NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

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

D5.4.2. Revision and Extension of the NeOn Methodology for Building Contextualized Ontology Networks

Deliverable Co-ordinator: **Mari Carmen Suárez-Figueroa**

Deliverable Co-ordinating Institution: **UPM**

Other Authors: **Eva Blomqvist (CNR), Mathieu D'Aquin (OU), Mauricio Espinoza (UPM), Asunción Gómez-Pérez (UPM), Holger Lewen (UKARL), Igor Mozetic (JSI), Raúl Palma (UPM), Maria Poveda (UPM), Margherita Sini (FAO), Boris Villazón-Terrazas (UPM), Fouad Zablith (OU), Martin Dzbor (OU).**

This deliverable presents the second version of the NeOn methodology for building ontology networks. This version extends the previous one presented in D5.4.1 [80] providing methodological guidelines for: developing and creating ontology networks, reusing and re-engineering non-ontological resources, reusing ontology design patterns, modularizing ontologies, evaluating ontologies, evolving ontologies, and localizing ontologies.

Document Identifier:	NEON/2009/D5.4.2/v1.0	Date due:	February 28, 2009
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 28, 2009
Project start date:	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is a 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 28 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.upm.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SI-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 – 655 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 Koblenz-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. Martino 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 1 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:

UPM

CNR

FAO

JSI

OU

UKARL

Change Log

Version	Date	Amended by	Changes
0.00	11-06-2008	Asunción Gómez-Pérez	ToC and first version of introduction
0.03	15-07-2008	Mari Carmen Suárez-Figueroa	Second version of introduction. First version of chapter 2 about the NeOn Methodology
0.06	10-09-2008	Mari Carmen Suárez-Figueroa	Second version of chapter 2 about the NeOn Methodology
0.09	20-09-2008	Asunción Gómez-Pérez	Revision and update of current draft
0.10	5-10-2009	Asunción Gómez-Pérez Mari Carmen Suárez-Figueroa	Third version of chapter 2 about the NeOn Methodology
0.12	7-10-2008	Mathieu d'Aquin	First version of chapter about Modularization
0.14	10-10-2008	Igor Mozetic	First version of chapter about Evaluation
0.16	30-10-2008	Mari Carmen Suárez-Figueroa	Revision of chapter about Modularization
0.18	5-11-2008	Mari Carmen Suárez-Figueroa	Revision of chapter about Evaluation
0.20	3-12-2008	Igor Mozetic	Second version of chapter about Evaluation
0.22	20-12-2008	Mari Carmen Suárez-Figueroa	Revision of chapter about Evaluation
0.24	9-01-2009	Mathieu d'Aquin	Second version of chapter about Modularization
0.26	19-01-2009	Igor Mozetic	Third version of chapter about Evaluation
0.28	28-01-009	Mauricio Espinoza	First draft of the chapter about Ontology Localization
0.30	29-01-2009	Eva Blomqvist	First version of chapter about ODP reuse
0.32	29-01-2009	Asunción Gómez-Pérez	Revision and update of current draft
0.34	29-01-2009	Igor	Fourth version of chapter about Evaluation
0.36	29-01-2009	Raúl Palma Holger Lewen	First version of chapter about Evolution

0.38	30-01-2009	Mari Carmen Suárez-Figueroa	Revision of chapter about Evaluation and Evolution
0.40	2-02-2009	Igor Mozetic	Fifth version of chapter about Evaluation
0.42	2-02-2009	Mauricio Espinoza	Second version of of chapter about Ontology Localization
0.44	9-02-2009	Eva Blomqvist	Second version of chapter about ODP reuse Update of chapter about Evaluation
0.46	9-02-2009	Igor Mozetic	Sixth version of chapter about Evaluation
0.48	10-02-2009	Mari Carmen Suárez-Figueroa	Revision of the current version of all chapters and update of current draft
0.50	11-02-2009	Fouad Zablith	Update of chapter about Evolution
0.52	12-02-2009	Mathieu d'Aquin	Third version of chapter about Modularization
0.54	12-02-2009	Raúl Palma Holger Lewen	Third version of chapter about Evolution
0.56	16-02-2009	Eva Blomqvist	Third version of chapter about ODP reuse
0.58	16-02-2009	Boris Villazón-Terrazas	First version of Reuse and Re-engineering of Non-Ontological Resources
0.60	16-02-2009	Fouad Zablith	Update of chapter about Evolution
0.62	16-02-2009	Mari Carmen Suárez-Figueroa	Revision and update of current draft
0.64	17-02-2009	Mari Carmen Suárez-Figueroa María Poveda	Update on chapter 2
0.66	17-02-2009	Eva Blomqvist	Update of chapter about Evaluation
0.68	20-02-2009	Igor Mozetic	Seventh version of chapter about Evaluation
0.70	21-02-2009	Mathieu d'Aquin	Fourth version of chapter about Modularization
0.72	26-02-2009	Margherita Sini Mauricio Espinoza	Inclusion of ontology localization example from the FAO use case
0.74	4-03-2009	Boris Villazón-Terrazas	Second version of Reuse and Re-engineering of Non-Ontological Resources
0.76	6-03-2009	Igor Mozetic	Eigth version of chapter about Evaluation
0.78	6-03-2009	Mauricio Espinoza	Inclusion of ontology localization example using an Ontology Localization tool
0.80	9-03-2009	Mari Carmen Suárez-Figueroa	Revision of the current version of all chapters and update of current draft
0.82	11-03-2009	Mari Carmen Suárez-Figueroa	Update on introduction and conclusions
0.84	17-03-2009	Fouad Zablith	Update of chapter about Evolution
0.86	18-03-2009	Mari Carmen Suárez-Figueroa	Revision of the current version of the draft
0.9	20-03-2009	Mari Carmen Suárez-Figueroa	Draft version ready for Q.A revision
0.9.1	28-03-2009	Martin Dzbor	Review, cleanup, consistency checks
0.9.3	6-04-2009	Boris Villazón-Terrazas Igor Mozetic Mari Carmen Suárez-Figueroa	Updates based on Q.A. comments
0.9.5	7-04-2009	Eva Blomqvist Raúl Palma	Updates based on Q.A. comments

