated as shown in Algorithms 1 , 2 and 3.

Algorithm 1 The instantiation algorithm

```
Require: ontology $\mathcal{O}$, ontology pattern $\mathcal{P}$, formulas $\mathcal{B}_{op}$
  result $\leftarrow \emptyset$ {result: set of valid instantiations}
  $P_s \leftarrow$ sortAlongFrequency(elements($\mathcal{P}$), $\mathcal{B}_{op}$)
  $a \leftarrow \emptyset$
  result $\leftarrow$ result $\cup$ trackPaths($\{a\}, P_s, \mathcal{O}, \mathcal{P}, \mathcal{B}_{op}$)
  return result
```

Algorithm 2 trackPaths

```
Require: partial instantiations $\mathcal{I}$, sorted pattern elements $P_s, \mathcal{O}, \mathcal{P}, \mathcal{B}_{op}$
  result $\leftarrow \emptyset$ {result: set of valid instantiations}
  while $P_s$ / {elements($a$)} $\neq \emptyset$ do
    $e_p \leftarrow$ next($P_s$)
    for all $a \in \mathcal{I}$ do
      for all $e_o$ with alignmentAxiom$(e_p, e_o) \in (\mathcal{B}_{op}/a$ do
        $a_n \leftarrow a \cup \{$alignmentAxiom$(e_p, e_o)\}$
        $\mathcal{I} \leftarrow \mathcal{I} \cup$ checkAxioms($a_n$)
      end for
      $\mathcal{I} \leftarrow \mathcal{I}/\{a\}$
    end for
  end while
  return result
```

Algorithm 3 checkAxioms

```
Require: $a$: partial instantiation, $\mathcal{O}$, $\mathcal{P}$, $\mathcal{B}_{op}$
  markInstantiationGreen($a, \mathcal{O}, \mathcal{P}$)
  for all $e_{pg} \in$ greenElements($\mathcal{P}$) do
    for all $a_r \in$ (redAxioms($e_{pg}$) / $\mathcal{B}_{op}$ do
      if entails(axioms($\mathcal{O}$) $\cup\, a, a_r$) then
        $a \leftarrow a \cup$ checkAxioms ($a \cup \{a_r\}$)
      end if
    end for
  end for
  return $a$
```

The algorithm receives an ontology, a Content ODP and a list of mapping axioms as input and generates a list of pattern instantiations as output. To avoid unnecessary computations, it selects one by one pattern elements, which have the fewest lexical matches in the ontology. Since the pattern can only be matched as long as all of its elements have a corresponding element in the ontology, considering only the occurrences of the pattern element with the fewest number of correspondents assures that the least number of ontology parts is analyzed.

Due to possible hyponymy between the elements of the pattern and the ontology, several valid instantiations are possible. For a particular initial partial instantiation, the output can differ depending on the order in which elements are matched. In order to find all valid instantiations, the algorithm tracks all possibilities to construct an instantiations in the recursive procedure *trackPaths* (Algorithm 2).

Each partial instantiation is analyzed using the recursive procedure checkAxioms (Algorithm 3). It marks the already matched pattern and ontology elements as well as already matched pattern axioms green and the remaining elements and pattern axioms red. For each green pattern element e_{pg} it calculates the remaining red axioms which reference only green elements and checks the entailment for each of them. If the entailment was successful for a pattern axiom, it is included into the currently analyzed partial instantiation which forms the input for another run of the described procedure.

2.4.4 Ontology Refinement Based on Partial Pattern Instantiations

The algorithm presented above can be used to find partial Content ODP instantiations by separating pattern elements into obligatory and optional elements and applying the algorithm to each interconnected set of obligatory pattern elements. The default obligatory elements are the concepts and properties of each pattern. Axioms however, are optional. The results are then combined and the list of all valid alignments is the list of all possible combinations of partial alignments. Combined alignments are checked for axiomatic incompatibility with the optional pattern elements in order to avoid refinement suggestions which result in an inconsistent ontology.

Finally, for each pattern, a list of refinement suggestions is generated and presented to the ontology engineer, who can select some suggestions for the integration into the ontology and start the automatic integration process.

Table 2.5: Integration of Content ODP elements into an ontology

Type of unmatched pattern element	Action before inserting the element into the ontology
Concept or Property	Optional renaming by an expert
Axiom	Replacement of all matched concepts and properties by their correspondents

During the integration of axioms, matched elements themselves are not integrated, but are replaced by their lexical ontology correspondents. However, concepts and properties missing in the ontology cannot be replaced, but can be optionally renamed by an expert in order to obtain less general names and, in this way, better suit the level of abstraction present in the ontology. Table 2.5 presents a summary of the integration rules.

2.4.5 Feasibility Study

We conducted an experiment on the ontologies contained in the Watson Ontology repository in order to assess the potential of the proposed method. In the experiment, we used the previously described example consisting of the transitive properties “before” and “after” shown in Figure 2.9 to examine how well the proposed method can perform for axioms involving transitivity and inverseness of properties.

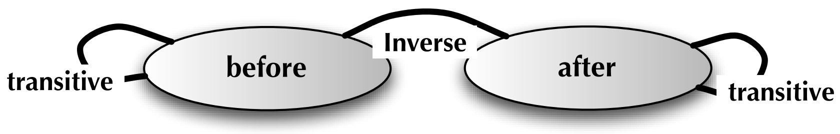


Figure 2.9: ODP “before-after"

In the experiment, we used only the property label itself for the lexical matching. The properties were considered as obligatory pattern elements whereas the axioms about their transitivity and inverseness were considered to be optional.

We used the Watson Search Engine described in [BGA+08] to identify the ontologies containing an Object-Property definition for at least one of the properties. Thereby, 14 documents were identified and matched against the pattern with results as displayed in Table 2.6.

Eight of fourteen documents resulted in a complete match of the pattern including all axioms. Three of the ontologies did not include the inverseness axiom, but the transitivity axioms. Two documents did not contain a definition for the property “after”, but a definition for the property “before” which was defined as transitive. One document did not contain any of the mentioned axioms. We manually examined the refined ontologies and found that the performed completions were semantically justified.

2.4.6 Summary and Outlook

In this section, we presented an algorithm for the identification of Content ODP instantiations in ontologies along with a method to transfer axioms contained in Content ODPs into a target ontology. The results of our

Table 2.6: Experiment results: matching of the after-before-pattern with ontologies indexed by the Watson Ontology Search Engine

Ontology URL	existing elements
morpheus.cs.umbc.edu/aks1/ontosem.owl	Inverseness only
lists.w3.org/Archives/Public/www-rdf-logic/2003Apr/att- 0009/SUMO.daml	All except “After”
secse.atosorigin.es:10000/ontologies/SUMO.owl	All except “After”
daml.umbc.edu/ontologies/cobra/0.3/daml-time	Inverseness only
ai.sri.com/daml/ontologies/time/Time.daml	Inverseness only
cs.umd.edu/ golbeck/daml/slaveOnt.daml	Elements without axioms
cs.vu.nl/ pmika/owl-s/time-entry-fixed.owl	Complete
isi.edu/ pan/damltime/time-entry.owl	Complete
pervasive.semanticweb.org/ont/2004/06/time	Complete
pervasive.semanticweb.org/ont/dev/time	Complete
isi.edu/ pan/damltime/time.owl	Complete
mogatu.umbc.edu/ont/2004/01/Time.owl	Complete
sweet.jpl.nasa.gov/sweet/time.owl	Complete
daml.umbc.edu/ontologies/cobra/0.4/time-basic	Complete

experiment demonstrate the potential of the reuse of formalized knowledge contained in ODPs. However, in order to assess the impact of the method more precisely, we plan a large-scale evaluation involving a large set of Content ODPs with different characteristics, but this evaluation will most likely be conducted after the end of this project.

The availability of appropriate and complete ODPs is essential for the effectiveness of our approach. Hence, we are currently working on semi-automatic methods to acquire useful Content ODPs as well as the necessary lexical information for each pattern. For the former, we are planning to exploit existing ontologies to identify frequently co-occurring characteristics of ontology elements and in this way to identify particularly useful ontology patterns for ontology refinement. For the latter, we expect existing broad-coverage data sets such as WordNet, BillionTriple-Challenge and DBPedia to be valuable resources.

Since the effectiveness of our approach is highly dependent on the quality of the lexical matching, we are currently working on the incorporation of disambiguation techniques in our lexical matching approach.

Chapter 3

Software support

3.1 XD Tools for the NeOn toolkit

The **eXtreme Design tools** for NeOn toolkit (XD Tools) offers the possibility to perform operations on ontology design patterns (ODPs) according to the eXtreme Design (XD) methods, as described previously in Section 2.1. XD has its own conceptual definition in accordance with the C-ODO light ontology (see D2.1.2 [GP09a]), implemented as an OWL file. XD uses C-ODO Light as a reference model for implementing its functionalities. Furthermore, part of its behaviour is determined by C-ODO Light-based ontologies (see Section 3.1.7). Concepts such as Ontology Library, Ontology, Ontology Design Pattern, Content Ontology Design Pattern (Content ODP) are sample items that the XD Tools manage. The concept of networked ontology is native in XD. The main idea is to provide the user with pattern-based operations, such as specialization, composition, instantiation, according to the guidelines defined in D2.5.1 [PGD$^+$08] in the context of the XD methodology (Section 2.1).
The tool provides a perspective - "eXtreme Design" - that groups all the user components: the *ODP Registry browser and ODP Details view*, the *XD Selector*, and the *XD Analyzer*. Other components can be activated in several ways within the interface, i.e. the *XD Wizards*, and the *XD Annotation dialog*. In the following sections we describe all functionalities in detail, giving examples related to the methods described in Section 2.1.4. Then we give an overview of the sofware architecture with particular focus on extendibility and ontology-based components in Section 3.1.7.

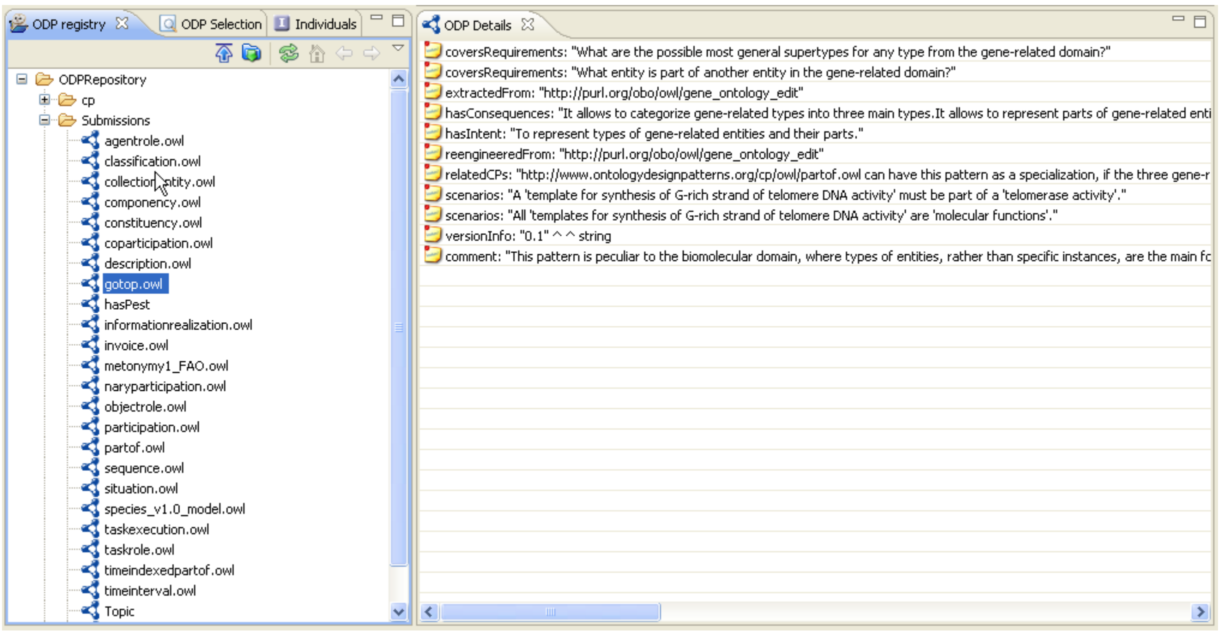
3.1.1 Pattern registry browsing

The **ODP Registry**, see Section 3.1.7 for an overview of the architecture, is an Eclipse view that exposes sets of ODPs to the user (see Figure 3.1). In this way users can have access to a set of reusable OWL patterns that can be directly browsed and exploited in the modelling process, without necessarily having them locally stored. This can be extremely beneficial if the process of ontology modelling is collaborative and based on shared components. ODP registries can be easily refreshed, showing new or updated content, and newly published patterns (for example published in the ODP portal, see Section 3.2) can be immediatly exploited by the user in different ways:

1. *Browsing and detailed visualization*: When a pattern is selected from the registry tree, all OWL annotations are visualized in the ODP Details view;

2. *Import*: The pattern can be directly imported in any working ontology;

3. *Clone*: The pattern can be copied locally into a working project folder of the tool.



ODPRepository
- cp
- Submissions
  - agentrole.owl
  - classification.owl
  - collectionentity.owl
  - componency.owl
  - constituency.owl
  - coparticipation.owl
  - description.owl
  - gotop.owl
  - hasPest
  - informationrealization.owl
  - invoice.owl
  - metonymy1_FAO.owl
  - naryparticipation.owl
  - objectrole.owl
  - participation.owl
  - partof.owl
  - sequence.owl
  - situation.owl
  - species_v1.0_model.owl
  - taskexecution.owl
  - taskrole.owl
  - timeindexedpartof.owl
  - timeinterval.owl
  - Topic

coversRequirements: "What are the possible most general supertypes for any type from the gene-related domain?"
coversRequirements: "What entity is part of another entity in the gene-related domain?"
extractedFrom: "http://purl.org/obo/owl/gene_ontology_edit"
hasConsequences: "It allows to categorize gene-related types into three main types.It allows to represent parts of gene-related enti
hasIntent: "To represent types of gene-related entities and their parts."
reengineeredFrom: "http://purl.org/obo/owl/gene_ontology_edit"
relatedCPs: "http://www.ontologydesignpatterns.org/cp/owl/partof.owl can have this pattern as a specialization, if the three gene-r
scenarios: "A 'template for synthesis of G-rich strand of telomere DNA activity' must be part of a 'telomerase activity'."
scenarios: "All 'templates for synthesis of G-rich strand of telomere DNA activity' are 'molecular functions'."
versionInfo: "0.1" ^ ^ string
comment: "This pattern is peculiar to the biomolecular domain, where types of entities, rather than specific instances, are the main fc

Figure 3.1: ODP Registry browser

The default registry is provided by the ODP portal (see Section 3.2[1]) as an OWL/RDF file[2]. Each registry groups ODP resources (public OWL files) on the basis of some criteria, defined by the registry provider. For example, the registry provided by the ODP web portal groups patterns as follows: *Catalogue*; *Submissions*; *Domains* (see also Section 3.2.5). The user can select a pattern and see all the OWL annotations in a related view (**ODP Details**). Since one of the features of the ODPs is that they must be fully annotated (see D2.5.2 [PGD$^+$08]), the users can base their selection on this information. Patterns can then be imported in the working ontology or cloned locally. In the context of the eXtreme design methodology described in Section 2.1.4 this capability is used within tasks 6 and 7, as described in Example 3.1.

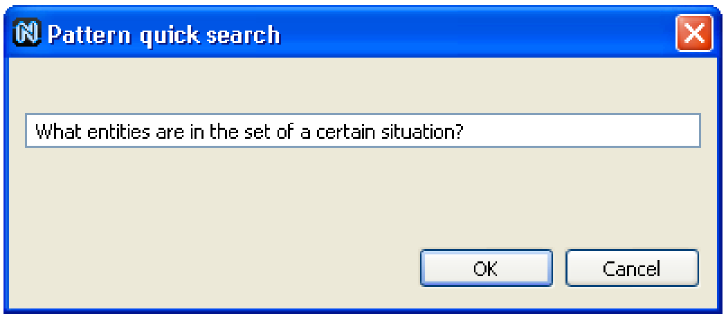
Example 3.1 XD Workflow Task 6-7 (see Section 2.1.4): **Match the CQ to GUCs - Select the Content ODPs to reuse**

CQ_1 : "What are the exploitation state and vertical distance observed in a given climatic zone for a certain resource?"
The user browses the ODP registry within the XD tool, reads the patterns' descriptions in the ODP Details view and eventually finds the following annotation of the 'Observation' Content ODP:
covers requirement: "What objects have been observed? What are the observations of this object? What are the parameters under which this object was observed? What objects where observed under this parameter?"
The user decides to select the 'Observation'-ODP, and imports it into the working ontology.

3.1.2 Pattern selection

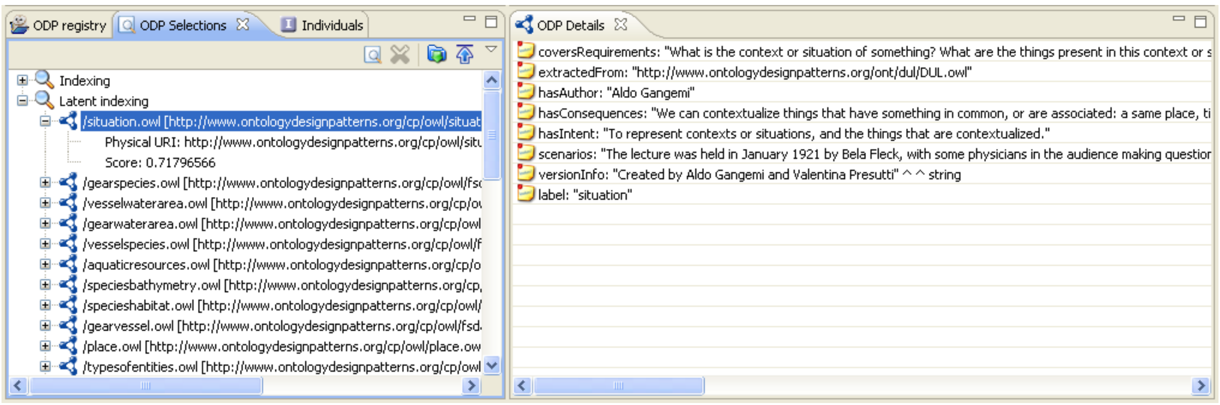
The aim is to give support to the end-user with a component for proposing patterns to be reused to cover requirements expressed as competency questions. In Example 3.1 the task of pattern selection can be challenging. It is necessary to have an overview of the available patterns, and the user needs to already be familiar with them, otherwise the effort of browsing the annotations is most likely too difficult and time consuming. For this reason we have developed another component that intends to provide an easier way of selecting patterns, i.e. the **XD Selector**.
Since pattern selection is one of the most difficult tasks to automate, we have developed the component as

[1]http://ontologydesignpatterns.org (also in Presutti et al. [PDGS08])
[2]http://ontologydesignpatterns.org/schemas/repository.owl. See Section 3.2.5



What entities are in the set of a certain situation?

Figure 3.2: XD Selector: The user can insert CQs into the quick search dialog.



Indexing
Latent indexing
/situation.owl [http://www.ontologydesignpatterns.org/cp/owl/situat
Physical URI: http://www.ontologydesignpatterns.org/cp/owl/situ
Score: 0.71796566
/gearspecies.owl [http://www.ontologydesignpatterns.org/cp/owl/fsc
/vesselwaterarea.owl [http://www.ontologydesignpatterns.org/cp/ow
/gearwaterarea.owl [http://www.ontologydesignpatterns.org/cp/owl
/vesselspecies.owl [http://www.ontologydesignpatterns.org/cp/owl/f
/aquaticresources.owl [http://www.ontologydesignpatterns.org/cp/o
/speciesbathymetry.owl [http://www.ontologydesignpatterns.org/cp,
/specieshabitat.owl [http://www.ontologydesignpatterns.org/cp/owl/
/gearvessel.owl [http://www.ontologydesignpatterns.org/cp/owl/fsd.
/place.owl [http://www.ontologydesignpatterns.org/cp/owl/place.ow
/typesofentities.owl [http://www.ontologydesignpatterns.org/cp/owl

coversRequirements: "What is the context or situation of something? What are the things present in this context or s
extractedFrom: "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl"
hasAuthor: "Aldo Gangemi"
hasConsequences: "We can contextualize things that have something in common, or are associated: a same place, ti
hasIntent: "To represent contexts or situations, and the things that are contextualized."
scenarios: "The lecture was held in January 1921 by Bela Fleck, with some physicians in the audience making questior
versionInfo: "Created by Aldo Gangemi and Valentina Presutti" ^ ^ string
label: "situation"

Figure 3.3: The user has marked the **Situation**-pattern in the result-list.

an extensible system that permits to plug multiple selection services (see Section 3.1.7 for technical details). The current version offers the following selection services, both targeting Content ODPs published in the ODP Portal.