		Holger Lewen	
0.9.7	8-04-2009	Fouad Zablith	Updates based on Q.A. comments
1.0	12-04-2009	Mari Carmen Suárez-Figueroa	Final revision and creation of the final version

Executive Summary

Within WP5, we are creating the NeOn methodology for building ontology networks. The main principles that are guiding the construction of such methodology are:

1. The methodology should be general enough in the sense that it should help software developers and ontology practitioners to build network of ontologies with NeOn toolkit and with other widely used platforms such as Protégé or Top Braid Composer.
2. For each process or each activity, the methodology should define it precisely, state clearly its purpose, its inputs and outputs, the actors involved, when it is more convenient its execution. Furthermore, the methodology should provide prescriptive guidelines for each process or each activity, including proposals of methods, techniques and tools to be used for executing the process or activity.
3. To facilitate a promptly assimilation by software developers and ontology practitioners, we present the methodology in a manner non-oriented to researchers. We also include examples of how to use the methodology in different use cases.

The first version of the NeOn methodology for building ontology networks is included in D5.4.1 [80], presenting the following contributions:

- Analysis of how argumentation and collaboration dimensions are related to the different nine scenarios identified for collaboratively building network of ontologies.
- Methodological guidelines for carrying out the ontology (requirement) specification activity.
- Methodological guidelines for reusing and re-engineering non-ontological resources.
- Methodological guidelines for reusing ontological resources, focused on general or common ontologies, domain ontologies as a whole, and ontology statements.
- Methodological guidelines for reusing ontology design patterns by naive users.

The main goal of deliverable D5.4.2 is to present the second version of the NeOn methodology for building ontology networks, including the following contributions:

- Overview of the scenarios for building ontology networks identified in the NeOn Methodology.
- Summary of how to develop ontology networks.
- Explanation of the difference between single ontologies and ontology networks.
- Summary and update of the proposed methodological guidelines for non-ontological resource reuse and re-engineering processes, presented in D5.4.1 [80].
- Proposed methodological guidelines for carrying out the ontology design pattern reuse.
- Proposed methodological guidelines for the ontology modularization activity.
- Proposed methodological guidelines for the ontology (network) evaluation.

- Proposed methodological guidelines for the ontology evolution.
- Proposed methodological guidelines for carrying out the ontology localization activity.

Table of Contents

List of Tables

List of Figures

1. Introduction

The development of large-scale semantic applications in the near future will be characterized by using a very large number of ontologies embedded in ontology networks. Such ontologies will be developed collaboratively by distributed teams following a methodology for building ontology networks.

Thus, in the context of WP5, we are creating the *NeOn methodology* that support the collaborative aspects of ontology development, and the reuse and the dynamic evolution of networked ontologies in distributed environments, in which contextual information is introduced by developers (domain experts, ontology practitioners) at different stages of the ontology development process.

The NeOn methodology for building collaboratively ontology networks will include methodological guidelines and propose methods, techniques and tools for carrying out processes and activities, defined in the NeOn Glossary of Processes and Activities, during the ontology network life cycle. Some research results were presented in D5.4.1 [80] and other results are presented in this deliverable.

The first version of the NeOn methodology [80] proposed methodological guidelines for the following processes and activities: ontology (requirement) specification; reusing and re-engineering non-ontological resources; reusing ontological resources, focused on general or common ontologies, domain ontologies as a whole, and ontology statements; and reusing ontology design patterns by naive users.

The second version of the NeOn methodology included in this deliverable is a revision and an extension of the first version of the methodology, thus, other processes and activities should be covered in D5.4.2 by providing methodological guidelines for their execution.

Here, we continue with the idea of reusing as much as possible other ontologies, ontology modules, ontology statements and ontology design patterns as well as knowledge aware resources (non-ontological resources) such as thesauri, lexicons, DBs, UML diagrams and classification schemas built by others that already have some degree of consensus. Such a reuse allows speeding up the ontology (network) development process, saving time and money, and promoting the application of good practices.

Concretely, we focus on improving the proposed methodological guidelines for reusing and re-engineering non-ontological resources and for reusing ontology design patterns by naive users.

Additionally, modularization approach should be taken into account in the ontology network development at three different aspects: (1) designing modular ontologies; (2) modularizing existing ontologies, and (3) reusing ontology modules. In this deliverable we provide methodological guidelines for modularizing existing ontologies.

Other important aspect in the ontology (network) development is the ontology (network) evaluation. An ontology network is a complex structure of connected ontologies. These connections can have a form of meta-relations between ontologies (e.g., versions of ontologies) or mappings and alignments between pairs of individual ontology elements (concepts, properties, instances of related ontologies). Therefore, it is often practical to focus the evaluation on different aspects of the ontology network and perform the evaluation of the constituent parts of the ontology network first: (1) evaluation of constituent ontologies and (2) evaluation of mappings/alignments between pairs of ontologies. After the relevant individual parts of the network are evaluated, the evaluation results can be combined into the overall evaluation. Alternatively, the ontology network can be evaluated as a whole within particular application scenarios for which it was designed in the first place.

Ontology networks need to be kept up to date in order to reflect the changes that affect the life-cycle of such systems (e.g. changes in the underlying data sets, need for new functionalities, etc). Within the ontology networks lifetime, they undergo changes. They evolve for example to correct errors or adapt to new knowledge about the world, or changed circumstances. Moreover, during the ontology development process sometimes errors are not spotted and have to be corrected later, i.e. after initial deployment. However, as ontologies may depend on several others and may also be related to other elements (e.g. instances, mappings, applications, metadata, etc.), one has to be careful with making changes.

Ontology evolution is described as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes [27]. While it seems necessary to apply such an activity consistently for most ontology-based systems, it is often a time-consuming and knowledge intensive activity, as it requires a knowledge engineer to identify the need for change, perform appropriate changes on the base ontology and manage its various versions.

And finally, it is worth to mention that in the context of the emerging Semantic Web, a great effort has been done in the construction of ontologies. However, although access to top-quality ontologies (e.g., Galen4, CYC5, or AKT6) is in some cases free and unlimited for users all around the world, most of these ontologies are available only in English. Due to the language barrier, non-English users therefore often encounter problems when trying to access ontological knowledge in other languages. Moreover, more and more ontology-based systems are being built for multilingual applications (e.g., multilingual machine translation, multilingual information retrieval). This has increased the need for multilingual ontologies, and thus, for localizing ontologies.