1. **Indexing**: pattern search based on keyword indexing. This service connects to a remote server and queries a keyword based index of ODP. Entity names and all labels and comments have been included in this index.

2. **Latent indexing**: pattern search using latent semantic indexing. This service connects to a remote server and queries indexes built using LSA. The results are also patterns that are indirectly related to the query.

The user can query these basic selection services providing either simple keywords or a complete competency question (CQ). In the following example we assume that the user provides a CQ:

Example 3.2 XD Workflow Task 6 (see Section 2.1.4): **Match the CQ to GUCs**

Assume that "What entities are in the setting of a certain situation?" is the provided CQ.
The user inserts the CQ into the XD Selector's quick search facility (see figure 3.2) and gets the result shown in figure 3.3. The **Latent indexing** selection service displays the pattern **Situation** as its first result. By selecting it, the ODP Details view displays the annotations of the pattern.

The XD Selector view displays as result a list of proposed ODPs to be reused in the current working ontology, with details about the motivation, the synopsis and a set of candidate axioms. By clicking on a result, details (ontology annotations) are shown in the ODP Details view.

3.1.3 Pattern specialization

The eXtreme Design methods identify several ways of reusing ODPs. In particular, regarding Content ODPs, specialization is the primary step for reusing modules (related to task 8, see Section 2.1.4). Content ODPs should be reused by linking pattern entities to locally defined entities with axioms such as subsumption. This task can be challenging for an inexperienced user if it is done at a detailed level, i.e. each element at a time. It is not easy, with current tools, for example, to make a clear distinction between pattern entities that are the most specific with respect to the domain (i.e. bottom-level entities) and the more generic ones. From the user perspective, the specialization operation is a process that follows, at a granular level, the following steps:

1. import the pattern to be specialized into the working ontology;
2. declare subClasses/subProperties for each of the (most specific) pattern entities needed, and define labels and comments for each newly created entity;
3. add domain and range declarations, class restrictions, disjointness axioms and any additional axiom needed.

The **Specialization wizard** is the component provided by XD Tools for guiding the user in the specialization of Content ODPs. The wizard can be accessed either from the Navigation view, i.e. through the contextual menu activated while right-clicking on an ontology file, or from the ODP Registry view (see Section 3.1.1). In both cases the selected file (marked in the graphical user interface) should be the pattern to specialize. The wizard guides the user through the following steps:

Step 1. Define specialization input and result

The user selects the working project and ontology (if needed), and selects one of three modalities:

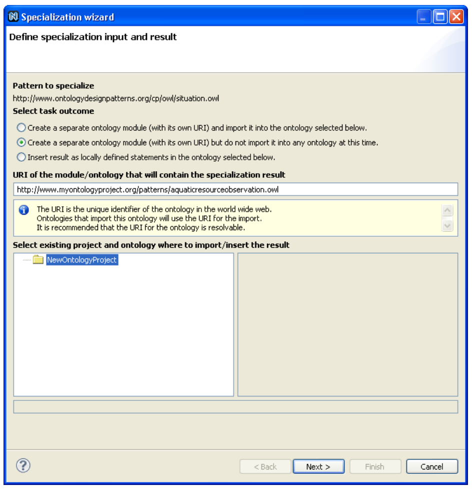
- *Create a separate ontology module (with its own URI) and import it.* The ODP is specialized and the result is a new ontology module, i.e. a new ontology, which is then imported into the working ontology.
- *Create a separate ontology module (with its own URI) but don't import it.* The ODP is specialized and the result is a new ontology module, i.e. a new ontology, but it is not imported into any ontology at this time, the module is simply stored in the working project.
- *Insert the result as locally defined statements.* The ODP is specialized, imported into the working ontology, and the newly created axioms, i.e. the specializations, are inserted locally in the working ontology.

Step 2. Select the entities to specialize

A tree of classes and properties, organized by subsumption, is displayed to the user who can select the entities to be specialized. As an initial solution only bottom-level entities, i.e. the most specific classes and properties, are displayed by default, then the user can expand the trees by selecting the *expand all* option to show all the entities. The intention is to help the user to select the correct entities to specialize without overloading the user with information in the graphical user interface. This wizard page provides support for inverse properties, which must be selected as a couple, i.e. if a property is selected its inverse is automatically selected as well.

Step 3. Create new entities

For each of the selected entities the user must create a new, specialized, one. Three buttons are provided in the graphical user interface: *add*,*edit*, and *remove*. Clicking on the add button opens a new entity dialog



Define specialization input and result

Pattern to specialize
http://www.ontologydesignpatterns.org/cp/owl/situation.owl

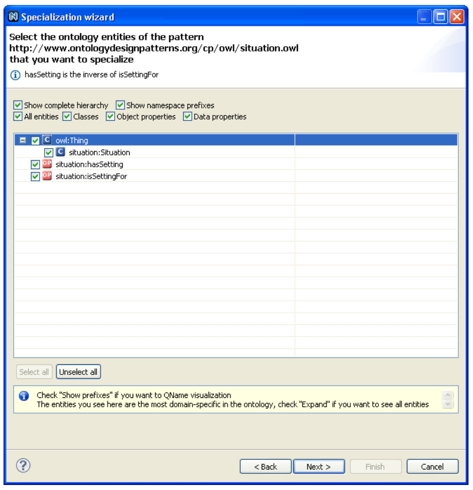
Select task outcome
- ○ Create a separate ontology module (with its own URI) and import it into the ontology selected below.
- [✓ Create a separate ontology module (with its own URI) but do not import it into any ontology at this time.]
- ○ Insert result as locally defined statements in the ontology selected below.

URI of the module/ontology that will contain the specialization result
http://www.myontologyproject.org/patterns/aquaticresourceobservation.owl

The URI is the unique identifier of the ontology in the world wide web.
Ontologies that import this ontology will use the URI for the import.
It is recommended that the URI for the ontology is resolvable.

Select existing project and ontology where to import/insert the result
NewOntologyProject

(a) The user decides to create a new ontology module.



Select the ontology entities of the pattern
http://www.ontologydesignpatterns.org/cp/owl/situation.owl
that you want to specialize

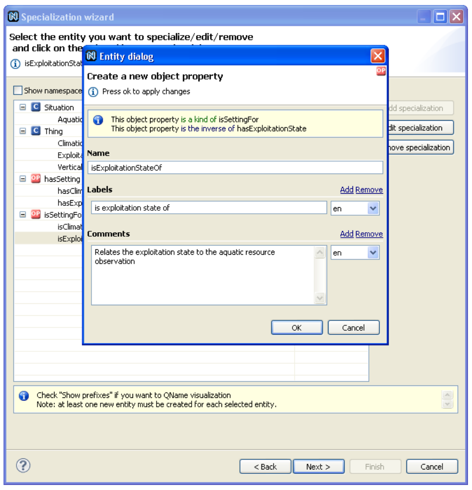
hasSetting is the inverse of isSettingFor

✓ Show complete hierarchy ✓ Show namespace prefixes
✓ All entities ✓ Classes ✓ Object properties ✓ Data properties

- ✓ owl:Thing
  - ✓ situation:Situation
- ✓ situation:hasSetting
- ✓ situation:isSettingFor

Check "Show prefixes" if you want to QName visualization
The entities you see here are the most domain-specific in the ontology, check "Expand" if you want to see all entities

(b) The user selects the entities to specialize.



Select the entity you want to specialize/edit/remove
and click on the

isExploitationStat

☐ Show namespace

- Situation
  - Aquatic
- Thing
  - Climatic
  - Exploita
  - Vertical
- hasSetting
  - hasClim
  - hasExp
- isSettingFo
  - isClimat
  - isExploi

Create a new object property
Press ok to apply changes

This object property is a kind of isSettingFor
This object property is the inverse of hasExploitationState

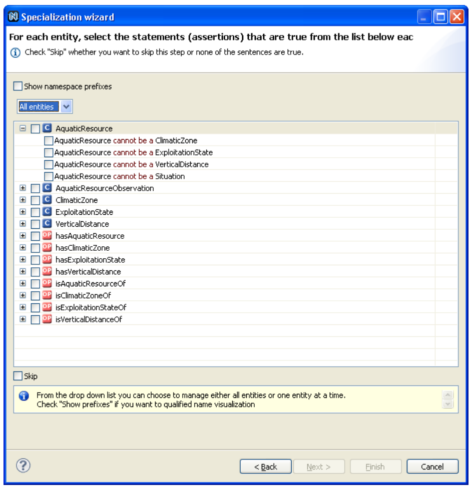
Name
isExploitationStateOf

Labels
is exploitation state of — en

Comments
Relates the exploitation state to the aquatic resource observation — en

Check "Show prefixes" if you want to QName visualization
Note: at least one new entity must be created for each selected entity.

(c) New entities are created. (b)



For each entity, select the statements (assertions) that are true from the list below eac

Check "Skip" whether you want to skip this step or none of the sentences are true.

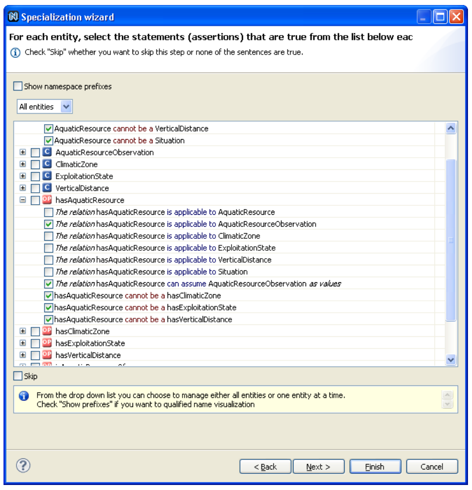
☐ Show namespace prefixes

- ☐ AquaticResource
  - ☐ AquaticResource cannot be a ClimaticZone
  - ☐ AquaticResource cannot be a ExploitationState
  - ☐ AquaticResource cannot be a VerticalDistance
  - ☐ AquaticResource cannot be a Situation
- ☐ AquaticResourceObservation
- ☐ ClimaticZone
- ☐ ExploitationState
- ☐ VerticalDistance
- ☐ hasAquaticResource
- ☐ hasClimaticZone
- ☐ hasExploitationState
- ☐ hasVerticalDistance
- ☐ isAquaticResourceOf
- ☐ isClimaticZoneOf
- ☐ isExploitationStateOf
- ☐ isVerticalDistanceOf

☐ Skip

From the drop down list you can choose to manage either all entities or one entity at a time.
Check "Show prefixes" if you want to qualified name visualization

(d) Additional axioms are proposed: disjointness.



For each entity, select the statements (assertions) that are true from the list below eac

Check "Skip" whether you want to skip this step or none of the sentences are true.

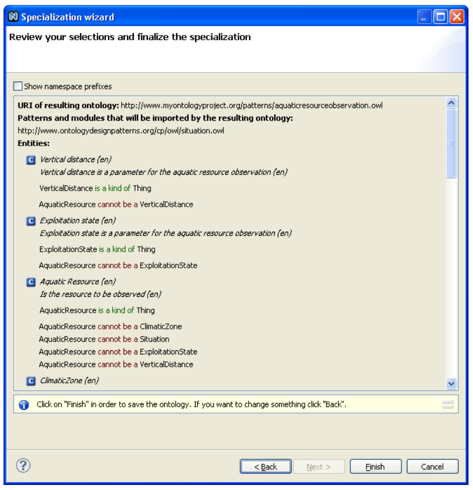
☐ Show namespace prefixes

- ✓ AquaticResource cannot be a VerticalDistance
- ✓ AquaticResource cannot be a Situation
- ☐ AquaticResourceObservation
- ☐ ClimaticZone
- ☐ ExploitationState
- ☐ VerticalDistance
- ☐ hasAquaticResource
  - ☐ *The relation* hasAquaticResource is applicable to AquaticResource
  - ✓ *The relation* hasAquaticResource is applicable to AquaticResourceObservation
  - ☐ *The relation* hasAquaticResource is applicable to ClimaticZone
  - ☐ *The relation* hasAquaticResource is applicable to ExploitationState
  - ☐ *The relation* hasAquaticResource is applicable to VerticalDistance
  - ☐ *The relation* hasAquaticResource is applicable to Situation
  - ✓ *The relation* hasAquaticResource can assume AquaticResourceObservation *as values*
  - ✓ hasAquaticResource cannot be a hasClimaticZone
  - ✓ hasAquaticResource cannot be a hasExploitationState
  - ✓ hasAquaticResource cannot be a hasVerticalDistance
- ☐ hasClimaticZone
- ☐ hasExploitationState
- ☐ hasVerticalDistance

☐ Skip

From the drop down list you can choose to manage either all entities or one entity at a time.
Check "Show prefixes" if you want to qualified name visualization

(e) Additional axioms are proposed: domain/range.



Review your selections and finalize the specialization

☐ Show namespace prefixes

URI of resulting ontology: http://www.myontologyproject.org/patterns/aquaticresourceobservation.owl
Patterns and modules that will be imported by the resulting ontology:
http://www.ontologydesignpatterns.org/cp/owl/situation.owl
Entities:

Vertical distance (en)
Vertical distance is a parameter for the aquatic resource observation (en)
VerticalDistance is a kind of Thing
AquaticResource cannot be a VerticalDistance

Exploitation state (en)
Exploitation state is a parameter for the aquatic resource observation (en)
ExploitationState is a kind of Thing
AquaticResource cannot be a ExploitationState

Aquatic Resource (en)
Is the resource to be observed (en)
AquaticResource is a kind of Thing
AquaticResource cannot be a ClimaticZone
AquaticResource cannot be a Situation
AquaticResource cannot be a ExploitationState
AquaticResource cannot be a VerticalDistance

ClimaticZone (en)

Click on "Finish" in order to save the ontology. If you want to change something click "Back".

(f) Overview.

Figure 3.4: The specialization wizard.

Example 3.3 XD Workflow Task 8 (see Section 2.1.4): **Reuse and integrate selected Content ODPs**

In order to address CQ_1 in the example ("What are the exploitation state and vertical distance observed in a given climatic zone for a certain resource?"), the user selects the pattern **Situation** to reuse from the XD Selector result view. Then the user starts the specialization process by right clicking the pattern, to access the contextual menu, and subsequently clicking on the specialization icon in the menu.

1. **Step 1 (Figure 3.4(a))**. The user decides to create a new ontology module for the **Aquatic Resource Observation** based on the **Situation** pattern, and insert the related URI, and import it into the working ontology.

2. **Step 2 (Figure 3.4(b))**. The user selects the following entities for specialization: *situation:Situation, situation:isSettingFor, situation:hasSetting, owl:Thing*.

3. **Step 3 (Figure 3.4(c))**. The following new classes are created (with labels and comments):

   - AquaticResourceObservation *specialization of situation:Situation*;
   - ExploitationState *specialization of owl:Thing*;
   - ClimaticZone *specialization of owl:Thing*;
   - VerticalDistance *specialization of owl:Thing*.

   Then, for each of the classes, related properties are created, all specializations of the couple *situation:hasSetting/situation:isSettingFor*. For example, `hasClimaticZone/isClimaticZoneOf` to relate an `AquaticResourceObservation` (a kind of `situation:Situation`) with its `ClimaticZone`.

4. **Step 4 (Figure 3.4(d) and Figure 3.4(e))**. Axioms are presented to the user, grouped by entity. For example, the user can state disjointness between the newly created entities, and domain/range relations for properties.

5. **Step 5 (Figure 3.4(f))**. The user browses an overview of the resulting axioms of the new ontology module.

asking for the name, label and comment of the new entity (this information is mandatory, according to the XD principles, since it is known as a good practice to add labels and comments for all entities). Inverse properties must be specialized as a couple, hence a new dialogue is automatically opened for specializing the inverse property when the first one has been specialized.

Step 4. Choose additional axioms

The fourth step gives the possibility to add additional axioms, by selecting them from a list of possible candidates generated by the tool. These axioms are proposed based on the definitions of the original ODP entities. Currently supported axiom types are: domain and range of properties, 'all values from' and 'some values from'-restrictions for classes and disjointness of classes. The proposals derived by the tool are listed in the graphical user interface and can then be considered and selected by the user. The axioms that a user wants to include in the result are marked using checkboxes.

Step 5. Overview

In the final wizard page the user can review all the resulting axioms, and either finalize the process or return to the previous steps to make changes. When the process is finalized, a message is displayed with the option to open the dialog for ontology annotation, i.e. to annotate the new ontology module that has been created. Example 3.3 shows how this component can be used within the XD Workflow.

3.1.4 Pattern annotation

Ontology annotation is a very important activity in the context of XD. The **XD Annotation** dialog (Figure 3.5) is the component supporting annotation of ontology modules, as well as new ontology patterns (in particular, but it can be applied to any ontology). It provides the facility to annotate the ontology using other vocabularies in addition to the annotation properties already provided by OWL/RDF. For the Content ODPs the CPAnnotationSchema[3] is supported[4].

Task 10 (see Section 2.1.4) of the XD Content ODP methodology is dedicated to the release of the new ontology module (which could also be a new Content ODP). A new module has been implemented and in this step it is going to be shared with other developers within the ontology project, and possibly even external users (e.g. users of ontologydesignpatterns.org). Example 3.4 shows how this task can be addressed within XD Tools.

Example 3.4 XD Workflow Task 10: **Release module**

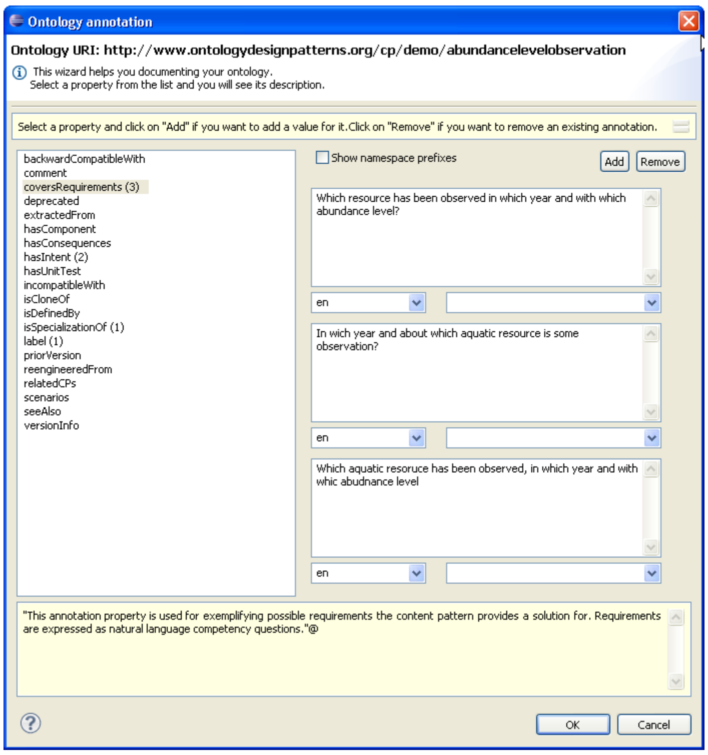
The new ontology module has been developed by reusing Content ODPs. The design pair is now going to share their module with other design pairs within the project, in particular the integration pair(s). For that reason the pair annotates the module using the XD annotation dialog, which provides a built-in set of annotations regarding module (or pattern) intent, requirement coverage, unit tests (figure 3.5). The design pair for example adds the CQs their module covers as values of the 'coversRequirements' annotation property from the CPAnnotationSchema.

3.1.5 Best practices analysis

Best practices are supported through the **XD Analyzer**. The aim of this tool is to provide suggestions and feedback to the user with respect to how good practices in ontology design have been followed, according to the XD methodology. The XD Analyzer has been designed as a pluggable architecture that gives the

[3]Published at http://ontologydesignpatterns.org/schemas/cpannotationschema.owl

[4]Ongoing work includes to support the possibility to add additional, user-defined, vocabularies through a preference page.



Ontology URI: http://www.ontologydesignpatterns.org/cp/demo/abundancelevelobservation

This wizard helps you documenting your ontology.
Select a property from the list and you will see its description.