1.1. Deliverable Main Goals and Contributions

The main goal of this deliverable is to present the second version of the NeOn Methodology for building networks of ontologies.

The principles that guide the construction of such a methodology, as presented in deliverable D5.4.1 [80] are:

1. The methodology should be general enough in the sense that it should help software developers and ontology practitioners to build networks of ontologies with the NeOn toolkit and with other widely used platforms such as Protégé or Top Braid Composer.
2. The methodology should define each process or activity precisely; state clearly its purpose, its inputs and outputs, the actors involved, when its execution is more convenient. Additionally, the methodology should propose methods, techniques and tools to be used for executing each process or activity.
3. To facilitate a prompt assimilation by software developers and ontology practitioners, we present the methodology in a clear and prescriptive way, including examples on how to use the methodology in different use cases.

The scope of this deliverable is limited to provide methodological guidelines for the following processes and activities:

- ❑ *Non-Ontological Resource Reuse and Re-engineering.*

  Currently, ontology engineers and software practitioners are realizing the benefits of "not reinventing the wheel" at each ontology development, thus looking towards new methods, techniques, and tools for reusing and re-engineering knowledge.

  In this sense, preliminary methodological guidelines for reusing and re-engineering non-ontological resources were included in deliverable D5.4.1 [80]. In deliverable D5.4.2, we summarize such guidelines, including improvements with respect to the preliminary ones and referring to the re-engineering patterns used in the guidelines, which are described in deliverable D2.2.2 [2].

❑ *Ontology Design Pattern Reuse.*

Ontology design patterns (OPs) is a way of encoding best practices, based on experiences and knowledge of 'good' solutions. Generally patterns (e.g. in software engineering) are perceived as having three kinds of benefits [53]: (1) reuse benefits, (2) guidance benefits, and (3) communication benefits. All these benefits can intuitively also be found in ontology engineering, although it is not until recently that experiments have been performed that empirically show such benefits [30].

Even when such benefits have been established, there is still need to support pattern usage in different ways with the goal of minimising the unnecessary overhead introduced and maximising benefits. Due to this fact there is need for guidelines on how to reuse patterns, as well as tool supporting the actual practical usage.

For this reason, in D5.4.1 [80] methodological guidelines for reusing patterns by naïve users were presented. In this deliverable, we proposed general methodological guidelines for reusing ontology design patterns, and taking into account that there are different kinds of OPs [67] (correspondence, presentation, reasoning, lexico-syntactic, structural, and content) we provide here specific guidelines for reusing content OPs.

❑ *Ontology Modularization.*

As already mentioned, modularization approach should be taking into account in the ontology network development. This approach is related with the following three aspects:

- Designing modular ontologies, that is, building ontologies from independent and self contained components.
- Modularizing existing ontologies, which involves different possibilities: (1) extracting modules from ontologies, (2) decomposing ontologies into modules, and (3) hiding modules from ontologies.
- Reusing ontology modules.

Methodological guidelines for the three aforementioned aspects were not available until now. For this reason, and taking into account the importance of modularizing ontologies, which is motivated by several different scenarios within the NeOn case studies and technical workpackages, as described in [21], in this deliverable we focus on such an activity and provide methodological guidelines for modularizing existing ontologies, that is, for creating modules as sub-components of larger ontologies.

Such methodological guidelines are supported by several operators included in the NeOn toolkit for modularization, as described in [20]. The proposed operators are designed to be generic, in order to be useful in the majority of the modularization scenarios. As such, they have to be used in an interactive process, where the user provides the relevant parameters and input.

❑ *Ontology (Network) Evaluation.*

Ontology evaluation is a topic that has been treated by researchers for more than a decade; however, it is not until recently that general frameworks and categories for ontology evaluation methods have emerged. Still, numerous techniques exist for all different kinds of ontology evaluation methods. The main challenge is to select the methods that are suitable for the case at hand and the data and resources available.

Additionally, an ontology network is a complex structure of connected ontologies. Ontology networks should be evaluated with respect to different aspects, and such an evaluation should be performed first on the constituent parts of the ontology network and then combining the results into the overall evaluation.

In this deliverable we provide general methodological guidelines for the ontology (network) evaluation activity, whose goal is to evaluate and compare the ontology network within a useful frame of reference and using appropriate evaluation criteria.

- *Ontology Evolution.*

  Ontologies are fundamental building blocks of the Semantic Web and are often used as the knowledge backbones of advanced information systems. As such, they need to be kept up to date in order to reflect the changes that affect the life-cycle of such systems (e.g. changes in the underlying data sets, need for new functionalities, etc).

  While knowledge engineers agree that it is necessary to apply the ontology evolution activity consistently in the ontology-based systems, it is often a time-consuming and knowledge intensive activity, as it requires a knowledge engineer to identify the need for change, perform appropriate changes on the base ontology and manage its various versions. Moreover, there was no existing approach available until now covering a complete cycle of ontology evolution, ranging from integrating new knowledge to managing changes.

  For these reasons, in this deliverable we provide two types of guidelines: (a) general methodological guidelines for facilitating the modification and evolution of an ontology, and (b) methodological guidelines for supporting ontology engineers and domain experts in exploiting tools to facilitate the evolution of their ontologies.

- *Ontology Localization.*

  In the context of the emerging Semantic Web, a great effort has been done in the construction of ontologies. Although access to top-quality ontologies (e.g., Galen4, CYC5, or AKT6) is in many cases free and unlimited for users all around the world, most of these ontologies are available only in English. Due to the language barrier, non-English users therefore often encounter problems when trying to access ontological knowledge in other languages. Moreover, more and more ontology-based systems are being built for multilingual applications (e.g., multilingual machine translation, multilingual information retrieval).