Select a property and click on "Add" if you want to add a value for it.Click on "Remove" if you want to remove an existing annotation.

backwardCompatibleWith
comment
coversRequirements (3)
deprecated
extractedFrom
hasComponent
hasConsequences
hasIntent (2)
hasUnitTest
incompatibleWith
isCloneOf
isDefinedBy
isSpecializationOf (1)
label (1)
priorVersion
reengineeredFrom
relatedCPs
scenarios
seeAlso
versionInfo

Which resource has been observed in which year and with which abundance level?

en

In wich year and about which aquatic resource is some observation?

en

Which aquatic resource has been observed, in which year and with whic abudnance level

en

"This annotation property is used for exemplifying possible requirements the content pattern provides a solution for. Requirements are expressed as natural language competency questions."@

Figure 3.5: The new ontology module is annotated through the XD Annotation dialog

possibility to easily extend the set of rules according to other/additional rationales. A first set of rules has been setup based on the current best practice principles within XD. Ongoing work includes extendibility-support for plugging in third-party components.

The XD Analyzer displays two levels of information to the user:

- **Warnings**. Warnings highlight situations that can be considered bad practices in ontology development. For instance they are shown when there are missing labels, missing type definitions or imported ontologies are not used.
- **Suggestions**. Suggestions are displayed for additional axioms that could improve the ontology. For instance adding domain and range restrictions is considered to be a good practice in XD.

Table 3.1 shows some examples of the kinds of analyses that can be applied by the component.

Example 3.5 refers to Task 10 (Section 2.1.4) of the XD workflow. In this step the new ontology module should be released. Is it fully annotated? Are there missing labels or comments? Are there entities without any relations? These are the kinds of questions that can be easily answered using the XD Analyzer.

Example 3.5 XD Workflow Task 10: **Release module**

The new ontology module has been developed. Its correctness according to the related CQs has been tested using unit tests, i.e. SPARQL queries. The developer pair then analyzes the pattern using the XD Analyzer and discovers several missing annotations for entities, i.e. labels and comments that should be added to make the pattern more understandable to other developers.

3.1.6 Help and support

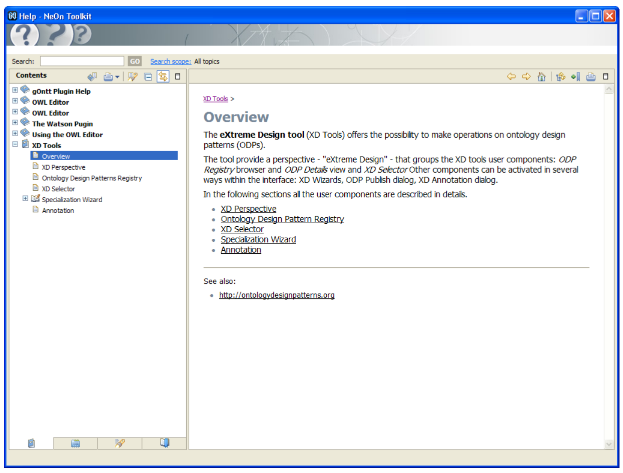
The tool implements several methods for user assistance:

Rule name	**Severity**	**Applies to**	**Rationale / *Message***
Missing label	Warning	All locally defined entities	All entities must have at least one `rdfs:label`. *The entity needs at least one label.*
Missing comment	Warning	All locally defined entities	All entities must have at least one `rdfs:comment`. *The entity needs at least one comment to describe its meaning/nature.*
Isolated entity	Warning	All locally defined entities	Each entity must be related at least to another one through some ontology axiom. *Isolated entity. Relate this entity to another one in the ontology.*
Missing type	Warning	All locally defined entities	Each entity must be the instance of something. This is valid for entities of the T-Box (e.g. a class should be an instance of `owl:Class`) as well as entities in the A-Box. *The entity type is undefined.*
Undefined property extension	Suggestion	All locally defined object properties	Each property should have its domain and range properly defined. *The property domain and range are not defined. To leave a property domain and range undefined or defined as `owl:Thing` is discouraged.*
Missing inverse property	Suggestion	All locally defined object properties	Each object property should have an inverse (except symmetric properties). *The ontology does not declare an inverse for the property and the property is not declared to be symmetric.*
Ontology dimension	Warning	The ontology	The number of locally defined entities is higher than the imported entities (i.e. entities from imported ontologies). *The ontology is too big! It contains declarations of too many entities. It is advisable to split it into two or more ontology modules (or patterns) and then compose them into a new ontology with fewer locally defined entities.*
Unused pattern	Warning	The ontology	All imported patterns should have at least one entity referenced locally. *The ontology imports the pattern but does not use any of the pattern entities.*

Table 3.1: XD Analyzer rule examples.

- **Inline info boxes**. Within the XD graphical user interface components, e.g. dialogues and windows, are yellow boxes with contextual information and hints (for example, in Figure 3.5 annotation properties' details are provided at the bottom).
- **Contribution to the Eclipse Help center**. The XD Tools has its section in the main platform's help center (Figure 3.6(a)).
- **Cheat sheets**. Cheat sheets are short tutorials that guide the users with respect in exploiting of specific tool functionalities. XD provides the following cheat sheets:
  - Pattern selection with the XD Selector (Figure 3.7(a))
  - Pattern specialization with the XD specialization wizard (Figure 3.7(b))
  - Annotating your Content ODP (Figure 3.7(c))

All help content can be found using the search feature of the help functionality (Figure 3.6(b)).



XD Tools >

Overview

The **eXtreme Design tool** (XD Tools) offers the possibility to make operations on ontology design patterns (ODPs).

The tool provide a perspective - "eXtreme Design" - that groups the XD tools user components: *ODP Registry* browser and *ODP Details* view and *XD Selector* Other components can be activated in several ways within the interface: XD Wizards, ODP Publish dialog, XD Annotation dialog.

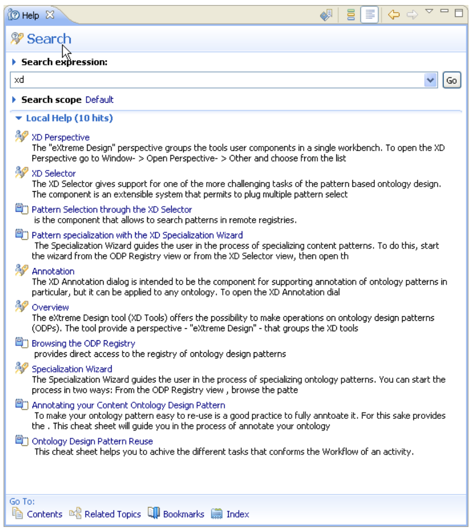
In the following sections all the user components are described in details.

- XD Perspective
- Ontology Design Pattern Registry
- XD Selector
- Specialization Wizard
- Annotation

See also:

- http://ontologydesignpatterns.org

(a) The XD section in the help center



XD Perspective
The "eXtreme Design" perspective groups the tools user components in a single workbench. To open the XD Perspective go to Window- > Open Perspective- > Other and choose from the list

XD Selector
The XD Selector gives support for one of the more challenging tasks of the pattern based ontology design. The component is an extensible system that permits to plug multiple pattern select

Pattern Selection through the XD Selector
is the component that allows to search patterns in remote registries.

Pattern specialization with the XD Specialization Wizard
The Specialization Wizard guides the user in the process of specializing content patterns. To do this, start the wizard from the ODP Registry view or from the XD Selector view, then open th

Annotation
The XD Annotation dialog is intended to be the component for supporting annotation of ontology patterns in particular, but it can be applied to any ontology. To open the XD Annotation dial

Overview
The eXtreme Design tool (XD Tools) offers the possibility to make operations on ontology design patterns (ODPs). The tool provide a perspective - "eXtreme Design" - that groups the XD tools

Browsing the ODP Registry
provides direct access to the registry of ontology design patterns

Specialization Wizard
The Specialization Wizard guides the user in the process of specializing ontology patterns. You can start the process in two ways: From the ODP Registry view , browse the patte

Annotating your Content Ontology Design Pattern
To make your ontology pattern easy to re-use is a good practice to fully anntoate it. For this sake provides the . This cheat sheet will guide you in the process of annotate your ontology

Ontology Design Pattern Reuse
This cheat sheet helps you to achive the different tasks that conforms the Workflow of an activity.

(b) XD contents from the help search feature

Figure 3.6: XD Help

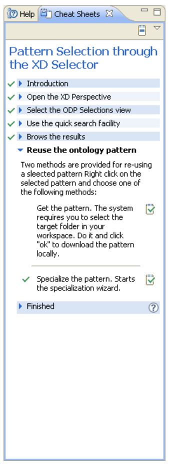
3.1.7 Software architecture (v1.0) and extendability

The XD Tools is based on the Eclipse infrastructure (version 3.5) and is distributed as an eclipse feature. In the context of the NeOn Toolkit it is compatible with version 2.3 of the toolkit and uses the OWLApi v.3.

Component overview

The XD Tools (v1.0) is composed of the following Eclipse plugin-bundles, all grouped in the **eXtreme Design** Feature:

- **it.cnr.stlab.xd**: includes core functionalities (the eXtreme Design perspective).
- **it.cnr.stlab.xd.registry**: the Registry browsing and details component.
- **it.cnr.stlab.xd.selection**: the Selector component.
- **it.cnr.stlab.xd.selection.base**: the built-in selection services (Indexing and Latent indexing) component.



Help | Cheat Sheets

Pattern Selection through the XD Selector

- ✓ Introduction
- ✓ Open the XD Perspective
- ✓ Select the ODP Selections view
- ✓ Use the quick search facility
- ✓ Brows the results
- **Reuse the ontology pattern**

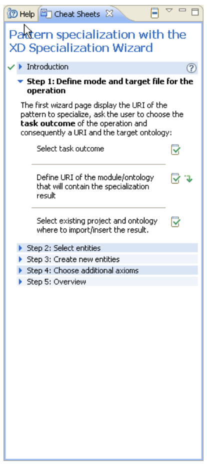
Two methods are provided for re-using a sleected pattern Right click on the selected pattern and choose one of the following methods:

Get the pattern. The system requires you to select the target folder in your workspace. Do it and click "ok" to download the pattern locally.

✓ Specialize the pattern. Starts the specialization wizard.

- Finished

(a) Pattern selection with the XD Selector



Help | Cheat Sheets

Pattern specialization with the XD Specialization Wizard

- ✓ Introduction
- **Step 1: Define mode and target file for the operation**

The first wizard page display the URI of the pattern to specialize, ask the user to choose the **task outcome** of the operation and consequently a URI and the target ontology:

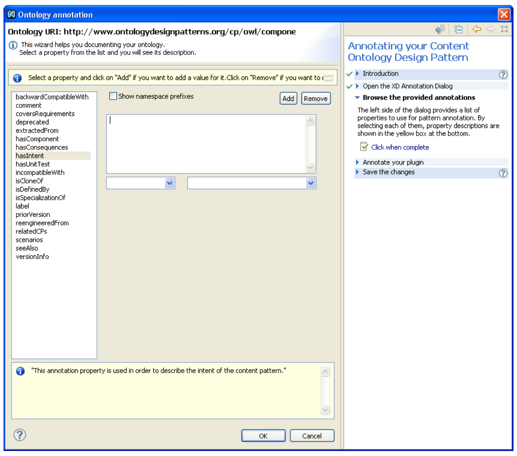
Select task outcome

Define URI of the module/ontology that will contain the specialization result

Select existing project and ontology where to import/insert the result.

- Step 2: Select entities
- Step 3: Create new entities
- Step 4: Choose additional axioms
- Step 5: Overview

(b) Pattern specialization with the XD specialization wizard



Ontology annotation

Ontology URI: http://www.ontologydesignpatterns.org/cp/owl/compone

This wizard helps you documenting your ontology.
Select a property from the list and you will see its description.

Select a property and click on "Add" if you want to add a value for it.Click on "Remove" if you want to

backwardCompatibleWith
comment
coversRequirements
deprecated
extractedFrom
hasComponent
hasConsequences
hasIntent
hasUnitTest
incompatibleWith
isCloneOf
isDefinedBy
isSpecializationOf
label
priorVersion
reengineeredFrom
relatedCPs
scenarios
seeAlso
versionInfo

Show namespace prefixes | Add | Remove

"This annotation property is used in order to describe the intent of the content pattern."

OK | Cancel

Annotating your Content Ontology Design Pattern

- ✓ Introduction
- ✓ Open the XD Annotation Dialog
- **Browse the provided annotations**

The left side of the dialog provides a list of properties to use for pattern annotation. By selecting each of them, property descriptions are shown in the yellow box at the bottom.

Click when complete

- Annotate your plugin
- Save the changes

(c) Annotating your Content Ontology Design Pattern

Figure 3.7: XD Cheat Sheets

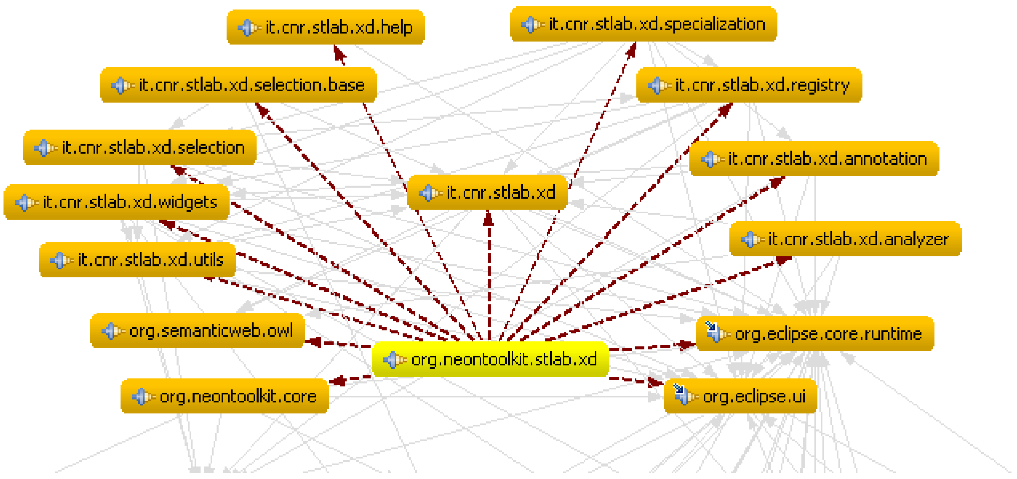


Figure 3.8: XD componenets within the NeOn Toolkit

- **it.cnr.stlab.xd.specialization**: the Specialization component.
- **it.cnr.stlab.xd.annotation**: the Annotation component.
- **it.cnr.stlab.xd.analyzer**: the Analyzer component.
- **it.cnr.stlab.xd.help**: inline help, contextual help and cheat sheets.
- **it.cnr.stlab.xd.widgets**: shared user interface components.
- **it.cnr.stlab.xd.utils**: shared packages for common functions.
- **org.neontoolkit.stlab.xd**: provides integration with the NeOn toolkit UI components.

Figure 3.8 depicts the plugins and their relations within the NeOn Toolkit.

Extending the XD Selector

The XD Selector has been developed with the aim of supporting multiple selection mechanisms. The reasons for providing extensibility features are twofold: matching requirements to ontologies is not a trivial task, and in particular in the case of ontology patterns, so different matching systems could use different algorithms and use this component as a container for test and benchmarking; additionally the other hand pattern retrieval can be seen from different points of view and different matching systems can have different approaches. The pattern selection service must be implemented as an Eclipse plugin that extends the extension-point `it.cnr.stlab.xd.selection.services`, provided by the XD Tools plugin. In the selection service plugin, developers should provide a class that extends the `it.cnr.stlab.xd.selection.AbstractpatternSelectionService`.
This is in an example of the plugin.xml file:

```
<pre>
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         point="it.cnr.stlab.xd.selection.services">
      <selectionService
            class="xdselectionserviceexample.PatternSelectionServiceExample">
      </selectionService>
   </extension>
</plugin>
</pre>
```

Further technical details can be found at http://stlab.istc.cnr.it/stlab/XDTools.

Thirdy-party and user-defined registries

The ODP Registry component of XD is based on an ontological description extending the C-ODO Light ontology [PGP+09]. Using OWL modelling and the C-ODO ontology to represent the registry brings various benefits. First of all, it provides flexibility, since any repository can be described through a C-ODO Light-based ontology. On the other hand other kinds of representations of the registry can be exploited in the future by means of the C-ODO description. The reference ontology to be used to provide a custom registry to the tool is http://ontologydesignpatterns.org/schemas/meta.owl, which reuses some basic entities and relations from the C-ODO ontology (the modules C-ODO/Data and C-ODO/Kernel - see [GLP+05]). The ontological representation must use at least the following entities:

- `coddata:OntologyLibrary`
- `codkernel:Ontology`
- `meta:hasOntology` and `meta:isOntologyOf` (these properties specialize the couple `partof:hasPart` and `partof:isPartOf` in the Part Of module within C-ODO)

XD uses this representation to build the navigation tree of its ODP Registry browser. The ODP library browser shows the folder-based structure of the registry, but at the core level it is reading the OWL representation. In this way thirdy-party ontology modules can be easily added to the tool through the C-ODO Light support, allowing the reuse of ontology libraries from external providers. Note that the provider of the registry must not necessarly be the provider of the patterns, since it is the subject that organizes the public knowledge based according to some criteria and exposes this reusable knowledge (i.e. patterns) to the community. The user can then easily add additional registries through the XD preference section.

3.1.8 Ongoing work

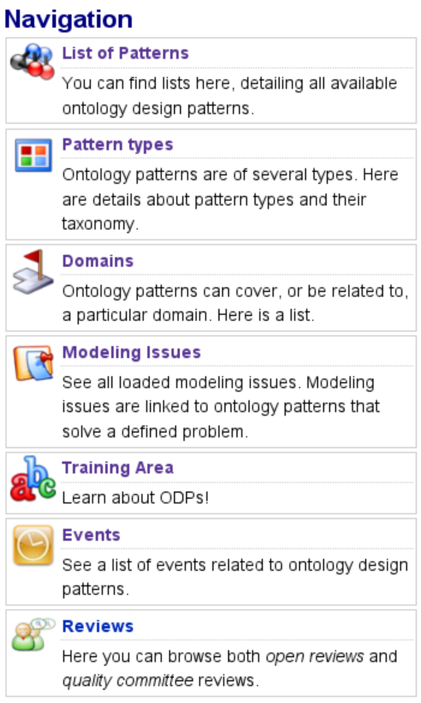
XD Tools is targeted to support operations with ontology design patterns, in particular Content ODPs. Some operations are not yet covered by the tool (instantiation and composition, for example), for this task we are studying how to provide simple and intuitive interfaces for that. Ongoing work includes support for the reuse of other OP types. Logical ODP, for instance, can be used for the definition of macros that generates knowledge patterns based on the logical structure (for example, n-ary relations). Refactoring operations can also be implemented by the mean of Logical ODPs, for example an allValuesFrom and someValuesFrom restriction couple can be refactored as domain and range definition for a property and viceversa. Future work includes support for Re-engineering ODPs, Lexyco-syntactic ODPs and Correspondence ODPs (more details at http://ontologydesignpatterns.org). Current research is also focused on exploring new ways for requirement expression and specification through a diagram-based interface . Finally, evaluation of the tool has been conducted during PhD courses but more feedback is needed before reporting it. Here we can mention that pattern based ontology design is an approach that can increase the quality of the ontology but brings the threat of an overload of work without an intuitive software tool to support it. We want to explore this further in the near future.

3.2 ontologydesignpatterns.org - The ODP Portal

Ontologydesignpatterns.org (hereafter ODP Portal) is a semantic web portal dedicated to ontology design best practices for the semantic web, with particular focus on ontology design patterns (ODPs). The ODP Portal is targeted at users who are interested in best practices for ontology design and ontology engineering. ODPs encode ontology engineering best practices, as such they are an aid to design high-quality ontologies. As described by Presutti et al. in [PG08] ODPs are classified into different types (see also D5.1.1 [SFBG$^+$07] and D2.5.1 [PGD$^+$08]), e.g. Logical ODPs, Content ODPs, Lexico-syntactic ODPs, Re-engineering ODPs, etc. The ODP Portal was originally supporting the lifecycle of *Content* ontology design patterns (Content ODPs), i.e. solutions to content modelling problems, e.g for time, space, biological sequencing, geographic areas, invoicing, etc. However, as will be explained later in this section, the ODP Portal now supports several ODP types.

The ODP Portal is a *semantic* web portal supporting the life cycle of ODPs, from proposals, to evaluation and certification, and that makes them available for download. For the sake of evaluation the portal exploits a (Semantic) Media Wiki extension, named *Evaluation WikiFlow*, that was developed specifically for this purpose. Evaluation WikiFlow supports the workflow for wiki article evaluation by storing a semantic report of the evaluation history, including the rationales expressed in the reviews.