  For these reasons the need for multilingual ontologies has increased. Additionally, multilingual ontologies are very time consuming and expensive to build. Let us take the example of the well-known EuroWordNet (EWN) a general purpose multilingual lexicon for eight European languages. Its development involved 11 academic and commercial institutions and took three years to complete. Therefore, it is attractive to consider new ways of building multilingual ontologies. One stream of research that is emerging as alternative to build a multilingual ontology is to localize an ontology. Since there is no universal way to localize an ontology and the choice of particular methods and tools to build an ontology localization system should be guided by the final purpose of the localized ontology and by the techniques deemed to be more efficient for each task of the activity, it is necessary to propose methodological guidelines to help ontology practitioners in the localization activity. Such methodological guidelines are included in this deliverable.

1.2. Deliverable Structure

The deliverable is structured as follows:

- Chapter 2 presents: (1) an overview of the different scenarios for building ontology networks identified in the NeOn Methodology, (2) a summary of how to develop ontology networks, and (3) an explanation the difference between single ontologies and ontology networks.

- Chapter 3 summarizes the proposed methodological guidelines for non-ontological resource reuse and re-engineering processes, presented in D5.4.1 [80].
- Chapter 4 presents the proposed methodological guidelines for carrying out the ontology design pattern reuse.
- Chapter 5 describes the proposed methodological guidelines for the ontology modularization activity.
- Chapter 6 presents the proposed methodological guidelines for the ontology (network) evaluation.
- Chapter 7 describes the proposed methodological guidelines for the ontology evolution.
- Chapter 8 presents the proposed methodological guidelines for carrying out the ontology localization activity.
- Chapter 9 presents the conclusions and future work.

1.3. Relation with other WP5 Deliverables and other Activities in NeOn

The relation between this deliverable and the rest of the work done in WPs in the NeOn project is briefly described below:

- Methodological guidelines for a subset of process and activities defined in the NeOn Glossary [1] are included in this deliverable.
- For some of the processes or activities described in this deliverable and for others in the NeOn Glossary we planned and executed experiments, which are described in D5.6.1 and in D5.6.2.
- The NeOn Methodology as a whole in combination with the NeOn Toolkit will be tested. The preliminary plan for such test is included in D5.7.1, and the final evaluation will be included in D5.7.2.
- The methodological guidelines for the non-ontological resource re-engineering process used re-engineering patterns that are included in D2.2.2.

2. NeOn Methodology for Building Ontology Networks

As we mentioned before, the 1990s and the first years of this new millennium have witnessed the growing interest of many practitioners in approaches that support the creation and management as well as the population of single ontologies built from scratch. There are some methodological approaches (e.g., METHONTOLOGY, On-To-Knowledge, and DILIGENT) that help develop ontologies from scratch. All these approaches have provoked a step forward by having transformed the art of constructing single ontologies into an engineering activity.

The development of ontologies in different international and national projects has revealed that there are different alternative ways or possibilities to build ontologies [80]. Thus, it is not premature to affirm that a new ontology development paradigm is starting, whose emphasis is on the reuse and possible subsequent re-engineering of knowledge resources, the collaborative and argumentative ontology development, and the building of ontology networks, as opposed to custom-building new ontologies from scratch.

Taking into account such a paradigm, we are creating the *NeOn Methodology for building ontology networks*. The first version of the methodology was published in D5.4.1 [80] and provided the following:

- ❑ Methodological guidelines for carrying out the ontology specification activity.
- ❑ Methodological guidelines for reusing and re-engineering non-ontological resources.
- ❑ Methodological guidelines for reusing ontological resources, focused on reusing general or common ontologies, reusing domain ontologies as a whole, and reusing ontology statements.
- ❑ Methodological guidelines for reusing ontology design patterns by naive users.

In the second version of the NeOn methodology that is included in this deliverable, we present the following methodological guidelines to extend D5.4.1:

- ❑ Summary and update of the methodological guidelines for reusing and re-engineering non-ontological resources, referring to the re-engineering patterns used in the guidelines, which are described in deliverable D2.2.2 [2].
- ❑ Methodological guidelines for reusing ontology design patterns in general; and guidelines for reusing content ontology design patterns.
- ❑ Methodological guidelines for modularizing existing ontologies.
- ❑ Methodological guidelines for the ontology (network) evaluation activity.
- ❑ Methodological guidelines for the ontology evolution activity and for supporting ontology engineers and domain experts in exploiting tools to facilitate the evolution of their ontologies.
- ❑ Methodological guidelines for localizing ontologies.

This chapter briefly includes:

- ❑ An overview of the different scenarios for building ontologies and ontology networks.
- ❑ A general description of how ontology networks should be developed.
- ❑ An explanation (and illustration) of when ontologies become ontology networks, and how to create and ontology network using single ontologies.

2.1. Scenarios for Building Ontology Networks

Based on the analysis of the three NeOn use cases, on the different studies carried out to revise the state of the art on ontology development, and on the building of ontologies in different international and national projects, we have detected that there are alternative ways or possibilities to build ontologies and ontology networks. These ways can be seen as different scenarios in the NeOn methodology for building ontology networks.

As already mentioned in previous WP5 deliverables, the scenarios we proposed within the NeOn methodology are flexible because a combination among them is allowed, unlike in the scenarios for building ontologies presented in the most well-known ontology engineering methodologies, which are very rigid.

Here we provide a summary of the 9 scenarios identified in the context of the NeOn methodology for building ontology networks.

Figure 1 presents the set of 9 scenarios for building ontologies and ontology networks. Directed arrows with numbered circles associated represent the different scenarios. Each scenario is decomposed into different processes or activities that are represented with colour circles or with rounded boxes. Such processes and activities are defined in the NeOn Glossary of Processes and Activities [78]. The figure also shows (as dotted boxes) existing knowledge resources to be reused; and possible outputs (implemented ontology networks and alignments) that result from the execution of some of the presented scenarios.

The most common scenarios[1] that may arise during the ontology development are the following:

- *Scenario 1*: Building ontology networks from specification to implementation.
- *Scenario 2*: Building ontology networks by reusing and re-engineering non-ontological resources.
- *Scenario 3*: Building ontology networks by reusing ontological resources.
- *Scenario 4*: Building ontology networks by reusing and re-engineering ontological resources.
- *Scenario 5*: Building ontology networks by reusing and merging ontological resources.
- *Scenario 6*: Building ontology networks by reusing, merging and re-engineering ontological resources.
- *Scenario 7*: Building ontology networks by reusing ontology design patterns.
- *Scenario 8*: Building ontology networks by restructuring ontological resources.
- *Scenario 9*: Building ontology networks by localizing ontological resources.

The activities of knowledge acquisition and elicitation, documentation, configuration management, evaluation and assessment should be carried out during the whole ontology development.

From this set of scenarios, we can say that scenario 1 is the most typical for building ontology networks without reusing existing knowledge resources. However, as already mentioned, more and more ontology developers build ontology networks by means of reusing existing knowledge aware resources (ontological and non-ontological). For this reason, the NeOn methodology differentiates scenarios involving reuse of ontological resources from those involving reuse and re-engineering of non-ontological resources.

It is worth mentioning that these scenarios can be combined in different ways, and that any combination of scenarios should include scenario 1 because this scenario is made up of the core activities that have to be performed in any ontology development. Indeed, as Figure 1 shows, the results of any other scenario should be integrated in the corresponding activity of scenario 1.

[1] The scenarios are valid for both building ontologies and ontology networks.

Although we think this set of scenarios covers the most plausible ways for building ontology networks, it can not be considered exhaustive.

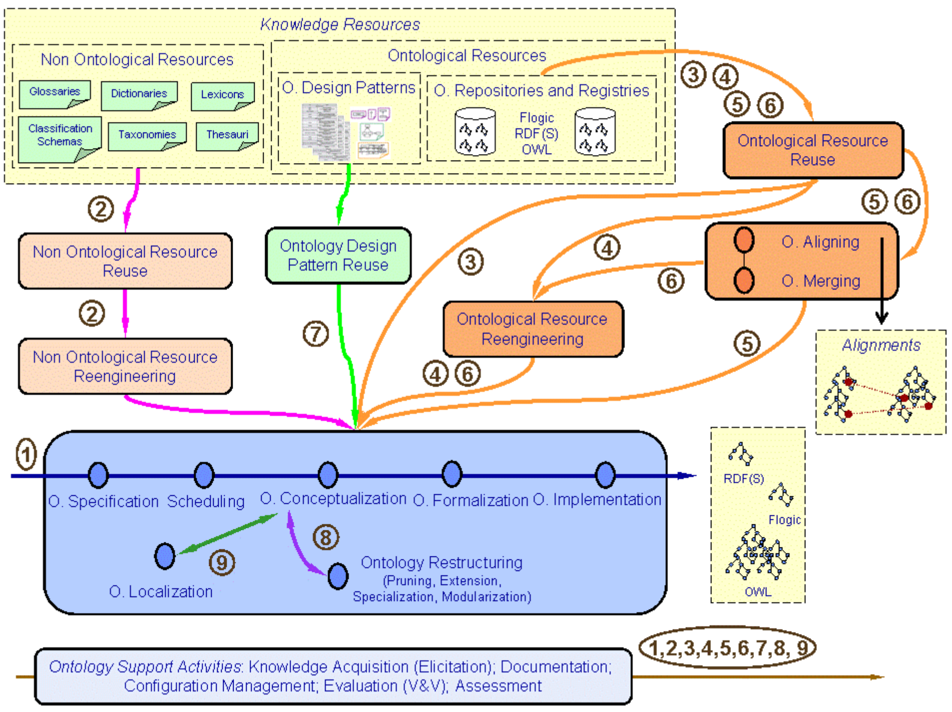

Knowledge Resources
Non Ontological Resources
Glossaries
Dictionaries
Lexicons
Classification Schemas
Taxonomies
Thesauri
Ontological Resources
O. Design Patterns
O. Repositories and Registries
Flogic
RDF(S)
OWL
Ontological Resource Reuse
Non Ontological Resource Reuse
Ontology Design Pattern Reuse
O. Aligning
O. Merging
Ontological Resource Reengineering
Non Ontological Resource Reengineering
Alignments
O. Specification
Scheduling
O. Conceptualization
O. Formalization
O. Implementation
O. Localization
Ontology Restructuring
(Pruning, Extension, Specialization, Modularization)
RDF(S)
Flogic
OWL
Ontology Support Activities: Knowledge Acquisition (Elicitation); Documentation; Configuration Management; Evaluation (V&V); Assessment
1,2,3,4,5,6,7,8, 9

Figure 1. Scenarios for Building Ontology Networks

The first version of the NeOn methodology [80] includes methodological guidelines for processes and activities of scenarios 1, 2, 3 and 7.

These second version of the methodology includes methodological guidelines for processes and activities involved in scenarios 2, 7, 8 and 9, and for transversal activities (evaluation and evolution). The processes and activities considered in this deliverable are: reuse and re-engineering of non-ontological resources, reuse of ontology design patterns, ontology modularization, ontology (network) evaluation, ontology evolution, and ontology localization.

In the third version of the methodology we will cover the remaining scenarios.

2.2. Building Ontology Networks in a Nutshell

When software developers and ontology practitioners consider using ontologies for solving a particular problem, a provisional work team (ideally involving ontology engineers, software developers, domain experts and final users) should be established. Such a team will be involved at least in the following pre-development and support activities: environment and feasibility study, knowledge acquisition, ontology (requirement) specification, and scheduling.

After the provisional team has been established, it usually carries out an environment and feasibility study. This allow them to decide whether ontologies should be developed or not for the specific problem. If the conclusion is positive, they have to decide if, for their problem, it is better to build a single ontology, a set of interconnected single ontologies, or an ontology network, where the differences between these choices are explained in Section 2.3.

Then, simultaneously with the knowledge acquisition activities, the provisional work team should specify the requirements that the ontology network should fulfil, by means of the ontology (requirement) specification activity. The objective of the ontology (requirement) specification activity is as stated in [80] to output a document, the ontology requirements specification document

(ORSD), that includes the purpose, the scope and the level of formality of the ontology network, target group and intended uses of the ontology network, and the set of requirements the ontology network should fulfil.

After the ontology (requirement) specification activity, it is advisable to carry out a search for knowledge-aware resources using as input the terms included in the ORSD. Such a search allows the provisional work team to know which type of resources is available for a possible reuse during the ontology (network) development.