The ODP Portal web portal was launched to the public in February 2009. In six months the web portal has been extended and improved in several respects, both regarding technical stability and usability (see Figure


Navigation

List of Patterns
You can find lists here, detailing all available ontology design patterns.

Pattern types
Ontology patterns are of several types. Here are details about pattern types and their taxonomy.

Domains
Ontology patterns can cover, or be related to, a particular domain. Here is a list.

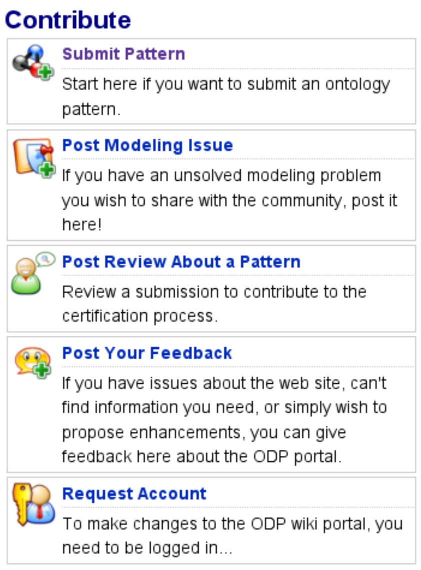
Modeling Issues
See all loaded modeling issues. Modeling issues are linked to ontology patterns that solve a defined problem.

Training Area
Learn about ODPs!

Events
See a list of events related to ontology design patterns.

Reviews
Here you can browse both *open reviews* and *quality committee* reviews.

(a) Navigation


Contribute

Submit Pattern
Start here if you want to submit an ontology pattern.

Post Modeling Issue
If you have an unsolved modeling problem you wish to share with the community, post it here!

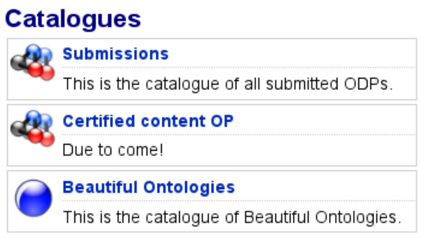
Post Review About a Pattern
Review a submission to contribute to the certification process.

Post Your Feedback
If you have issues about the web site, can't find information you need, or simply wish to propose enhancements, you can give feedback here about the ODP portal.

Request Account
To make changes to the ODP wiki portal, you need to be logged in...

(b) Contribute

Figure 3.9: ODP Portal organization - the navigation menu on the front page showing the different areas of the portal for contributing and navigating through the portal.

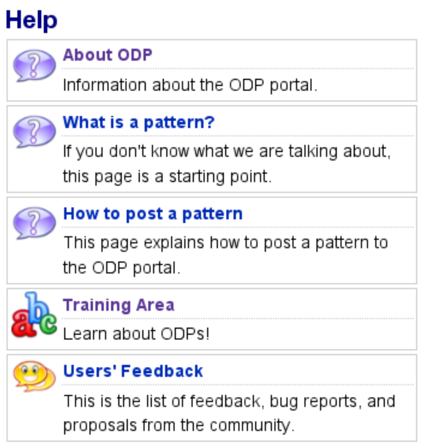

Catalogues

Submissions
This is the catalogue of all submitted ODPs.

Certified content OP
Due to come!

Beautiful Ontologies
This is the catalogue of Beautiful Ontologies.

(a) Catalogues


Help

About ODP
Information about the ODP portal.

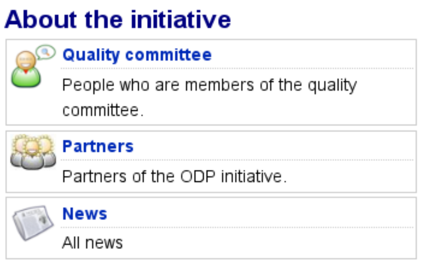
What is a pattern?
If you don't know what we are talking about, this page is a starting point.

How to post a pattern
This page explains how to post a pattern to the ODP portal.

Training Area
Learn about ODPs!

Users' Feedback
This is the list of feedback, bug reports, and proposals from the community.

(b) Help


About the initiative

Quality committee
People who are members of the quality committee.

Partners
Partners of the ODP initiative.

News
All news

(c) About the initiative

Figure 3.10: ODP Portal organization - the second part of the navigation menu on the front page showing the catalogues, and additional user support functionalities.

3.9 and 3.10 for an overview of ODP Portal areas and functionalities). In its first version the portal supported only the lifecycle of Content ODPs. Currently, the ODP Portal has been extended to support the lifecycle of the following ODP types:

- Content ODPs
- Re-engineering ODPs
- Alignment ODPs
- Logical ODPs
- Architectural ODPs
- Lexico-syntactic ODPs

The ODP Portal is also used as a resource for enhancing collaborative ontology design with the NeOn Toolkit through the XD Tools (see Section 3.1), e.g. by retrieving ODPs and their related information from the portal to support matching, specializing, and composing Content ODPs in specific ontology projects according to the guidelines defined in D2.5.1[PGD+08].

The following sections are organized as follows: Section 3.2.1 describes ODP Portal organization and main functionalities, Section 3.2.2 focuses on the user types within the portal, Section 3.2.3 explains and shows how ODPs are presented within the ODP Portal, Section 3.2.4 depicts the evaluation process exploiting Evaluation WikiFlow, and, finally, Section 3.2.5 discusses the pattern registry.

3.2.1 ODP Portal organization

In the scope of the functionality that we have implemented so far, depicted in Figure 3.11 through a use-case diagram, the ODP Portal contains different areas and several different types of users. Currently, the web portal is organized into the following areas (see also Figure 3.9 and 3.10 for an overview of the areas from a user perspective, i.e. the navigation facilities on the ODP Portal main page):

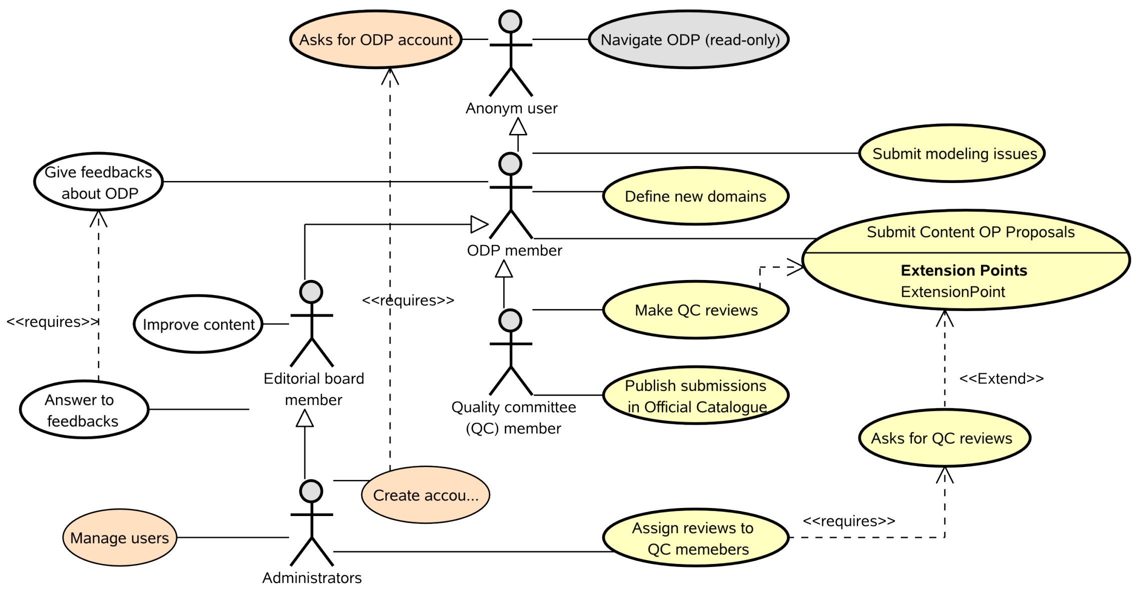


Figure 3.11: ODP Portal Use Case Diagram: main functionalities and user roles.

Navigation This area summarizes the main 'areas' in which any user can find information and content concerning ODPs, other resources, information about pattern types, domains covered by the portal, modelling issues, training, events and reviews (see Figure 3.9(a)). The domain page lists all domains that have been defined by ODP Portal users, and allows them to create new ones. Each Content ODP or Modeling Issue is associated to one or more domains e.g. organization, business, health, music, etc., through the semantic property `domain`. This semantic property is extremely useful for users who are looking for Content ODPs for a certain domain of knowledge. In fact, they can query the ODP Portal for all Content ODPs addressing a certain domain. The Training area collects tutorials, publications, and sample modelling exercises. The review area collects and publishes the reviews of proposed patterns.

Contribute This area is meant to receive ODP Portal users' contributions and handle discussions around contributions (see Figure 3.9(b)). In this area users find information about how to submit a pattern. There are two alternative procedures for proposing a pattern, both are accessible through the *Submit a Pattern* page:

- The user chooses a name for the pattern and is guided by a simple form that, once filled out and saved, is posted as a new pattern proposal.
- The user uploads the OWL file of a Content ODP and the system automatically fills the Content ODP form, which (once checked by the user and accepted) is posted as a Content ODP proposal. The form underlying the Content ODP page can be further edited by the user by the *edit with form* functionality. In order to exploit this facility at its best, the OWL file must be annotated in terms of `rdfs:label`, `rdfs:comment`, and annotation properties defined by an annotation schema, in this case the *CPAnnotationSchema*[5]. All these annotation properties are mapped to semantic properties in the ODP Portal[6].

In the *feedback area* ODP Portal users can post feedback entries. Typically, feedback entries identify issues to be addressed in order to improve the ODP Portal. The editorial board and the ODP Portal core developers treat such feedback as suggestions for starting new development tasks and/or as maintenance requirements. This area also includes facilities for posting modelling issues, with the aim of sharing problems, experiences, and adopted solutions with the community. Finally, users can post reviews about patterns. Proposed ODPs can be reviewed by all registered users, i.e. by posting *open reviews*. However, they are eventually also reviewed by at least two members of the ODP Portal Quality Committee, formed by ontology experts, these are denoted *QC reviews*. The aim of reviews is twofold: on one hand, they provide ODP Portal users with explicit rationales behind the evaluation of specific domain solutions, on the other hand reviews provide the author of a certain ODP with guidance for addressing possible problems in order to get the ODP certified and accepted by the community. Open reviews are extremely useful to Quality Committee members when formulating their review, and also for identifying new Quality Committee members from the community.

Catalogues This is the area where ODP Portal users find the ODPs and ontology resources (see Figure 3.10(a)):

- **Submissions**. The proposed ODPs are expected to come from practical and successful experiences of ontology development, and are submitted by registered users. All proposed patterns belong to the ODP Portal namespace named `Submissions`;
- **Certified patterns**. Certified patterns belong to the ODP Portal namespace named `Catalogue`;
- **Beautiful ontologies**. Beautiful ontologies belong to the ODP Portal namespace named `Ontology`.

[5] http://ontologydesignpatterns.org/schemas/cpannotationschema.owl

[6] This feature, i.e. upload of ODP files and automatic generation of the pattern page, is currently only available for Content ODPs. At the moment this is the only pattern type which has a clearly defined OWL implementation. The ODP Portal community is working on providing more standards for annotating other kind of patterns (e.g. Correspondence Patterns).

While *submissions* contain all submitted patterns, *certified patterns* collect all patterns that are certified by the ODP Portal Quality Committee. The only difference between certified and proposed patterns is that the former are guaranteed to be fully described (with regard to ODP specification)[7], certified by the ODP Portal Quality Committee and thereby to some extent accepted by the community. An additional section, *beautiful ontologies*, has been recently added, for exposing and discussing well-designed ontologies.

Help This section groups user support (Figure 3.10(b)): general information about the portal, introduction to ontology patterns for novice users, the training area and user feedback.

About the initiative This section groups information about the ODP Portal initiative: people involved in the Quality Committee are listed, the partner institutions of the ODP Portal are named, and a news section for news bulletins about the portal (Figure 3.10(c)).

3.2.2 User types

Read and write access to the ODP Portal areas is handled by a policy based on five different types of users. The types of users are: *AnonymousUser*, *ODPUser*, *EBMember*, *QCMember*, *Administrator*, and they have the following interpretation:

- **Anonymous/unregistered users** *(AnonymousUser)*. These users have read-only access to the whole portal, and can request an account.

- **ODP Portal users** *(ODPUser)*. This user type represents everybody who has registered to the ODP Portal through the account request page (accounts are granted manually by an administrator), and hence is part of the ODP Portal community. Such users can create and edit articles in the Community area (namespace), post modelling issues, create and edit pages about domains, submit proposals, post feedback entries, submit open reviews about proposals, and participate in the discussions (through the discussion page).

- **Editorial Board** *(EBMember)*. Users in this group are administrators and developers of the portal, or specific areas of the portal. They answer user feedback and propose and contribute to content improvement through a page dedicated to development tasks[8]. The feedback area is used by Editorial Board members in order to identify possible development tasks. The development task page is directly used by core developers, and the Editorial Board members are entitled to manage task priority. Common tasks of the Editorial Board is for example to administrate and improve the ODP description templates and forms.

- **Quality Committee members** *(QCMember)*. These users are entitled to post *QC reviews* about ODP proposals. Certification of Content ODPs that are published in the official catalogue is based on QC reviews, and they are handled in a peer review-like approach. The QC evaluation workflow is managed by the Evaluation WikiFlow extension presented in Section 3.2.4.

- **Administrators** *(Administrator)*. Users in this group manage account creation, ODP Portal technical issues and upgrades.

[7]See also the CPAnnotationSchema: http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl
[8]http://ontologydesignpatterns.org/index.php/Odp:Development

3.2.3 ODP presentation

The ODP Portal software is based on Media Wiki[9], Semantic Media Wiki (SMW)[10], Semantic Forms (SF)[11], and other extensions [12]. We have exploited these extension features in order to semantically represent as much knowledge as possible about the ODPs. All wiki articles are assigned to a (semantic) category, and can be related to each other through semantic relations (properties). The core of the ODP Portal is composed of articles that represent ODPs. For example, for Content ODPs, according to [PG08], we have defined a set of semantic properties that comply to the *CPAnnotationSchema*[13]. These properties allow us to specify useful semantic information about Content ODPs, and have guided us in defining the visualization template of Content ODP pages. Other kinds of patterns have related information, according to their specificity:

- **Content ODPs** [14]: *Name; Submitted by; Also Known As; Intent; Domains; Competency Questions; Solution description; Reusable OWL Building Block; Consequences; Scenarios; Known Uses; Web References; Other References; Examples (OWL files); Extracted From; Reengineered From; Has Components; Specialization Of; Related CPs*.
- **Reengineering ODPs**[15]: *Name; SubmittedBy; Author; Also known as; Known uses; Related to; Other References; Problem; Non-Ontological Resource Description and Graphical Representation; Ontology Description and Graphical Representation; Process Description and Graphical Representation; Example (Scenario, Non-Ontological Resource, Ontology, Process);*
- **Alignment ODPs**: *Name; Also known as; Author(s); SubmittedBy; Domain (if applicable); Alignment problem addressed; Alignment solution; Alignment workflow; Reusable component; Problem example; Solution example; Example solution in the ontology alignment language; Consequences; Origin; Known use; Reference; Related to; Test.*
- **Logical ODPs**: *Name; SubmittedBy; Author; Also known as; Motivation; Aim; Solution Description; Elements; Implementation; Reusable component; Component type; Example (Problem, Solution, Consequences); Origin; Known use; Reference; Related ODP; Used in combination with; Test;*
- **Architectural ODPs**: *Name; SubmittedBy; Author; Also known as; Domain (if applicable); Problem description; Solution description; Implementation workflow; Reusable component; Example (Problem, Solution, Consequences); Origin; Known use; Reference; Related ODP; Diagram;*
- **Lexico-syntactic ODPs**[16]: *Name; Language; Also known as; Intent; Solution description; Related ODPs; Author(s); SubmittedBy; Submission date; Case form(s) (NL formulation, LSP formalization; Link to reusable JAPE code for NLP applications; Link to abbreviations and Symbols used in formalization)*

Consider for example Figure 3.12. It depicts the ODP Portal page of the Content ODP named *situation*, which is in the catalogue of proposals[17]. Under the diagrammatic representation (graphical illustration) of the Content ODP, to the center-right side of the page, there is a box containing Content ODP properties. They include the person who submitted the proposal, the name of the Content ODP, its intent, consequences, and typically associated competency questions, relations to other Content ODPs (such as its components, if it specializes other Content ODPs, and Content ODPs that are specializations of it), the URL of the downloadable OWL building block, the source from which it has been extracted, and so on. On the center-left side

[9] http://www.mediawiki.org
[10] http://www.semantic-mediawiki.org
[11] http://www.mediawiki.org/Extension:SemanticForm
[12] The full list of the extensions can be found at http://ontologydesignpatterns.org/wiki/Special:Version
[13] http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl
[14] see also Section 2.1.4 and 3.1.4
[15] see also Section 2.2
[16] see also Section 2.3
[17] http://ontologydesignpatterns.org/index.php/Submissions:Situation

of the page, all ontology elements defined in the Content ODP are listed and described. Furthermore, each of them has its own wiki article in the category `OntologyElement`. At the bottom of the page, there are links to example scenarios modeled using the Content ODP. Such examples includes a graphical illustration of the example, a natural language description, and a link to the OWL file that implements it. Finally, at the very bottom of the page, there are links to existing reviews of the patterns.

3.2.4 Pattern Evaluation ***Wiki*****Flow**

The Evaluation WikiFlow extension has been released in alpha version as open source software and can be downloaded from the MediaWiki wiki site[18]. The reader can test it on the ODP Portal, where it is associated to `ProposedCP` articles. Evaluation WikiFlow is designed in order to store a semantic representation of the evaluation history of an article. This feature is motivated by the goal of semantically representing the rationales behind an evaluation so as to identify recurrent mistakes and good practices (of ontology design in the case of the ODP Portal).
The Evaluation WikiFlow is substantiated by a tab, labelled *evaluation*, which is added on top of the wiki page, as shown in Figure 3.13. The Evaluation WikiFlow features can be described from two perspectives: configuration and functionality. Configuration features include the following:

Selection of article categories that are to be associated with the evaluation tab. This feature allows users to activate the evaluation tab and its functionalities (see later in this section) for a set of categories defined by the ontology behind the wiki, e.g. currently the ODP Portal activates it for the `ProposedCP` category.

Customization of the semantic form associated to the review page. This feature allows users to define the review schema associated to the corresponding semantic form.

Definition of different semantic review forms associated to different article categories. This feature allows users to associate different categories (or sets of categories) to semantic forms having different review schemas, e.g. in the ODP Portal Logical ODPs will be reviewed by a different form than Content ODPs.

Access rights configuration. Evaluation WikiFlow adds five new permissions to the existing ones. They can be configured in order to allow or prevent users to certify articles, and to perform the following actions on reviews: view, ask for, assign, and make, e.g. the ODP Portal *QCMember* users have the rights to 'make' a review, while every *ODPUser* can 'ask for' a review.