The provisional work team should perform the scheduling activity, using the ORSD and the results of such a quick search. During the scheduling activity, the team establishes the ontology (network) life cycle and the human resources for the ontology project. Such resources could include or not people form the provisional team, depending on organizational issues.

After this, human resources assigned to the ontology project should follow up the established schedule, that is, the concrete plan for the ontology network development.

2.3. From Ontologies to Ontology Networks

Based on our experience, we identified three different possibilities when building ontologies: (1) building single ontologies; (2) building sets of interconnected single ontologies; and (3) building ontology networks.

Next, we describe the three possibilities:

- A *single ontology* is an ontology that has not got any type of relationship (domain dependent or independent) with other ontologies.
- A *set of interconnected single ontologies* includes a set of ontologies that have domain dependent relationships among them.
- An *ontology network or a network of ontologies* is a collection of ontologies together through a variety of formal relationships such as mapping, modularization, version, and dependency relationships [37].

To clarify what is the difference between a single ontology, a set of interconnected single ontologies, and an ontology network, we provide here some examples.

- An isolated ontology O_1 is a single ontology, as it is shown in Figure 2 (a).
- *N* single ontologies related to each other by means of ad-hoc relations between concepts included in such ontologies are considered as a set of interconnected single ontologies, as in Figure 2 (b) shows.
- If ontology (O_2) is developed as a new version of O_1 and the explicit metarelation "*priorVersionOf*" between O_1 and O_2 is established, then a network of ontologies has been created (shown Figure 2 (c)).
- Figure 2 (d) presents the ontology network associated with a set of interconnected single ontologies presented in Figure 2 (b), in which the meta-relationships among the different ontologies involved have been explicitly expressed.

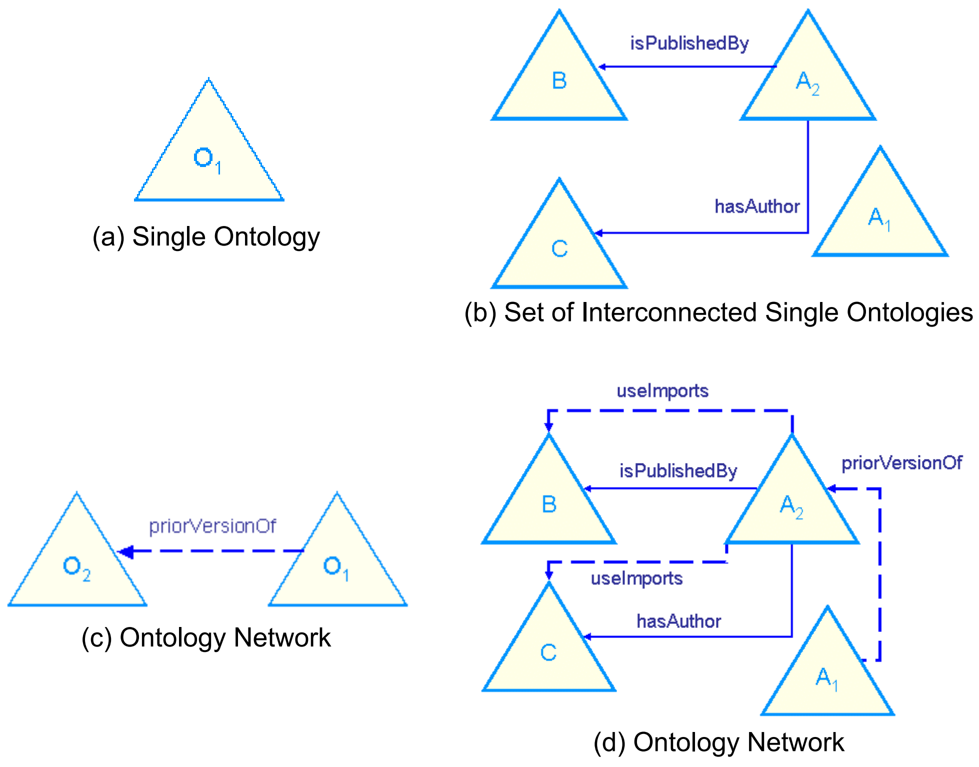


Figure 2. Simple Graphical Examples of Single Ontologies, Set of Interconnected Single Ontologies, and Ontology Networks

2.3.1. How to create the network

An ontology network should be developed if there is a requirement or it is advisable to express: (a) meta-relationships between the ontology to be developed and other existing ontologies available, or (b) meta-relationships between the ontology to be developed and its components. Examples of these meta-relationships are:

- *priorVersionOf*: if the ontology to be developed is a new version of an existing one.
- *useImports*: if the ontology is importing any other ontology due to the fact that it consists of different knowledge domains.
- *extendingBy*: if the ontology is extending an existing one.
- *composedbyModules*: if the ontology to be developed is composed of a number of modules.
- *haveMapping*: if some ontology components have mappings with other existing ontologies.

Thus, in summary, if software developers and ontology practitioners explicitly define meta-relationships such as mapping, modularization, version, and dependency, between a set of ontologies and/or between an ontology and its components, then de-facto, they are developing an ontology network. Meta-relationships, such as "*priorVersionOf*", "*useImports*", "*isIncompatibleWith*", are formally explained in D1.1.1 [39].

In this case, the ontology to be developed is in constant relation with other members of the ontology network, and this permits a continuous knowledge sharing and an easy enrichment of the network. Furthermore, ontology networks favour the knowledge growth in Internet, and thus, knowledge sharing and spreading.

Based on our experience building ontology networks in different domains, we realized that there are different ways of creating an ontology network having a set of single ontologies.

During our developments we have identified the following two situations based on the relationship existing among the ontologies that will be part of the network:

- The existing relation is an ad-hoc or domain-specific, or is a 'subclass-of' relation among concepts represented in different ontologies.
- Concepts are repeatedly used or equivalent concepts are used in different ontologies.

In this deliverable we provide preliminary and general methodological guidelines for the aforementioned situations to help ontology developers in the connection of ontologies to create an ontology network. It is important to mention that such situations cannot be considered exhaustive, and that other situations can appear during the building of other ontology networks.

- Ad-hoc or 'subclass-of' relation between ontologies. In this situation there is a relation between concepts belonging to different ontologies, and it is needed to decide in which ontology the relation should be defined. Based on our experience we proposed the following possibilities:
    - If one of the ontologies represents a sub-domain of another ontology, then ontology developers should identify one ontology as more specific and the other as more general, import the first one in the second ontology, and finally establish the ad-hoc relation between the concepts.
    - If the relation between the concepts is 'subclass-of', then ontology developers should identify one ontology as more specific and the other as more general, import the first one in the second ontology, and finally establish the 'subclass-of' relation between the concepts.
    - In other situations, we proposed to build a new ontology that imports the ontologies among which there is the relation and to create the relation in such a new ontology.
- Repeated or equivalent concepts. In this situation equivalent relations should be defined among all the concepts that are equal in the set of ontologies. The number of equivalent relations to be defined could be minimized by taking into account the following two cases:
    - If the equivalent relation exists between concepts of ontologies that are being reused, then we propose that ontology developers import the reused ontologies in another one, and establish the equivalent relation in this last ontology.
    - If the equivalent relation exits between a concept in an ontology to be reused and a concept in the ontology being developed, then we propose that ontology developers import as soon as possible the ontology to be reused in order to avoid the creation of the concept in the ontology being built and thus the establishment of the equivalent relation.

3. Non-Ontological Resource Reuse and Re-engineering

Non-Ontological Resource Reuse refers to taking available non-ontological resources[2] (e.g., databases, controlled vocabularies, etc.) for the development of ontologies. *Non-Ontological Resource Reuse* is defined in [80] as the process of choosing the most suitable non-ontological resources for the development of ontologies.

Non-Ontological Resource Re-engineering is defined in [81, 78] as the process of retrieving and transforming an existing non-ontological resource (e.g., databases, controlled vocabularies, etc.) into an ontology.

Non-ontological resource reuse and re-engineering processes belong to the development scenario called *Building Ontology Networks by Reusing and Re-engineering Non-Ontological Resources* identified in the NeOn Methodology [80]. In this scenario, software developers and ontology practitioners develop the ontology network by means of reusing and re-engineering existing non-ontological resources.

The NeOn approach to carry out non-ontological resource reuse and re-engineering processes has been already described in D5.4.1 [80]. Software developers and ontology practitioners should accomplish first the *non-ontological resource reuse process* with the goal of choosing the most suitable non-ontological resource to be used for building ontologies. If they decide that one or more resources are useful for the development, then the *non-ontological resource re-engineering process* should be carried out to transform the selected non-ontological resources into ontologies.

In this chapter we present a brief summary of the methodological guidelines for the non-ontological resource reuse process described in [80]. Additionally, we also include a summary of the methodological guidelines for the non-ontological resource re-engineering process that was also presented in [80] and for which we have created 10 new re-engineering patterns. Those patterns were originally proposed and included in D2.2.2 [2].

3.1. Summary of the Guidelines for Non-Ontological Resource Reuse

As we mentioned before, the goal of the Non-Ontological Resource Reuse process is to choose the most suitable non-ontological resource to be used for building ontologies. Domain experts, software developers and ontology practitioners carry out this process taking as input the ontology requirements specification document (ORSD) to find the most suitable non-ontological resources for the development of ontologies. The output of the process is a set of non-ontological resources that to some extend covers the expected domain. It is worth to mention that we have included new criteria for assessing the quality of the non-ontological resource. These criteria are described within Activity 2.

In the framework of the NeOn methodology for building ontology networks, we propose the non-ontological resource reuse process filling card, presented in Table 1, which includes the definition, goal, input, output, who carries out the process and when the process should be carried out.

[2] Non-Ontological Resources (NORs), which were defined in [80], are knowledge-aware resources whose semantics have not been formalized yet by an ontology.

Non-Ontological Resource Reuse	
Definition	
Non-Ontological Resource Reuse refers to the process of choosing the most suitable non-ontological resources for the development of ontologies.	
Goal	
To choose the most suitable non ontological resources for building ontologies.	
Input	*Output*
The ontology requirements specification document (ORSD).	A set of non-ontological resources that to some extend covers the expected domain.
Who	
Domain experts, software developers and ontology practitioners.	
When	
After the ontology specification activity and before the non-ontological resource re-engineering process.	

Table 1. Non-Ontological Resource Reuse Filling Card

The activities for carrying out the non-ontological resource reuse process are briefly explained in the following; details are available in [80]:

Activity 1. Search non-ontological resources.

The goal of the activity is to search non-ontological resources from highly reliable Web sites, domain-related sites and resources within organizations. Domain experts, software developers and ontology practitioners carry out this activity taking as input the ontology requirements specification document (ORSD). They use those terms that have a highest frequency in the ORSD to search for candidate non-ontological resources that cover the desired terminology. The activity output is a set of candidate non-ontological resources that might present any of the identified typologies described in D5.4.1 [80].

Activity 2. Assess the set of candidate non-ontological resources.

The goal of this activity is to assess the set of candidate non-ontological resources. Domain experts, software developers and ontology practitioners carry out this activity taking as input the set of candidate non-ontological resources. We propose to take into account the following criteria: *coverage* and *precision*, which are measurable criteria; and consensus, which is a subjective criterion. The activity output is an assessment table that shows the evaluation criteria for every non-ontological resource.

Within this activity we propose in this deliverable to include the quality of the non-ontological resource. Quality attributes include:

- Well documentation of the non-ontological resources.
- Lack of anomalies of the non-ontological resource, such redundancies or inconsistencies.
- Reliability of the non-ontological resource, it means analysing whether we can trust in the resource or not.

Activity 3. Select the most appropriate non-ontological resources.

The goal of this activity is to select the most appropriate non-ontological resources. Domain experts, software developers and ontology practitioners carry out this activity taking as input the non-ontological resource assessment table. The selection is performed manually and looking for resources with: consensus, high value of coverage, and high value of precision. The activity output is a ranked list of non-ontological resources that to some extend covers the expected domain.

3.2. Summary of the Guidelines for Non-Ontological Resource Re-engineering

As we mentioned before, the goal of the Non-Ontological Resource Re-engineering process is to transform a non-ontological resource into an ontology. The output of the process is an ontology.