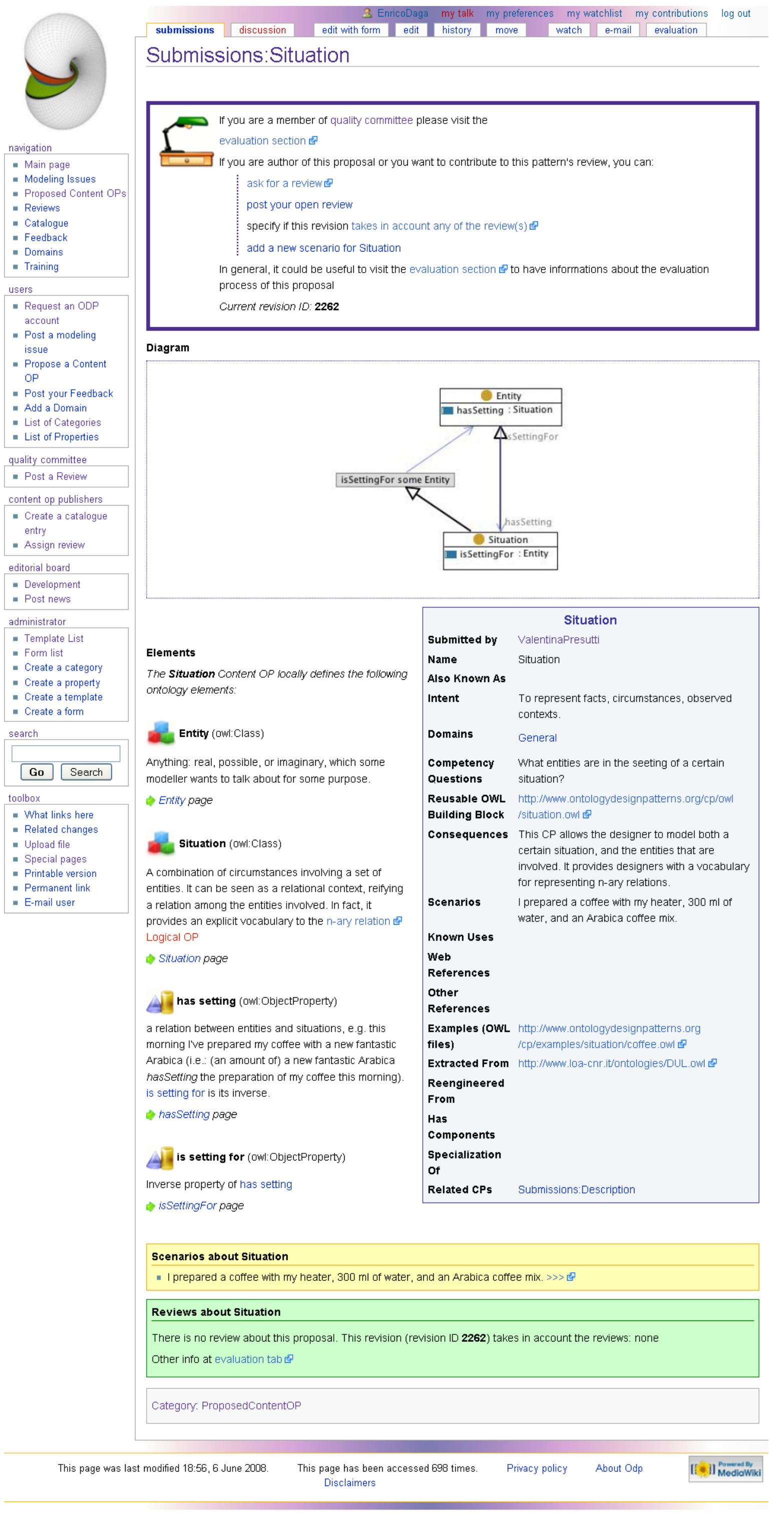
From the functionality side, the Evaluation WikiFlow provides the following features, accessible from the evaluation tab (see Figure 3.13):

Ask for review. This functionality allows users to ask for a review of the current article. Once this action has been performed, the article is automatically assigned to the category `WaitingForReview` that represents the current state of the article.

Assign review. This functionality allows users to commit another user to the task of reviewing an article, e.g. the in the ODP Portal *EBMember* users are allowed to perform this action in order to assign the reviewing of an ODP proposal to some *QCMember* user.

Make review. By performing this action, the reviewer is confronted with a review form and can post a review. A new article containing the review is created and named by concatenating the user name of the reviewer and the name of the article under evaluation. If the article under evaluation has the `WaitingForReview` category, this is updated by removing it. In the ODP Portal *QCMember* users can perform this action. A possible future enhancement of this feature would be to make the system

[18] http://www.mediawiki.org/Extensions:EvalWF



EnricoDaga my talk my preferences my watchlist my contributions log out

submissions | discussion | edit with form | edit | history | move | watch | e-mail | evaluation

Submissions:Situation

If you are a member of quality committee please visit the

evaluation section

If you are author of this proposal or you want to contribute to this pattern's review, you can:

- ask for a review
- post your open review
- specify if this revision takes in account any of the review(s)
- add a new scenario for Situation

In general, it could be useful to visit the evaluation section to have informations about the evaluation process of this proposal

Current revision ID: **2262**

Diagram

Elements

*The **Situation** Content OP locally defines the following ontology elements:*

Entity (owl:Class)

Anything: real, possible, or imaginary, which some modeller wants to talk about for some purpose.

Entity page

Situation (owl:Class)

A combination of circumstances involving a set of entities. It can be seen as a relational context, reifying a relation among the entities involved. In fact, it provides an explicit vocabulary to the n-ary relation Logical OP

Situation page

has setting (owl:ObjectProperty)

a relation between entities and situations, e.g. this morning I've prepared my coffee with a new fantastic Arabica (i.e.: (an amount of) a new fantastic Arabica *hasSetting* the preparation of my coffee this morning). is setting for is its inverse.

hasSetting page

is setting for (owl:ObjectProperty)

Inverse property of has setting

isSettingFor page

Situation

Submitted by	ValentinaPresutti
Name	Situation
Also Known As	
Intent	To represent facts, circumstances, observed contexts.
Domains	General
Competency Questions	What entities are in the seeting of a certain situation?
Reusable OWL Building Block	http://www.ontologydesignpatterns.org/cp/owl/situation.owl
Consequences	This CP allows the designer to model both a certain situation, and the entities that are involved. It provides designers with a vocabulary for representing n-ary relations.
Scenarios	I prepared a coffee with my heater, 300 ml of water, and an Arabica coffee mix.
Known Uses	
Web References	
Other References	
Examples (OWL files)	http://www.ontologydesignpatterns.org/cp/examples/situation/coffee.owl
Extracted From	http://www.loa-cnr.it/ontologies/DUL.owl
Reengineered From	
Has Components	
Specialization Of	
Related CPs	Submissions:Description

Scenarios about Situation

- I prepared a coffee with my heater, 300 ml of water, and an Arabica coffee mix. >>>

Reviews about Situation

There is no review about this proposal. This revision (revision ID **2262**) takes in account the reviews: none

Other info at evaluation tab

Category: ProposedContentOP

navigation
- Main page
- Modeling Issues
- Proposed Content OPs
- Reviews
- Catalogue
- Feedback
- Domains
- Training

users
- Request an ODP account
- Post a modeling issue
- Propose a Content OP
- Post your Feedback
- Add a Domain
- List of Categories
- List of Properties

quality committee
- Post a Review

content op publishers
- Create a catalogue entry
- Assign review

editorial board
- Development
- Post news

administrator
- Template List
- Form list
- Create a category
- Create a property
- Create a template
- Create a form

search

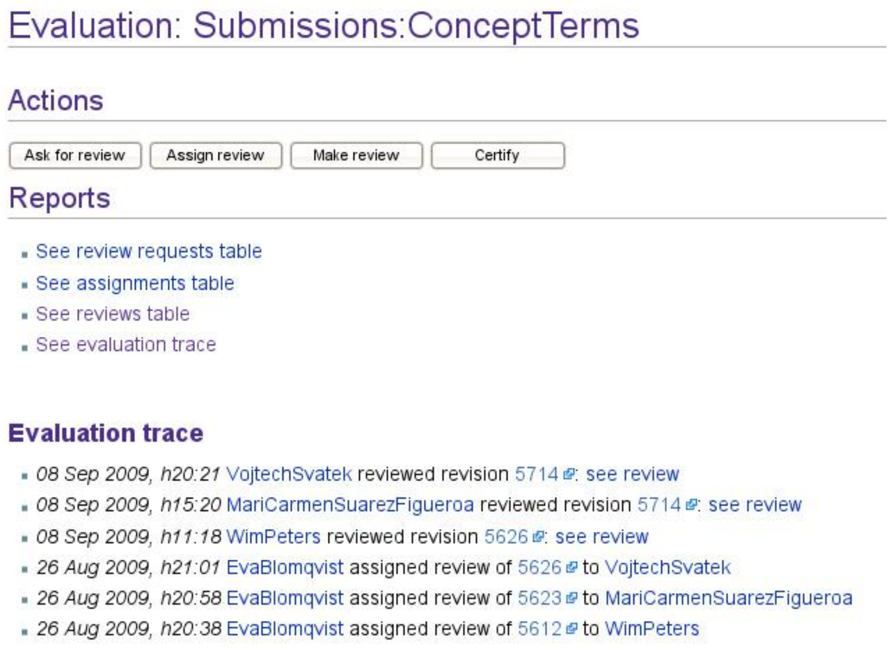
Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- E-mail user

This page was last modified 18:56, 6 June 2008. This page has been accessed 698 times. Privacy policy About Odp Disclaimers

Powered By MediaWiki

Figure 3.12: The *situation* Content ODP proposal as it appears on its page in the ODP Portal.


Evaluation: Submissions:ConceptTerms

Actions

Reports

- See review requests table
- See assignments table
- See reviews table
- See evaluation trace

Evaluation trace

- *08 Sep 2009, h20:21* VojtechSvatek reviewed revision 5714: see review
- *08 Sep 2009, h15:20* MariCarmenSuarezFigueroa reviewed revision 5714: see review
- *08 Sep 2009, h11:18* WimPeters reviewed revision 5626: see review
- *26 Aug 2009, h21:01* EvaBlomqvist assigned review of 5626 to VojtechSvatek
- *26 Aug 2009, h20:58* EvaBlomqvist assigned review of 5623 to MariCarmenSuarezFigueroa
- *26 Aug 2009, h20:38* EvaBlomqvist assigned review of 5612 to WimPeters

Figure 3.13: The WikiFlow evaluation tab on an ODP Portal page.

aware of additional statuses of an article, e.g. a status associated to the number reviews compared to the number of assignments, so that when an article has received all expected reviews it enters a `WaitingForRevision` status and the system asks the author(s) to revise the article according to all reviews.

Certify. This functionality allows users to certify an article. Once an article is certified it is *freezed*, i.e. it cannot be modified anymore. Furthermore, in the evaluation tab of the article the evaluation history is reported (see next bullet). The user has the possibility to automatically create a copy of the article, such a copy will have a lifecycle completely separate from the parent article, e.g. a copy of a certified Content ODP proposal is created in the ODP Portal `Catalogue` namespace and can be modified in order to indicate a persistent identifier for the building block.

Semantic report of evaluation history. The evaluation tab of an article is dynamically updated with its evaluation history. Each review of the article is associated to two article versions: the one under evaluation and the updated version of the article that takes the review into account (if present). The editor of the article can express the association "takes into account" between the revised article and the addressed review, by means of a "one click" functionality. The semantic report of the evaluation history exploits the MediaWiki versioning system. In the context of the ODP Portal, our aim is to use the evaluation history as a source for analyzing the rationales behind the evaluation of ODPs as well as for extracting good practices and common mistakes, e.g. anti-patterns [PGD$^+$08], in ontology design.

In order to exemplify the Evaluation WikiFlow extension, we here describe its instantiation in the ODP Portal case (see Figure 3.14). The evaluation tab is used in order to support the certification process of patterns. After posting an ODP proposal, an ODP Portal user can submit it to the certification process by asking for a review, i.e. by clicking the `ask for review` button in the evaluation tab. The ODP proposal is added to the list of ODPs that are waiting for reviews (the category `WaitingForReview` is assigned to the ODP). The list of ODPs waiting for a review is handled by the editorial board.

An *EBMember* user eventually assigns the reviewing task to at least two *QCMember* users. When a review is created, the category of the ODP proposal is updated, i.e. `WaitingForReview` is removed from the list of the ODP's categories, and a link to the review is added in the ODP proposal page. The author of the ODP

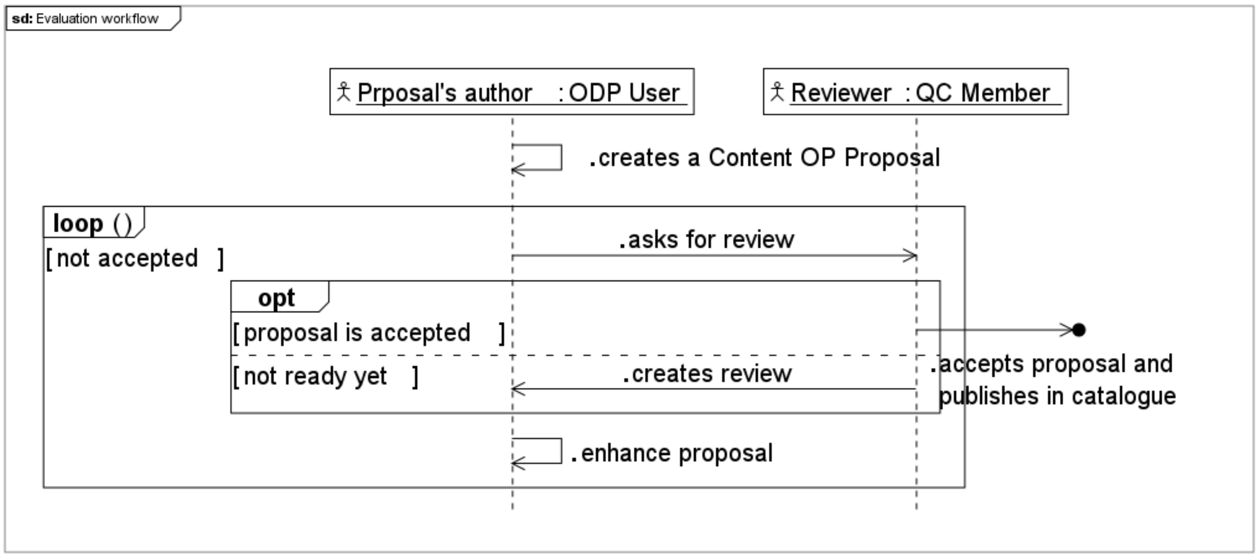


Figure 3.14: Evaluation example.

can modify it in order to address the issues pointed out in the review. Once the author believes that a certain version of the ODP addresses one or more reviews, the user can associate such a version to the reviews using the relation `takesIntoAccountReviews`[19].

This process is iterated until reviewers agree on the certification of the ODP. Certified ODPs are copied into the official catalogue, i.e the `Catalogue` namespace (see example in Figure 3.15).

3.2.5 The ODP Portal pattern registry

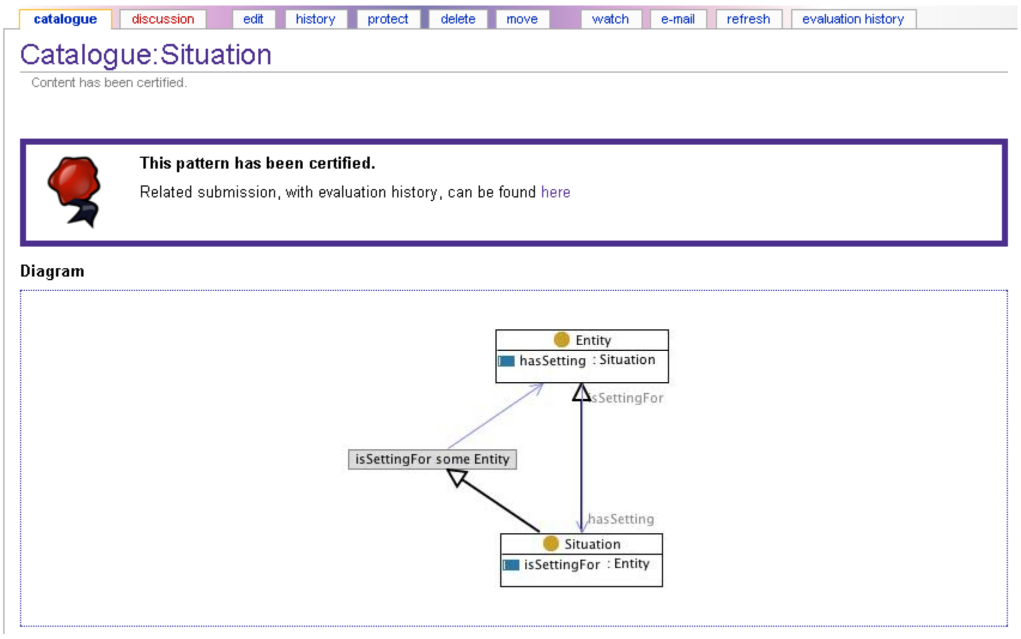
The ODP portal exposes a registry of all the OWL building blocks as OWL files, according to the guidelines defined in Section 3.1.7 and for this purpose it is available within the ODP Registry browser of XD Tools. The registry groups patterns as follows: **Catalogue** - all certified patterns (further grouped by type, e.g. Content ODPs, Logical ODPs, and so on); **Submissions** - all patterns not yet certified (further grouped by type, e.g. Content ODPs, Logical ODPs, and so on); **Domains** - pattern resources grouped by domain, e.g. General (for top-level patterns), Academy, Market, Fishery, and so on. The ODP Portal pattern registry can be downloaded from this address: http://ontologydesignpatterns.org/schemas/registry.owl.

3.2.6 Outcome and future work

The ODP Portal has currently more then 150 registered users. It has been used in the first Workshop on Ontology Patterns[20] as platform for the presention of patterns' proposals and for the evaluation process. It is our intent to go on with initiatives for involving the ODP community in the future. From the software perspective, currently the portal supports the automatic generation of a Content ODP submission page from the encoded OWL file. Equivalent mechanisms will be provided for other kind of patterns that can have a formal serialization (and annotation) format. Finally, a section dedicated to ontology design pattern registries published on the web is in plan.

[19]This is a "one click" operation.

[20]WOP2009 (http://ontologydesignpatterns.org/wiki/WOP2009:Main) at the 8th International Semantic Web Conference (ISWC 2009, http://iswc2009.semanticweb.org/).



Catalogue:Situation

Content has been certified.

This pattern has been certified.

Related submission, with evaluation history, can be found here

Diagram

Figure 3.15: A certified pattern.

3.3 Re-engineering ODPs: a software library

In this section we present the Re-engineering ODP software library, a Java library that implements the transformation process suggested by the Re-engineering ODPs, described in section 2.2. A high level conceptual architecture diagram of the modules involved is shown in Figure 3.16. The implementation of this library follows a modular approach, therefore it is possible to extend it to include other types of non-ontological resources, data models, and implementations in a simple way, as well as exploiting other external resources for relation disambiguation.

Figure 3.16 depicts the modules of the Re-engineering ODP software library: NOR Connector, Transformer, Semantic Relation Disambiguator, External Resource Service, and OR Connector. In the following sections these modules are described in detail.

3.3.1 NOR Connector

The Non-Ontological Resource (NOR) Connector is in charge of loading the non-ontological resource elements. According to the non-ontological resource categorization presented in D2.2.2 [VTAGS$^+$08], this module is able to deal with classification schemes, thesauri, and lexicons modelled with their corresponding data models, and implemented in databases, XML, flat files and spreadsheets.

This module utilizes an XML configuration file for describing the non-ontological resource. The main sections of the description are:

- *Schema* section. The schema entities of the resource and the relationships among the entities are described in this section.
- *DataModel* section. The descriptions of the resource's internal data model are included in this section.
- *Implementation* section. The information needed to physically access the resource is defined in this section.

An example of the XML configuration file is presented in Figure 3.17. The Figure shows that the file describes a classification scheme. The classification scheme has one schema entity, it is modelled following the path enumeration data model, and is implemented in a MSAccess database.

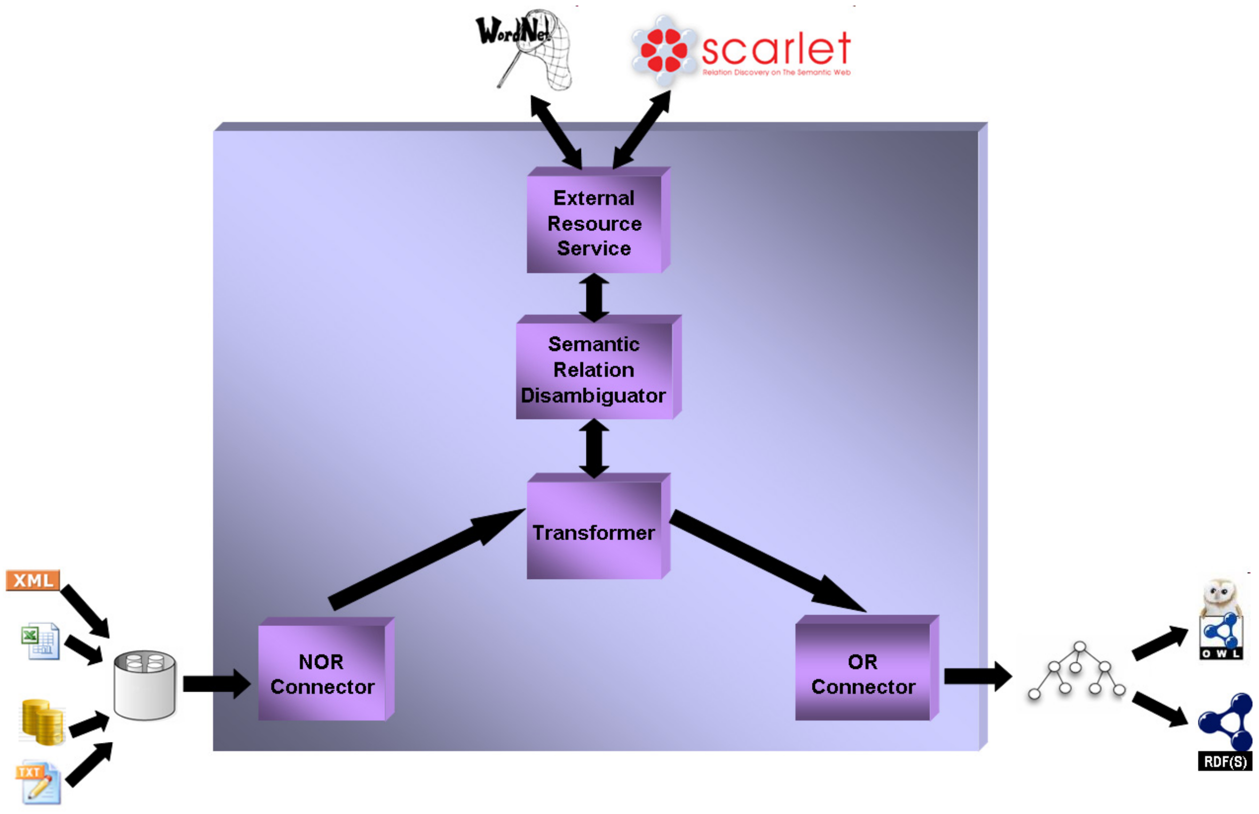

WordNet
scarlet
Relation Discovery on The Semantic Web
External Resource Service
Semantic Relation Disambiguator
Transformer
XML
TXT
NOR Connector
OR Connector
OWL
RDF(S)

Figure 3.16: Modules of the Re-engineering ODP software library.

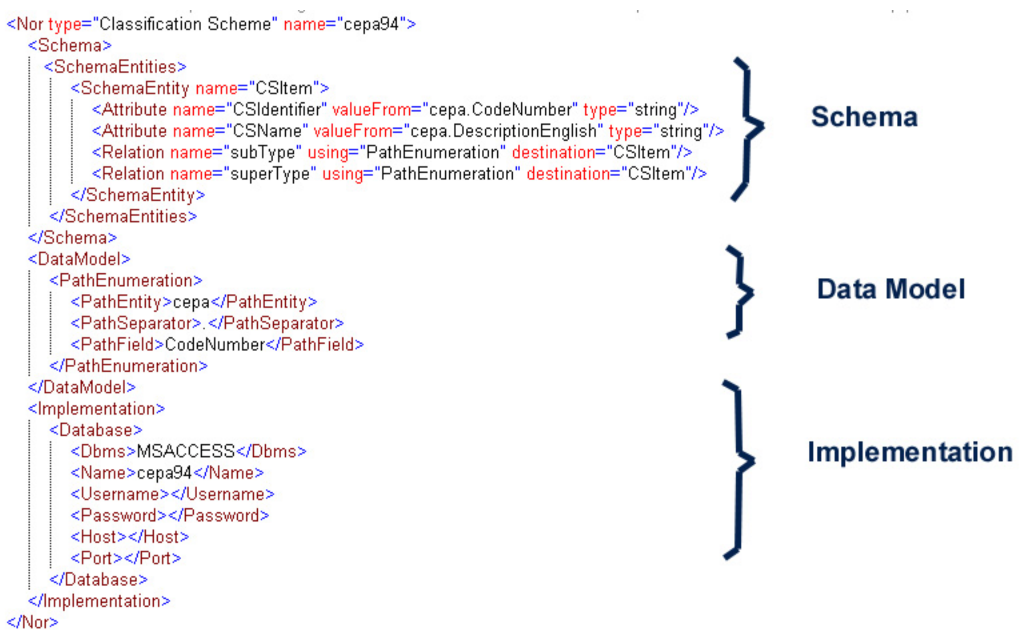

```
<Nor type="Classification Scheme" name="cepa94">
  <Schema>
    <SchemaEntities>
      <SchemaEntity name="CSItem">
        <Attribute name="CSIdentifier" valueFrom="cepa.CodeNumber" type="string"/>
        <Attribute name="CSName" valueFrom="cepa.DescriptionEnglish" type="string"/>
        <Relation name="subType" using="PathEnumeration" destination="CSItem"/>
        <Relation name="superType" using="PathEnumeration" destination="CSItem"/>
      </SchemaEntity>
    </SchemaEntities>
  </Schema>
  <DataModel>
    <PathEnumeration>
      <PathEntity>cepa</PathEntity>
      <PathSeparator>.</PathSeparator>
      <PathField>CodeNumber</PathField>
    </PathEnumeration>
  </DataModel>
  <Implementation>
    <Database>
      <Dbms>MSACCESS</Dbms>
      <Name>cepa94</Name>
      <Username></Username>
      <Password></Password>
      <Host></Host>
      <Port></Port>
    </Database>
  </Implementation>
</Nor>
```

Figure 3.17: NOR Connector configuration file example.

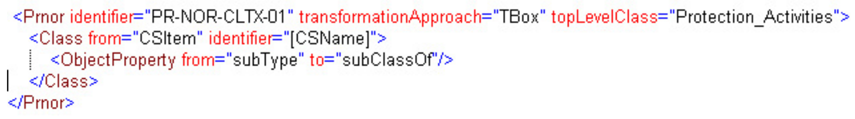
3.3.2 Transformer

This module is in charge of performing the transformation suggested by the patterns, by implementing the sequence of activities included in the patterns. The module transforms the non-ontological resource elements, loaded by the NOR Connector module, into ontology elements. This module interacts with the Semantic Relation Disambiguator module for obtaining the suggested semantic relations of the non-ontological resource elements.

This module also utilizes an XML configuration file for describing the transformation between the non-ontological resource elements and the ontology elements. The main section of this XML configuration file is:

- *PRNOR* section. The description of the transformations are included in this section. This description includes the transformation from the non-ontological resource schema components (e.g. schema entities, attributes and relations) into the ontology elements (e.g. classes, objectproperties, dataproperties and individuals). Additionally, it indicates the transformation approach[21] (e.g. TBox, ABox or Population).

An example of the XML configuration file is shown in Figure 3.18. The Figure indicates that the pattern follows the TBox transformation approach and it transforms the elements of the *CSItem* schema component into ontology classes. Also, it transforms the *subType* schema relation into a *subClassOf* relation, unless the Semantic Relation Disambiguator module suggests another relation.



```
<Prnor identifier="PR-NOR-CLTX-01" transformationApproach="TBox" topLevelClass="Protection_Activities">
  <Class from="CSItem" identifier="[CSName]">
    <ObjectProperty from="subType" to="subClassOf"/>
  </Class>
</Prnor>
```

Figure 3.18: PRNOR configuration file example.

3.3.3 Semantic Relation Disambiguator

This module is in charge of obtaining the semantic relation between two non-ontological resource elements. Basically, the module receives two non-ontological resource elements from the Transformer module and returns the semantic relation between them. First the module verifies whether it can obtain the *subClassOf* relation by identifying attribute adjetives[22] within the two given elements of the resource. If is not the case, the module connects the external resource through the External Resource Service module.

3.3.4 External Resource Service

This module is in charge of interacting with external resources for obtaining the semantic relations between two non-ontological resource elements. At this moment the module interacts with the following resources:

- Scarlet[23], which is a technique for discovering relations between two concepts by making use of ontologies available online. Scarlet provides a Java API to access the relations discovered between two concepts. The External Resource Service uses this API for retrieving the semantic relation.

[21]The transformation approaches are described in D2.2.4 [ALVT09]

[22]Attributive adjectives are part of the noun phrase headed by the noun they modify; for example, happy is an attributive adjective in "'happy people"'. In English, attributive adjectives usually precede their nouns in simple phrases, but often follow their nouns when the adjective is modified or qualified by a phrase acting as an adverb.

[23]`http://kmi.open.ac.uk/technologies/name/Scarlet`

- WordNet[24], which is an electronic lexical database created at Princeton University. The WordNet organizes the lexical information into meanings (senses) and synsets. What makes WordNet remarkable is the existence of various relations between the word forms (e.g. lexical relations, such as synonymy and antonymy) and the synsets (meaning to meaning or semantic relations e.g. hyponymy/hypernymy relation, meronymy relation). The External Resource Service uses the JWNL[25] Java API for accessing the WordNet. This External Resource Service deals with the following WordNet relations:
  - hyponym relation, interpreted as *subClassOf*.
  - hypernym relation, interpreted as *superClassOf*.
  - member meronym relation, interpreted as *partOf*.
  - member holonym relation, interpreted as *hasPart*.

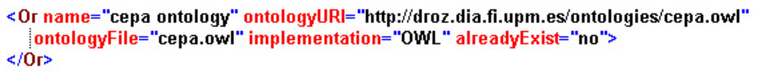
3.3.5 OR Connector

The Ontological Resource Connector is in charge of generating the ontology in an implementation language. The module is currently able to generate ontologies implemented in OWL. To this end, this module is relying on the OWL API[26].

This module also utilizes an XML configuration file for describing the transformation between the non-ontological resource elements and the ontology elements. The main section of this XML configuration file is:

- *OR* section. The descriptions of the generated ontology are included in this section. These descriptions include the name, URI, file and implementation language of the ontology. Additionally, this section indicates if the ontology already exists, in the case we want to populate an existing ontology.

An example of the XML configuration file is shown in Figure 3.19. The Figure indicates that the ontology generated will be stored in the *cepa.owl* file, its name will be *cepa ontology* and it will implemented in OWL.



```
<Or name="cepa ontology" ontologyURI="http://droz.dia.fi.upm.es/ontologies/cepa.owl"
    ontologyFile="cepa.owl" implementation="OWL" alreadyExist="no">
</Or>
```

Figure 3.19: Ontological Resource Connector configuration file example.

3.3.6 Prospective further work

As further work, we are applying a follow-up study to investigate precisely both quality and impact of the external resources used for discovering and disambiguating the relations among the non-ontological resource entities. We are planning to include DBpedia[27] as an additional external resource.

We are carrying out a user-based evaluation concerning the use of the Re-engineering ODP software library. The goal of this evaluation is to gather evidence on whether the usage of this software library leads to users being able to design ontologies faster and/or better conforming to quality standards. This evaluation is reported in [ALVT09].

[24] `http://wordnet.princeton.edu/`

[25] `http://sourceforge.net/projects/jwordnet/`

[26] The OWL API is a Java interface implementation of the W3C Web Ontology Language OWL. The latest version of the API is focused towards OWL 2 which encompasses, OWL-Lite, OWL-DL and some elements of OWL-Full. `http://owlapi.sourceforge.net/`

[27] DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web. `http://dbpedia.org/About`

3.4 Lexico-Syntactic Patterns: reusable JAPE code for NLP applications

As already described in Section 2.3.1, our approach for the reuse of ODPs involves the annotation of the input provided by the novice ontology designer in Natural Language (NL), and its comparison against the respository of Lexico-syntactic ODPs. If a matching is obtained between the annotations of the input and one of the Lexico-syntactic ODPs, the ODP or ODPs associated to the Lexico-syntactic ODP are returned to the user as the solution for the modelling problem expressed by him or her in NL.

With the aim of performing the annotation of the NL input, we decided to use GATE[28]. GATE, the General Architecture for Text Engineering, is a framework for the development and deployment of software components for Natural Language Processing (NLP). Its basic component, ANNIE, is an information extraction (IE) system that relies on basic processing resources. Additionally, GATE has a large number of plugins that can be combined to create different NLP applications. For the purposes of this research we relied on the annotations provided by ANNIE and some additional processing resources that will be described below. One of the most important processing resources contained in ANNIE is the JAPE transducer. JAPE stands for Java Annotation Patterns Engine and is a grammar that “provides a finite state transduction over annotations based on regular expressions”, as documented in [CMT00]. To put it in simple words, JAPE allows users to identify certain structures or patterns in documents relying on available annotations (previously provided by other processing resources in GATE), and creates new annotations. In this way, GATE provided us with the necessary fuctionalities to perform the annotation step (step 2) in the approach for semi-automatic reuse of ODPs (see 2.3.1).

In Section 3.4.1, our aim is to give a brief report of the processing resources used in this approach to analyze the information in NL, specifically the JAPE rules. For the actual implementation task of the JAPE rules, people from the UPM team visited the Natural Language Processing Group at the University of Sheffield. Then, in appendix B, we provide the JAPE rules created for the identification of Lexico-syntactic ODPs in NL sentences. The JAPE code provided is also expected to be made available at the ODP portal[29], so that it can be reused, extended or improved in further NLP applications.

3.4.1 NL processing with GATE

As already mentioned, for the purposes of this research we relied on ANNIE, GATE’s basic component, to obtain annotations that would be used by the JAPE transducer to create new annotations with the purpose of identifying Lexico-syntactic ODPs.

ANNIE contains the following processing resources (PRs):

- Tokeniser: divides the text into tokens, such as number, punctuation, or word, and adds a Token annotation to it.
- Sentence Splitter: divides the text into sentences.
- POS-Tagger: adds part-of-speech information to Token annotations.
- Gazetteer: contains lists of words grouped into categories to perform Named Entity (NE) recognition or key phrase lookups.
- Orthomatcher: adds identity relations between NEs found by the NE Transducer, to perform co-reference resolution.
- JAPE transducers: execute JAPE rules to create complex annotations based on the results of the previous PRs.

[28]http://gate.ac.uk/
[29]http://www.ontologydesignpatterns.org

To the resources contained in ANNIE, we needed to add further processing resources also available in GATE to obtain additional annotations, namely:

- Morphological Analyser: adds morphological information (lemma and affixes of words).
- Noun Phrase Chunker: identifies noun phrases.
- Flexible Gazetteer: a gazetteer that allows us to create our own lists of NEs or keywords to perform lookups.

Once we had selected the processing resources to be used in our application, we created a pipeline to determine the execution order, which was: Tokeniser, Sentence Splitter, POS tagger, Morphological Analyser, Noun Phrase Chunker, Flexible Gazetteer, and JAPE transducer. As can be observed, the last processing resource to be run is the JAPE transducer that relies on the previous annotations to create complex annotations as a result of the execution of the rules.

As reported in the GATE User Guide [CMB$^+$09], JAPE consists of a set of phases, each of which contains pattern/action rules. JAPE rules are divided into two parts: the so-called left-hand-side (LHS) of the rule, which contains the annotation pattern, and the right-hand-side (RHS) of the rule, which contains the "annotation manipulation statements". This means that the LHS of the rule needs to match certain annotation patterns in the document, so that the RHS can perform a certain action.

This is the structure of a JAPE rule: The LHS of the JAPE rule is separated from the RHS by the symbol >. The LHS relies on annotations, and optionally, on its features and values. Any annotation to be used must be included in the input header, and is to be enclosed in curly braces. Let us take the following example:
(Token.string == "of"), in which the pattern aims at identifying the string "of". The attributes and values of an annotation can as well be specified, as in:
`(Token.kind == "number")`, in which the annotation token with its feature "kind" and value "number". Besides the possibility of expressing annotations or strings of text, the presence or absence of annotations can also be indicated. For instance, if we want to match a Lookup annotation with the feature "minorType", which is the annotation resulting from the Gazetteer, we should express it in the following way:
`(Lookup.minorType == be).`
On the other hand, if we want the LHS to match whenever the same annotation is not present, we will have to express it by means of the symbol `!`, as in `(!Lookup.minorType == be).`

The LHS of the rule also permits the employment of the traditional Klene operators to combine annotation patterns:

- | meaning "or"
- * meaning 0 or more occurrences
- **+** meaning 1 or more occurrences
- **?** meaning 0 or 1 occurrences

Finally, each pattern to be matched in the LHS is enclosed in round brackets and can have a label attached, as in the case of (NounChunk):subclass, in which we specify that the annotation "NounChunk" identified in the pattern is assigned the label "subclass", which is to be used in the RHS of the rule, as we will see below.

The RHS uses **;** as a separator and has the following format:
(LHS)
–>
Annotation type; attribute1=value1; ...; attribute n=value n

Now, we will try to explain this by means of an example. Below we have included the JAPE rule we created for the Lexico-syntactic ODP 1 in LSP-SC-EN A.1:

[(NP<subclass>,)* and] NP<subclass> be [CN-CATV] NP<superclass>

The first step is to create a matching pattern for a list of noun phrases. Since a list of noun phrases is a recurrent pattern in the set of Lexico-syntactic ODPs, GATE developers recommend us to create a Macro with the name LIST. In this pattern we specify that whenever a list of noun phrases followed by a comma appears in the text (from 0 to 10 times) followed optionally by a comma before a coordinating conjunction ("and" in this case), and a further noun phrase, then the pattern LIST is matched. `Macro:LIST`

```
(
(NounChunkToken.string == ",")[0,10]
NounChunk
(Token.string == ",")?
Token.category == CC
NounChunk
)
```

The macro LIST will be the first element of the LHS in this JAPE rule for Lexico-syntactic ODP 1 in the pattern LSP-SC-EN. Each of the noun phrases identified here will correspond to the subclasses of the pattern, hence the label assigned to the LIST. Then, we specify a lookup annotation with the feature "minorType" and value "be". After this, a determiner can be optionally matched. Then, another lookup annotation, this time with feature "majorType" and value CN (Class Name) should be matched. This gazetteer includes a set of generic names for classes or concepts such as "type of", "group of", "kind of", "class of". Finally, a last noun phrase has to be matched that will correspond to the superclass. After this, the RHS is specified separated by `->`. This symbol is followed by the label assigned in the LHS "superclass" and the name of the new annotation (Superclass) separated by a dot. Then, JAVA code is used for the second part of the RHS, and for the specification of the second new annotation (Subclass).

```
(
(LIST):subclass
Lookup.minorType == be
(Token.category == DT)?
Lookup.majorType == CN
(NounChunk):superclass
)

->
:superclass.Superclass = rule="SC1_1",
{
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("subclass"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
{ Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

{ FeatureMap features = Factory.newFeatureMap();
// change this for a different rule name"
features.put("rule", "SC1_1");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),"Subclass",features);}}}
```

Finally, in appendix B, we offer the rest of the JAPE rules created to match some of the patterns defined in Section 2.3.2. This code is expected to be made available at the *www.ontologydesignspattern.org* portal in the next months, although we keep updating and improving it. Each JAPE rules is preceded by the name of the rule (Rule:SC1_1), the *Formalization* from the patterns repository, and some examples in NL.

We plan to keep updating the repository of Lexico-Syntactic ODPs, as well as the corresponding reusable JAPE code. It is left as a future line of work the development and implementation of the supporting tool for the reuse of ODPs intended at novice users relying on Lexico-Syntactic ODPs.

Chapter 4

Conclusions

Pattern-based ontology design is a relatively new field. Studies have recently been conducted to empirically prove the methodological benefits that these approaches bring, but the studies (e.g. as presented in D5.6.2 [DSFGP+09]) have also shown that there is a lack of tool support for pattern-based design, as well as a need for detailed methodological guidelines. However, as we have shown in this deliverable, this lack of software support is now being remedied, in parallel with the development and refinement of elaborate methodologies.

NeOn contributes to this development with a pattern-based methodology (i.e. eXtreme Design) that covers all the phases of ontology development, but currently with particular focus on collaborative aspects and re-use of Content ODPs. Collaboration is inherent within this approach, and from this perspective the ODP portal provides a space for discussion but also for hosting of resources, in particular registries of patterns.

The use of patterns in the modelling process needs software tools to help users (both naïve users and experts) to fully benefit from the methodologies. The NeOn Toolkit plug-in - XD Tools - aims at providing end-user support for pattern-based modelling, particularly using Content ODPs, which represent successful modelling solutions ready to be reused.

Content ODPs can also be discovered to be partially instantiatited in ontologies, while important and useful axioms may be missing in the instantiation. Based on this problem, we have also presented an algorithm for the identification of Content ODP instantiations in ontologies along with a method to transfer axioms contained in Content ODPs into a target ontology, to support ontology refinement.

Other pattern types have different roles in pattern-based design. We have investigated the role of ODPs in re-engineering tasks and developed a software library targeted on the re-engineering of non-ontological resources. This library, which is easily extendable, is a starting point for supporting domain experts in the reuse of legacy resources to setup a semantic knowledge base. For naïve users natural language is the common way of expressing requirements. In this deliverable we have seen how Lexico-syntactic ODPs can be exploited for identifying other ODPs in natural language sentences, to help novice users in the modelling process. A method and a repository of Lexico-syntactic ODPs, with a reusable rule-base to be used in NLP applications, were presented.

The research on ODP-based software is, however, still at the very beginning. In this deliverable we have summarized the latest results in this area, within the context of the NeOn project, and thereby laid the groundwork for future developments.

Appendix A

Lexico-syntactic ODP repository

In the following we provide the tables of Lexico-syntactic ODPs and classify them according to the correspondence degree to ODPs, as explained in section 2.3.1. Lexico-syntactic ODPs have been grouped into 3 sets: **(1)** 1:1 correspondence: 1 Lexico-syntactic ODP - 1 ODP; **(2)** 1:N correspondence: 1 Lexico-syntactic ODP - combination of 2 or more ODPs; **(3)** 1:1/2 correspondence: 1 Lexico-syntactic ODP - pairwise disjoint ODPs. The set of Lexico-syntactic ODPs presented here will be available at the Ontology Design Patterns portal by the end of 2009. The JAPE code resulting from the implementation of English patterns in GATE will also be uploaded to the portal.

(1) 1:1 correspondence: 1 Lexico-syntactic ODP - 1 ODP (English)

Table A.1: LSPs corresponding to *subclass-of relation* ODP

LSP Identifier : LSP-SC-EN	
NeOn ODPs Identifier : LP-SC-01	
Formalization	
1	[(NP<subclass>,)* and] NP<subclass> be [CN-CATV] NP<superclass>
2	[(NP<subclass>,)* and] NP<subclass> classify as NP<superclass>
3	[(NP<subclass>,)* and] NP<subclass> (belong to)| (fall into) CN-CATV NP<superclass>
4	There are QUAN CN-CATV NP<superclass> PARA [(NP<subclass>,)* and] NP<subclass>
5	[A(n) | QUAN] example of | CN-CATV NP<superclass> be | include [PARA] [(NP<subclass>,)* and] NP<subclass>
Examples	
1	*An orphan drug is a type of drug.* *Odometry, speedometry and GPS are types of sensors.*
2	*Prefixes and suffixes are classified as affixes.*
3	*Thyroid medicines belong to the general group of hormone medicines.* *Starfish fall into the class Asteroidea.*
4	*There are several kinds of memory: fast, expensive, short term memory, and long-term memory.*
5	*Some examples of peripherals are keyboards, mice, monitors, printers, scanners, disk and tape drives, microphones, speakers, joysticks, plotters and cameras.* *Types of criteria for assessing applications are: quality, safety and efficacy.*

Table A.2: LSPs corresponding to *equivalence relation between classes* ODP

LSP Identifier : LSP-EQ-EN
NeOn ODPs Identifier : LP-EQ-01

Table A.2: LSPs corresponding to *equivalence relation between classes* ODP (follow-up)

Formalization	
1	NP<class> be (the same as/synonym of) | know as | call | (refer to as) NP<class>
Examples	
1	*Poison dart frogs are also known as poison-arrow frogs.*

Table A.3: LSPs corresponding to *object property* ODP

LSP Identifier : LSP-OP-EN	
NeOn ODPs Identifier : LP-OP-01	
Formalization	
1	NP<class> VB NEG(be | have | CATV | PART) NP<class>
Examples	
1	*Birds build nests.*

Table A.4: LSPs corresponding to *datatype property* ODP

LSP Identifier : LSP-DP-EN	
NeOn ODPs Identifier : LP-DP-01	
Formalization	
1	Property | characteristic| | attribute of NP<class> be [PARA] [(NP<property>,)* and] NP<property>
2	NP<class> be [(AP<property>,)*] and AP<property>
Examples	
1	*Properties of mammals are hair, sweat glands, milk, and giving live birth.*
2	*Metals are lustrous, malleable and good conductors of heat and electricity.*

Table A.5: LSPs corresponding to *disjoint classes* ODP

LSP Identifier : LSP-Di-EN	
NeOn ODPs Identifier : LP-Di-01	
Formalization	
1	NP<class> differ | be different | be differentiate from NP<class>
Examples	
1	*Non-opioid agents differ from opioid agents.* *Universal aspects of language are differentiated from language-specific ones.*

Table A.6: LSPs corresponding to *participation* ODP

LSP Identifier : LSP-PA-EN	
NeOn ODPs Identifier : CP-PA-01	
Formalization	
1	NP<object> participate | take part in | be involved in (NP<event>,)* and] NP<event>
Examples	
1	*Engineering project managers participate in writing specifications, researching, and selecting suppliers and materials.* *Players are involved in competitions.*

Table A.7: LSPs corresponding to *co-participation* ODP

LSP Identifier : LSP-PCP-EN	
NeOn ODPs Identifier : CP-PCP-01	
Formalization	
1	(NP<object>,)* and NP<object> participate | (take part) | (be involved) in [(NP<event>,)* and] NP<event>
2	NP<object> participate | (take part) | (be involved) with in [(NP<event>,)* and] NP<event>
Examples	
1	*Aldo Gangemi and Valentina Presutti participate in the ISWC 2007 conference.*
2	*Action Engine participates with Microsoft at 3GSM World Congress.*

Table A.8: LSPs corresponding to *specified values* ODP

LSP Identifier : LSP-SV-EN	
NeOn ODPs Identifier : LP-SV-01	
Formalization	
1	NP<feature> can|may be [(AP<value>,)*] or AP<value>
Examples	
1	*Size may be small, medium, or big.* *Business plans can be accepted, non-accepted, or in process of revision.*

Table A.9: LSPs corresponding to *multiple inheritance* ODP

LSP Identifier : LSP-MI-EN	
NeOn ODPs Identifier : LP-MI-01	
Formalization	
1	NP<subclass> be NP<superclass> and NP<superclass>
Examples	
1	*Amphibians are water-living and land-living animals.*

Table A.10: LSPs corresponding to *location* ODP

LSP Identifier : LSP-LO-EN	
NeOn ODPs Identifier : CP-LO-01	
Formalization	
1	NP<place> be | has (locate | find | set | situate| place | (a site)) in [(NP<location >,)* and] NP<location>
Examples	
1	*The school is located in Bocas Town.* *T-cadherin is located in the nucleus and in the centrosomes.*

Table A.11: LSPs corresponding to *object-role* ODP

LSP Identifier : LSP-OR-EN	
NeOn ODPs Identifier : CP-OR-01	
Formalization	
1	NP<object> (be used)| work | act |serve as [(NP<role >,)* and] NP<role>
Examples	

Table A.11: LSPs corresponding to *object-role* ODP (follow-up)

1	*Gold is used as the reflective layer on some high-end CDs* *Induced bronchus-associated lymphoid tissue serves as a general priming site for T cells.*

(2) 1:N correspondence: 1 Lexico-syntactic ODP - combination of 2 or more ODPs (English)

Table A.12: LSPs corresponding to *object property* and *universal restriction* ODPs

LSP Identifier : LSP-OP-UR-EN	
NeOn ODPs Identifier : LP-OP-01 + LP-UR-01	
Formalization	
1	NP<class> VB NEG(be | have | CATV | PART) just | only | exclusively NP<class>
Examples	
1	*Herbivore eat only plants.*

Table A.13: LSPs corresponding to *defined classes* and *subclass-of relation* ODPs

LSP Identifier : LSP-DC-SC-EN	
NeOn ODPs Identifier : LP-DC-01 + LP-SC-01	
Formalization	
1	[A | any] NP<subclass> be [a | any] NP<superclass> REPRO VB [(NP<class>,)* and] NP<class>
2	[A | any] NP<subclass> be [a | any] NP<superclass> PREP [(NP<class>,)* and | or NP<class>
3	[A | any] NP<subclass> REPRO VB [(NP<class>,)* and | or NP<class> be [a] NP<superclass>
4	[A | any] NP<subclass> PREP [(NP<class>,)* and] NP<class> be VB [a] NP<superclass>
Examples	
1	*A device is any machine or component that attaches to a computer.* *Non-narcotic analgesics are drugs that have principally analgesic, antipyretic, and anti-inflammatory actions.*
2	*A vegetarian pizza is a pizza without fish or meat.*
3	*A workflow that contains at least one business task is a business plan.*
4	*Animal with backbones are called vertebrates.*

Table A.14: LSPs corresponding to *subclass-of relation*, *disjoint classes* and *exhaustive classes* ODPs

LSP Identifier : LSP-SC-Di-EC-EN	
NeOn ODPs Identifier : LP-SC-01 + LP-Di-01 + LP-EC-01	
Formalization	
1	NP<superclass> be | CATV [either] NP<subclass> or | and NP<subclass>
2	NP<superclass> CATV CD CN-CATV [PARA] [(NP<subclass>,)*and] NP <subclass>
3	There are CD CN-CATV NP<superclass> [PARA] [(NP<subclass>,)* and] NP<subclass>
4	NP<superclass> be divided | separate in|into CD CN-CATV [PARA] [(NP<subclass >,)* and] NP<subclass>
Examples	
1	*Animals are either vertebrates or invertebrates.*
2	*Membrane proteins are classified into two major categories, integral proteins and peripheral proteins.* *Flat roofing materials fall into three major categories: built-up felt roofing, mastic asphalt and single-ply membranes.* *In BC classes can be grouped into four main areas: Philosophy, Science, History and Technologies and Arts.*

Table A.14: ELSPs corresponding to *subclass-of relation*, *disjoint classes* and *exhaustive classes* ODPs (follow-up)

3	*There are two types of narcotic analgesics: the opiates and the opioids.*
4	*Marine mammals are divided into three orders: Carnivora, Sirenia and Cetacea.*

(3) 1:1/2 correspondence: 1 Lexico-syntactic ODP - pairwise disjoint ODPs (English)

Table A.15: LSPs corresponding to *subclass-of relation*, or *simple part-whole relation* ODPs

LSP Identifier : LSP-SC-PW-EN	
NeOn ODPs Identifier : LP-SC-01 CP-PW-01	
Formalization	
1	NP<class> include [(NP<class >,)* and] NP<class>
2	NP<class> be divided | separate in|into [CN] [(NP<class >,)* and] NP<class>
Examples	
1	*Arthropods include insects, crustaceans, spiders, scorpions, and centipedes. (LP-SC-01)* *Reproductive structures in female insects include ovaries, bursa copulatrix and uterus. (CP-PW-01)*
2	*Seed producing plants are divided into angiosperms and gymnosperms. (LP-SC-01)* *Cells are divided into distinct sub-cellular compartments. (CP-PW-01)*

Table A.16: LSPs corresponding to *object property* or *datatype property* or *simple part-whole relation* ODPs

LSP Identifier : LSP-OP-DP-PW-EN	
NeOn ODPs Identifier : LP-OP-01 LP-DP-01 CP-PW-01	
Formalization	
1	NP<class> have NP<class>
Examples	
1	*Birds have feathers.* *Water areas have names in natural language.*

Table A.17: LSPs corresponding to *simple part-whole relation* or *constituency* or *componency* or *collection-entity* ODPs

LSP Identifier : LSP-PW-CONS-COM-CE-EN	
NeOn ODPs Identifier : CP-PW-01 CP-CONS-01 CP-COM-01 CP-CE-01	
Formalization	
1	[(NP<part>,)* and] NP<part> PART NP<whole>
2	NP<whole> PART [(NP<part>,)* and] NP<part>
3	NP<whole> be PART [CD] CN-PART [PARA] [(NP<part>,)* and] NP<part>
4	CN-PART NP<whole> be [PARA] [(NP<part>,)* and] NP<part>
5	NP<whole> include | (be divide in|into) | (be separate in|into) CD CN-PART [PARA] [(NP<part>,)* and] NP<part>
Examples	

Table A.17: LSPs corresponding to *simple part-whole relation* or *constituency* or *componency* or *componency* ODPs (follow-up)

1	*Proteins form part of the cell membrane.*
2	*Lysosomes contain enzymes.* *Most clays consist of flat particles.* *Water is made up of hydrogen and oxygen.* *The United Arab Emirates is a country composed of seven emirates or sheikdoms.*
3	*A state machine workflow is made up of a set of states, transitions, and actions.*
4	*The parts of a tree are the root, trunk(s), branches, twigs and leaves.*
5	*The cerebrum is divided into two major parts: the right cerebral hemisphere and left cerebral hemisphere.*

(1) 1:1 correspondence: 1 Lexico-syntactic ODP - 1 ODP (Spanish)

Table A.18: Lexico-syntactic ODPs corresponding to *subclassOf relation* ODP

Lexico-syntactic ODP Identifier : LSP-SC-ES	
NeOn ODP Identifier : LP-SC-01	
Formalization	
1	NP <subclass> se clasifica como NP<superclass>
2	NP<subclass> se clasifica dentro de [CN] NP<superclass>
Examples	
1	*La pimienta común (Piper nigrum) se clasifica como perteneciente al género Piper.*
2	*Esta grave enfermedad neurodegenerativa se clasifica dentro del grupo de las enfermedades hereditarias recesivas.*

(2) 1:N correspondence: 1 Lexico-syntactic ODP - combination of 2 or more ODPs (Spanish)

Table A.19: Lexico-syntactic ODPs corresponding to *subclassOf relation, disjoint classes and exhaustive classes* ODPs

Lexico-syntactic ODP Identifier : LSP-SC-Di-EC-ES	
NeOn ODP Identifier : LP-SC-01 + LP-Di-01 + LP-EC-01	
Formalization	
1	Los/las NP<superclass> se clasifican en|como | se dividen en [CD] [los/las siguientes] [CN] [PARA] [(NP<subclass>,)* and] NP<subclass>
2	Se distinguen CD CN de NP<superclass> : [(NP<subclass>,)* y] NP<subclass>
Examples	
1	*Los hongos se clasifican en cuatro grandes grupos: Ficomicetos, Ascomicetos, Basidiomicetos y Deuteromicetos.* *Las grasas se dividen en saturadas e insaturadas.*
2	*Se distinguen dos tipos de tilacoides: los tilacoides de las granas y los tilacoides del estroma.*

Appendix B

JAPE rules

```
Phase:  SCRules
Input:  Lookup Split Token NounChunk
Options:  control = appelt

Macro:  LIST

( (NounChunkToken.string == “,”)[0,10]
NounChunk
(Token.string == “,”)?
Token.category == CC
NounChunk
)
Macro:  ADJLIST


(
(Token.category == JJToken.string == “,”)[0,10
Token.category == JJ
(Token.string ==
Token.category == CC
Token.category == JJ
)
// 1
Rule:SC1

// NP<subclass> be [CN] NP<superclass>
// An orphan drug is a type of drug.

(
(NounChunk):subclass
Lookup.minorType == be
(Token.category == DT)?
Lookup.majorType == CN
(NounChunk):superclass
) ->
```

```
:subclass.Subclass = rule=SC1,
:superclass.Superclass = rule=SC1

// 2
Rule:SC1_1
// [(NP<subclass>,)* and] NP<subclass> be [CN] NP<superclass>
// Odometry, speedometry and GPS are types of sensors.
(
(LIST):subclass
Lookup.minorType == be
(Token.category == DT)?
Lookup.majorType == CN
(NounChunk):superclass
)
->
:superclass.Superclass = rule=”SC1_1”,
// “subclass” matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get(“subclass”));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals(“NounChunk”)) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name”
features.put(“rule”, “SC1_1”);

// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), “Subclass”,
features);

//3
Rule:SC2
//[(NP<subclass>,)* and] NP<subclass> (classify as) | (group in|into|as)
| (fall into) | (belong to) [CN] NP<superclass>
Thyroid medicines belong to the general group of hormone medicines.
Thyroid medicines are classified as hormone medicines.

( ((LIST)|NounChunk):subclass
(Lookup.minorType == be)?
Lookup.minorType == SCverbs
(Token.category == DT)?
(Token.category == JJ)?
(Lookup.majorType == CN)?
(NounChunk):superclass
)
->
:superclass.Superclass = rule="SC2",
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("subclass"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "SC2");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Subclass",
features);

// 4
Rule:SC3
// There are CD | QUAN [CN] NP<superclass> PARA [(NP<subclass>,)* and]
NP<subclass>
// There are two types of narcotic analgesics:  the opiates and the
opioids.
// There are several kinds of memory:  expensive memory, short term
memory, and long-term memory.

(
Token.category == EX
Lookup.minorType == be
(Lookup.majorType == numbers|Token.category == JJ)
Lookup.majorType == CN
(NounChunk):superclass
(Token.kind == punctuation)?
(LIST):subclass
)

->
```

```
:superclass.Superclass = rule=”SC3”,
// “subclass” matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get(“subclass”));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name”
features.put(“rule”, “SC3”);

// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), “Subclass”,
features);

// 5
Rule:SC4
// [A(n) | QUAN] example | examples | [CN] of NP<superclass> be |
include [(NP<subclass>,)* and] NP<subclass>
Some examples of peripherals are keyboards, mice, monitors, printers,
scanners, disk and tape drives, microphones, speakers, joysticks,
plotters and cameras.
Types of criteria for assessing applications are:  quality, safety and
efficacy.

(
(Token.category == DT)?
(Lookup.majorType == CN)?
(Token.root == example)?
(Token.category == IN)?
(Lookup.majorType == CN)?
(NounChunk):superclass
(Lookup.minorType == be|Lookup.majorType == include)
(Token.kind == punctuation)?
(LIST):subclass
)

->
:superclass.Superclass = rule=”SC4”,
// “subclass” matches LHS label
```

```
List annList = new ArrayList((AnnotationSet)bindings.get("subclass"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "SC4");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Subclass",
features);

//5
Rule:OP1
// NP<class> VB ň (be | have | CATV | PART) NP<class>
// Birds build nests.  All cows eat grass.

(
(Lookup.majorType == forAll)?
(NounChunk):class1
(Token.category == VB, !Lookup.minorType == CATV, !Lookup.minorType ==
SCverbs, !Lookup.minorType ==
PART|Token.category == VBP, !Lookup.minorType == be,!Lookup.minorType ==
have)
(Token.category == NN|Token.category == NNS|NounChunk):class2
)

->
:class1.Class = rule="OP1",
:class2.ObjectPropery = rule=OP1

// 5.1
Rule:OP_ER
Some NP<class> VB ň (be | have | CATV) NP<class>
Some fish species eat water plants.

(
(Lookup.majorType == some)?
(NounChunk):class1
```

```
(Token.category == VB, !Lookup.minorType == CATV, !Lookup.minorType ==
SCverbs|Token.category == VBP,
!Lookup.minorType == be,!Lookup.minorType == have)
(Token.category == NN|Token.category == NNS|NounChunk):class2
)
->
:class1.Class = rule="OP_ER",
:class2.ObjectProperyExisRes = rule=OP_ER

// 6
Rule:OP_UR
// NP<class> VB ň (be | have | CATV) just | only | exclusively NP<class>
// Herbivore only eat plants.

(
(NounChunk):class1
(Lookup.majorType == forAll)
(Token.category == VB, !Lookup.minorType == CATV, !Lookup.minorType
== SCverbs|Token.category == VBP, !Lookup.minorType == be,
!Lookup.minorType == have)
(NounChunk):class3
)
->
:class1.Class = rule=OP_UR,
:class3.ObjectPropertyUniRes = rule=OP_UR

// 7
Rule:OP_UR2
// Herbivore eat just plants.

(
(NounChunk):class1
(Token.category == VB, !Lookup.minorType == CATV, !Lookup.minorType
== SCverbs|Token.category == VBP, !Lookup.minorType == be,
!Lookup.minorType == have)
(Lookup.majorType == forAll)
(Token.category == NN|Token.category == NNS):class3
)

->
:class1.Class = rule=OP_UR2,
:class3.ObjectPropertyUniRes = rule=OP_UR2

// 8
Rule:DP
// Properties of mammals are hair, sweat glands, and milk.

(
Lookup.majorType == properties
(NounChunk):class
```

```
Lookup.minorType == be
(LIST):properties
)

->
:class.Class = rule=DP,
// "properties" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("properties"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)

Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "DP");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
"DataTypeProperty", features);

// 9
Rule:DP1
// Copper is red and though.

(
(NounChunk):class
Lookup.minorType == be
(Token.category == JJ|(ADJLIST)):properties
)

->
:class.Class = rule=DP1,

// "properties" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("properties"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());
```

```
//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)

Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if (anAnn.getType().equals("Token") &&
anAnn.getFeatures().containsKey("category") &&
anAnn.getFeatures().get("category").equals("JJ") )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "DP2");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
"DataTypeProperty",features);

// 10
Rule:OPDPPW
// Birds have feathers.  Water areas have names in natural language.

(
(NounChunk):class
Lookup.minorType == have
(LIST|NounChunk):properties
)

->
:class.Class = rule=OPDPPW,

// "properties" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("properties"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)

Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk

if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();
```

```
// change this for a different rule name"
features.put("rule", "OPDPPW");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
"ObjectPDataTPartW",features);

// 11
Rule:DC

// Non-opioid agents differ from opioid agents.

(
(NounChunk):class
(Lookup.minorType == be)?
Lookup.majorType == disjointness
(NounChunk):class4
)

->
:class.Class = rule=DC,
:class4.DisjointClasses = rule=DC

// 12
Rule:PW
// (NP<part>,)* and NP<part> COMP [CN] NP<whole>
// Proteins form part of the cell membrane.

(
(LIST|NounChunk):part
Lookup.minorType == COMP
(NounChunk):whole
)

->

// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )
```

```
FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "PW");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Part",
features);


,
:whole.Whole = rule=PW

// 13
Rule:PW1
// NP<whole> be COMP [CN] (NP<part>,)* and NP<part>
// A state machine workflow is made up of states, transitions, and
actions.

(
(NounChunk):whole
Lookup.minorType == be
Lookup.minorType == COMP
(LIST|NounChunk):part
)

->
:whole.Whole = rule=PW1,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "PW1");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Part",
features);
```

```
// 14
Rule:PW2
// The part | parts of a NP<whole> be [PARA] NP<part>,)* and NP<part>
// The parts of a tree are the root, the trunk, branches, twigs and
leaves.

(
Lookup.majorType == PN
(Token.category == DT)?
(NounChunk):whole
Lookup.minorType == be
(LIST):part
)

->
:whole.Whole = rule=PW2,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "PW2");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Part",
features);

// 15
Rule:PW3
// NP<class> include | comprise | consist of CD parts PARA [(NP<class
>,)* and] NP<class>
// Cluster bombs consist of two parts:  the dispenser and the payload.

(
(NounChunk):whole
(Token.category == IN)?
(NounChunk)?
```

```
Lookup.minorType == include
Lookup.majorType == numbers
Lookup.majorType == PN
(Token.kind == punctuation)?
(LIST):part
)

->
:whole.Whole = rule=PW3,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "PW3");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Part",
features);

// 16
Rule:PW4
// NP<class> be divide | split | separate in|into CD PN PARA [(NP<class
>,)* and] NP<class>
// The cerebrum is divided into two major parts:  the right cerebral
hemisphere and left cerebral hemisphere.

(
(NounChunk):whole
Lookup.minorType == be
Lookup.minorType == include
Lookup.majorType == numbers
Lookup.majorType == PN
(Token.kind == punctuation)?
(LIST):part
)

->
```

```
:whole.Whole = rule=PW4,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "PW4");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), "Part",
features);

// 17
Rule:SC_PW
// NP<class> include | comprise | consist of [(NP<class >,)* and]
NP<class>
// Common mass storage devices include disk drives and tape drives.
// Reproductive structures in female insects include ovaries, bursa
copulatrix and uterus.

(
(NounChunk):superclass_whole
(Token.category == IN)?
(NounChunk)?
Lookup.minorType == include
(LIST):subclass_Di_EC_part
)

->
:superclass_whole.SuperclassWhole = rule=SC_PW,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("subclass_Di_EC_part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
```

```
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "SC_PW");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
"Subclass_Di_ECPart", features);

// 18
Rule:SC_PW1
// NP<class> be divide | split | separate in|into [(NP<class >,)* and]
NP<class>
// Books are divided into chapters.

(
(NounChunk):superclass_whole
(Token.category == IN)?
(NounChunk)?
Lookup.minorType == be
Lookup.minorType == include
(LIST|NounChunk):subclass_Di_EC_part
)

->
:superclass_whole.SuperclassWhole = rule=SC_PW1,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("subclass_Di_EC_part"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "SC_PW1");
```

```
// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
“Subclass_Di_ECPart”, features);

// 19
Rule:SC_Di_EC
// NP<class> include | comprise | consist of CD CN PARA [(NP<class >,)*
and] NP<class>
// Actuated signals consist of two types:  semi-actuated signals and
fully actuated signals.

(
(NounChunk):superclass
(Token.category == IN)?
(NounChunk)?
Lookup.minorType == include
Lookup.majorType == numbers
Lookup.majorType == CN
(Token.kind == punctuation)?
(LIST):subclass_Di_EC
)

->
:superclass.Superclass = rule=SC_Di_EC,
// “subclass” matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get(“subclass_Di_EC”));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals(“NounChunk”)) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name”
features.put(“rule”, “SC_Di_EC”);

// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
“Subclass_Di_EC”, features);

// 20
Rule:SC_Di_EC1
// NP<class> be divide | split | separate in|into CD CN PARA [(NP<class
>,)* and] NP<class>
// Sensors are divided into two groups:  contact sensors and non-contact
sensors.

(
(NounChunk):superclass
(Token.category == IN)?
(NounChunk)?
Lookup.minorType == be
Lookup.minorType == include
Lookup.majorType == numbers
Lookup.majorType == CN
(Token.kind == punctuation)?
(LIST):subclass_Di_EC
)

->
:superclass.Superclass = rule=SC_Di_EC1,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get(“subclass_Di_EC”));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals(“NounChunk”)) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name”
features.put(“rule”, “SC_Di_EC1”);

// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
“Subclass_Di_EC”, features);

// 21
Rule:SC_Di_EC2
// NP<superclass> be either NP<subclass> or NP<subclass>
// Animals are either vertebrates, or invertebrates.  Necesitamos
la coma entre vertebrates y or, pq si no, lo identifica todo como
NounChunk.

(
(NounChunk):superclass
```

```
Lookup.minorType == be
(Token.category == CC)?
(NounChunk):subclass_Di_EC
(Token.kind == punctuation)?  Token.root == or
(NounChunk):subclass_Di_EC2
)

->
:superclass.Superclass = rule=SC_Di_EC2,
:subclass_Di_EC.Subclass_Di_EC = rule=SC_Di_EC2,
:subclass_Di_EC2.Subclass_Di_EC = rule=SC_Di_EC2

// 22
Rule:SC_Di_EC3
// NP<superclass> be CATV either NP<subclass> or NP<subclass>
// Animals are classified into either vertebrates, or invertebrates.

(
(NounChunk):superclass
Lookup.minorType == be
Lookup.minorType == CATV
(Token.category == CC)?
(NounChunk):subclass_Di_EC
Token.kind == punctuation
Token.root == or
(NounChunk):subclass_Di_EC2
)

->
:superclass.Superclass = rule=SC_Di_EC3,
:subclass_Di_EC.Subclass_Di_EC = rule=SC_Di_EC3,
:subclass_Di_EC2.Subclass_Di_EC = rule=SC_Di_EC3

// 23
Rule:SC_Di_EC4
// NP<superclass> be CATV CD CN PARA (NP<subclass>,)*and NP <subclass>
// Membrane proteins are classified into two categories:  integral
proteins and peripheral proteins.

(
(NounChunk):superclass
Lookup.minorType == be
Lookup.minorType == CATV
Lookup.majorType == numbers
Lookup.majorType == CN

(Token.kind == punctuation)?  (LIST):subclass_Di_EC
)

->
```

```
:superclass.Superclass = rule=SC_Di_EC6,
// "subclass" matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get("subclass_Di_EC"));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals("NounChunk")) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name"
features.put("rule", "SC_Di_EC6");

// change "Subclass" for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(),
"Subclass_Di_EC", features);

// 24
Rule:EQ
// NP<class> be (the same) as | (also) know as | (also) refer to as
called |synonymous with | synonym of NP<class>
// Poison dart frogs are also known as poison-arrow frogs.

(
(NounChunk):class
Lookup.minorType == be
(Token.category == RB)?
Lookup.majorType == equivalent
(NounChunk):equivalentClass
)

->
:class.Class = rule=EQ,
:equivalentClass.EquivalentClass = rule=EQ

// 25
Rule:Participate
// NP<object> participate | take part in | be involved in [(NP<event>,)*
and] NP<event> [in| from | during] [NP<time-interval> to //
NP<time-interval>]
// Project managers participate in business unit management and
marketing.
```

```
(
(NounChunk):participant
(Lookup.minorType == be)?
Lookup.minorType == participate
(LIST|NounChunk):event
)

->
:participant.Participant = rule=Participate,
// “subclass” matches LHS label
List annList = new ArrayList((AnnotationSet)bindings.get(“event”));

//sort the list by offset
Collections.sort(annList, new OffsetComparator());

//iterate through the matched annotations
for(int i = 0; i < annList.size(); i++)
Annotation anAnn = (Annotation)annList.get(i);

// check that the new annotation is a NounChunk
if ((anAnn.getType().equals(“NounChunk”)) )

FeatureMap features = Factory.newFeatureMap();

// change this for a different rule name”
features.put(“rule”, “Participate”);

// change “Subclass” for a different annotation name
annotations.add(anAnn.getStartNode(), anAnn.getEndNode(), “Event”,
features);

// 26
Rule:SC_DefinedClass
// [A | any] NP<subclass> be [a] NP<superclass> REPRO VB
[(NP<class/property >,)* and] NP<class>
// A device is any machine or component that attaches to a computer.

(
(NounChunk):subclass
Lookup.minorType == be
(NounChunk):superclass
Token.category == WDT
(Token.category == VB|Token.category == VBZ|Token.category == VBP)
(Token.category == TO)?
NounChunk
)

->
:subclass.SubclassDefined = rule=SC_DefinedClass,
:superclass.Superclass = rule=SC_DefinedClass

// 27
Rule:SC_DefinedClass1
// [A | any] NP<subclass> be [a] NP<superclass> REPRO VB
[(NP<class/property >,)* and] NP<class>
// A vegetarian pizza is a pizza without fish or meat.

(
(NounChunk):subclass
Lookup.minorType == be
(NounChunk):superclass
Token.category == IN
NounChunk
)

->
:subclass.SubclassDefined = rule=SC_DefinedClass1,
:superclass.Superclass = rule=SC_DefinedClass1

// 28
Rule:SC_DefinedClass2
// [A | any] NP<subclass> REPRO VB [(NP<class >,)* and] NP<class> be [a]
NP<superclass>
// A workflow that contains at least one business task is a business
plan.

(
(NounChunk):subclass
Token.category == WDT
(Token.category == VB|Token.category == VBZ|Token.category == VBP)
NounChunk
Lookup.minorType == be
(NounChunk):superclass
)

->
:subclass.SubclassDefined = rule=SC_DefinedClass2,
:superclass.Superclass = rule=SC_DefinedClass2

// 29
Rule:SC_DefinedClass3
// [A | any] NP<subclass> PREP [(NP<class/property >,)* and] NP<class>
be [a] NP<superclass>
// Animal with backbones are called vertebrates.

(
(NounChunk):subclass
Token.category == IN
NounChunk
Lookup.minorType == be
(Lookup.majorType == equivalent)?
(NounChunk):superclass
)

-->
:subclass.SubclassDefined = rule=SC_DefinedClass3,
:superclass.Superclass = rule=SC_DefinedClass3

// 30
Rule:ObjectRole
Some university researchers are also university lecturers.
Some researchers also work as university lecturers.

(
(Lookup.majorType == some)?
(NounChunk):object
(Token.category == RB)
(Lookup.minorType == be)?
Lookup.minorType == ObjectRole
(NounChunk):role
)
-->
:object.Object = rule=ObjectRole,
:role.Role = rule=ObjectRole

// 31
Rule:Place
// The Colosseum is located in Rome.

(
(NounChunk):object
Lookup.minorType == be
Lookup.minorType == place
(NounChunk):place
)
-->
:object.Object = rule=Place,
:place.Place = rule=Place
```

Bibliography

[ALVT09] S. Angeletou, Holger Lewen, and B. Villazón-Terrazas. NeOn Deliverable D2.2.4 Final version of methods for re-engineering and evaluation. Technical report, NeOn, 2009.

[BCR94] V. R. Basili, G. Caldiera, and D. Rombach. *Experience Factory*, pages 469–476. Wiley & Sons, 1994.

[Bec99] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[BGA+08] Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta. Watson: A gateway for next generation semantic web applications. 2008.

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

[CGPMPSF08] Guadalupe Aguado De Cea, Asunción Gómez-Pérez, Elena Montiel-Ponsoda, and Mari Carmen Suárez-Figueroa. Natural language-based approach for helping in the reuse of ontology design patterns. In Springer Berlin / Heidelberg, editor, *In Knowledge Engineering: Practice and Patterns, Proceedings of the 16th International Conference on Knowledge Engineering (EKAW)*, volume 5268/2008 of *LNCS*, pages pp. 32–47, 2008. ISBN 978-3-540-87695-3.

[CMB+09] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Marin Dimitrov, Mike Dowman, Niraj Aswani, Ian Roberts, Yaoyong Li, Adam Funk, Genevieve Gorrell, Johann Petrak, Horacio Saggion, Danica Damljanovic, and Angus Roberts. *Developing Language Processing Components with GATE Version 5 (a User Guide)*. The University of Sheffield, November 2009.

[CMT00] Hamish Cunningham, Diana Maynard, and Valentin Tablan. *JAPE: a Java Annotation Patterns Engine*. University of Sheffield, (second edition) edition, November 2000. Research Memorandum CS-00-10.

[CP97] Peter Clark and Bruce Porter. Building concept representations from reusable components. In *Proceedings of AAAI'97*, pages 369–376. AAAI press, 1997.

[CRBP09] Oscar Corcho, Catherine Roussey, Luis Manuel Vilches Blazquez, and Ivan Perez. Pattern-based owl ontology debugging guidelines. In *Proceedings of WOP2009 collocated with ISWC2009*, volume 516. CEUR-WS.org, November 2009.

[dCMPSF09] Guadalupe Aguado de Cea, Elena Montiel-Ponsoda, and Mari Carmen Suárez-Figueroa. Approaches to ontology development by non ontology experts. In *International Symposoum on Data and Sense Mining, Machine Translation and Controlled Languages (ISMTCL 2009)*, pages 23–29, 2009.

[DSFGP+09] Martin Dzbor, Mari Carmen Suarez-Figueroa, Asuncion Gomez-Perez, Eva Blomqvist, Holger Lewen, and Mauricio Espinoza. D5.6.2 experimentation with parts of neon methodology. Technical report, NeOn Project., 2009.

[EBC+09] Michael Erdmann, Claudio Baldassarre, Caterina Caracciolo, Aldo Gangemi, German Herrero Carcel, Tomas Pariente Lobo, and Wolfgang Schoch. D7.6.2 second prototype of the fisheries stock depletion assessment system (fsdas). Technical report, The NeOn Project., February 2009.

[Euz08] Jérôme Euzenat. Algebras of ontology alignment relations. In *International Semantic Web Conference*, pages 387–402, 2008.

[GF94] M. Gruninger and M. S. Fox. The role of competency questions in enterprise engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.

[GLP+05] Aldo Gangemi, Jos Lehmann, Valentina Presutti, Malvina Nissim, and Carola Catenacci. Codo: an owl metamodel for collaborative ontology design. In *Workshop on Social and Collaborative Construction of Structured Knowledge*, May 2007-05.

[GP09a] Aldo Gangemi and Valentina Presutti. D2.1.2the collaborative ontology design ontology (v2). Technical report, The NeOn Project., Available at: http://www.neon-project.org, February 2009.

[GP09b] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on Ontologies, 2nd Ed.*, International Handbooks on Information Systems. Springer, 2009.

[GSGPSFVT08] Andrés García-Silva, Asunción Gómez-Pérez, Mari Carmen Suárez-Figueroa, and Boris Villazón-Terrazas. A pattern based approach for re-engineering non-ontological resources into ontologies. In *ASWC '08: Proceedings of the 3rd Asian Semantic Web Conference on The Semantic Web*, pages 167–181, Berlin, Heidelberg, 2008. Springer-Verlag.

[MFP09] Diana Maynard, Adam Funk, and Wim Peters. Using lexico-syntactic ontology design patterns for ontology creation and population. In *Proceedings of WOP2009 collocated with ISWC2009*, volume 516. CEUR-WS.org, November 2009.

[MJ09] Wolfgang Maass and Sabine Janzen. A pattern-based ontology building method for ambient environments. In *Proceedings of WOP2009 collocated with ISWC2009*, volume 516. CEUR-WS.org, November 2009.

[MPdCGPSF08] Elena Montiel-Ponsoda, Guadalupe Aguado de Cea, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. Helping naive users to reuse ontology design patterns. In *In Proceedings of the 1st International Workshop on Knowledge Reuse and Reengineering over the Semantic Web (KRRSW 2008)*, 2008.

[PDGS08] Valentina Presutti, Enrico Daga, Aldo Gangemi, and Alberto Salvati. http: //ontologydesignpatterns.org [odp]. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters and Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[PG08] V. Presutti and A. Gangemi. Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008)*, Berlin, 2008. Springer.

[PGD+08] Valentina Presutti, Aldo Gangemi, Stefano David, Guadalupe Aguado de Cea, Mari-Carmen Suárez-Figueroa, Elena Montiel-Ponsoda, and María Poveda. A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. Deliverable D2.5.1, NeOn project, 2008.

[PGP+09] Wim Peters, Aldo Gangemi, Valentina Presutti, Dunja Mladenic, Raúl Palma, Klaas Dellschaft, Alessandro Adamou, Enrico Daga, Holger Lewen, Michael Erdmann, and Anne Becker. Practical methods to support collaborative ontology design. Deliverable D2.3.2, NeOn project, 2009.

[SBD+09] Mari Carmen Suárez Figueroa, Eva Blomqvist, Mathieu D'Aquin, Mauricio Espinoza, A sunción Gómez-Pérez, Holger Lewen, Igor Mozetic, Raúl Palma, Maria Poveda, Margherita Sini, Boris Villazón-Terrazas, Fouad Zeblith, and Martin Dzbor. Revision and extension of the neon methodology for building contextualized ontology networks. Deliverable D5.4.2, NeOn project, 2009.

[SE05] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. pages 146–171. 2005.

[SFBG+] Mari Carmen Suárez-Figueroa, Saartje Brockmans, Aldo Gangemi, Asunción Gómez-Pérez, Jos Lehmann, Holger Lewen, Valentina Presutti, and Marta Sabou. D5.1.1 neon modelling components. Technical report, UPM.

[SFBG+07] Mari Carmen Suárez-Figueroa, Saartje Brockmans, Aldo Gangemi, Asunción Gómez-Pérez, Jos Lehmann, Holger Lewen, Valentina Presutti, and Marta Sabou. D 5.1.1 neon modelling components. Technical report, NeOn Project, March 2007. Available at: http://www.neon-project.org.

[SFDMP+08] Mari Carmen Suárez-Figueroa, Klass Dellschaft, Elena Montiel-Ponsoda, Boris Villazon-Terrazas, Zehn Yufei, Guadalupe Aguado de Cea, Andrés García, Mariano Fernández-López, Asunción Gómez-Pérez, Mauricio Espinoza, and Marta Sabou. Neon deliverable d5.4.1. neon methodology for building contextualized ontology networks. Technical Report D5.4.1, NeOn Project, 2008.

[SW07] J. Shore and S. Warden. *The Art of Agile Development*. O'Reilly, Sebastopol, CA, USA, 2007.

[VB08] Johanna Völker and Eva Blomqvist. D3.8.1 prototype for learning networked ontologies. Technical report, Institute AIFB, University of Karlsruhe, FEB 2008. NeOn Deliverable.

[VBBR09] J. Völker, E. Blomqvist, C. Baldassarre, and S. Rudolph. D3.8.2 Evaluation of Methods for Contextualized Learning of Networked Ontologies. Technical report, March 2009.

[VG06] Denny Vrandecic and Aldo Gangemi. Unit tests for ontologies. In *Proceedings of the 1st International Workshop on Ontology content and evaluation in Enterprise*, LNCS. Springer, 2006.

[VTAGS+08] B. Villazón-Terrazas, S. Angeletou, A. García-Silva, A. Gómez-Pérez, D. Maynard, M. C. Suárez-Figueroa, and W. Peters. NeOn Deliverable D2.2.2 Methods and Tools for Supporting Re-engineering. Technical report, NeOn, 2008.

[vWAB99] C. G. von Wangenheim, K.-D. Althoff, and R. M. Barcia. Goal-oriented and similarity-based retrieval of software engineering experienceware. In *SEKE*, volume 1756 of *Lecture Notes in Computer Science*, pages 118–141. Springer, 1999.