In the framework of the NeOn methodology for building ontology networks, we propose the non-ontological resource re-engineering process filling card, presented in Table 2, which includes the definition, goal, input, output, who carries out the process and when the process should be carried out.

In a nutshell, the method for re-engineering non-ontological resources proposed in the NeOn methodology [80] considers as input a pool of non-ontological resources and a set of patterns for re-engineering such resources. The proposed patterns provide general solutions to the problem of transforming non-ontological resources into ontologies.

The Non-Ontological Resource Re-engineering process can be divided into the following activities [77]:

- *Non-Ontological Resource Reverse Engineering* is defined as the activity of analyzing a non-ontological resource to identify its underlying components and creating a representation of the resource at higher levels of abstraction.
- *Non-Ontological Resource Transformation* is defined as the activity of generating an ontological model, at different levels of abstraction, from the non-ontological resource. It is worth to mention that we have included a new task within this activity: *manual refinement of the resultant transformation*.
- *Ontology Forward Engineering* is defined in D5.3.1 [81] as the activity of generating a new implementation of the ontology, on the basis of a new conceptual model.

Non-Ontological Resource Re-engineering	
Definition	
Non-Ontological Resource Re-engineering refers to the process of taking an existing non-ontological resource and transforms it into an ontology.	
Goal	
Create an ontology from a non-ontological resource.	
Input	*Output*
One or more non-ontological resources selected by the reuse process.	An ontology.
Who	
Domain experts, software developers and ontology practitioners.	
When	
After the non-ontological resource reuse process and before the conceptualization activity.	

Table 2. Non-Ontological Resource Re-engineering Filling Card

The activities for carrying out the non-ontological resource re-engineering process are summarized in the following:

Activity 1. Non-ontological Resource Reverse Engineering.

The goal of this activity is to analyze a NOR to identify its underlying components and create representations of the resource at the different levels of abstraction (design, requirements and conceptual). Since NORs can be implemented as XML files, databases or spreadsheets among others, we can consider them as software resources, and therefore, we use the software abstraction levels (implementation, design, requirements and conceptual). Here the requirements and the essential design, structure and content of the NOR must be recaptured.

Activity 2. Non-ontological Resource Transformation.

The goal of this activity is to generate a conceptual model from the NOR. We propose the use of Patterns for Re-engineering Non-Ontological Resources (PR-NOR) to guide the transformation process.

To perform such transformation, the following characteristics have to be identified: (1) the non-ontological resource type; (2) the internal data model of the non-ontological resource; and (3) the semantics of the relations among the non-ontological resource entities.

The semantics of the relations among the non-ontological resource entities can be a) *subClassOf*, b) a *partOf* relation or c) a mix of *subClassOf* and *ad-hoc* relations.

After the identification of the aforementioned characteristics, a pattern for re-engineering non-ontological resources has to be searched according to the type of non-ontological resource, the internal data model, and the semantics of the relations among the non-ontological resource entities. The pattern search is carried out on the set of patterns for re-engineering non-ontological resources described in deliverable D2.2.2 [2]. This set of patterns will be available on the ontology design patterns portal[3].

Finally, the selected re-engineering pattern has to be applied to transform the non-ontological resource into a conceptual model.

Next, the selected re-engineering pattern has to be applied to transform the non-ontological resource into a conceptual model. Here, we include the task of *manual refinement of the resultant transformation*, within this task, software developers and ontology practitioners with domain experts' support can perform a disambiguation of some relationships.

Activity 3. Ontology Forward Engineering.

The goal of this activity is to output a new implementation of the ontology on the basis of the new conceptual model. We use the ontology levels of abstraction to depict this activity because they are directly related to the ontology development process.

3.3. Future Work

In this chapter we have presented a brief summary of the methodological guidelines for carrying out the non-ontological resource reuse and re-engineering processes, emphasizing the updates with respect to the guidelines in [80].

Further work related with these processes includes:

- Refinement of the methods and the patterns for re-engineering non-ontological resources.
- Inclusion of guidelines on how to generate Linked Data, following the Linking Open Data[4] recommendations.
- Implementation of software libraries that read the data from the resource implementation and automatically generate the corresponding classes, attributes, and relations of the new ontology following the suggestions given by the PR-NOR.
- Evaluation of the methodological guidelines and the patterns for re-engineering non-ontological resources (PR-NORs).

[3] http://ontologydesignpatterns.org/

[4] http://linkeddata.org/

4. Ontology Design Pattern Reuse

Ontology Design Pattern (OP) Reuse is defined in [77] as the activity of using available ontology design patterns in the solution of different modeling problems during the development of new ontologies. Ontology design patterns (OPs) is an emerging technique for the reuse of encoded experience and best practises. Ontology Design Pattern Reuse is part of the overall NeOn methodology, as described in scenario 7 in NeOn deliverables D5.4.1 [80], and D5.3.2 [77].

Design patterns were already proposed in NeOn deliverables D5.1.1 [82] and D2.5.1 [67] as a set of guidelines and reusable components for ontology design. In deliverable D5.4.1 [80] already guidelines were proposed for letting naive users exploit some types of logical OPs. In this deliverable the focus is on general users, proposing general methodological guidelines for OP reuse, as well as specific guidelines for reusing content OPs that is more suited for developers with some prior modelling experience.

First we 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; and we then propose the NeOn methodological guidelines for carrying out the activity.

4.1. State of the Art and Related Work in NeOn

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 D2.1.1 [14]), ontology design patterns (OPs) 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. As mentioned above the NeOn methodology for building ontology networks described in D5.4.1 [80] incorporates a scenario where OPs are the basis for ontology design (scenario 7), starting from ontology requirements and resulting in OPs integrated in the ontology network to be built.

Even though the use of patterns is widespread, also 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 pattern-based methods are present primarily on the logical 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. Such methods have been proposed in several NeOn deliverables, e.g. D3.8.1 [87]. For this deliverable, on the other hand, the focus is on methodological support, rather than tool support, i.e. guidelines for using patterns rather than tools and algorithms that automatically exploit patterns.

As already mentioned, within this focus, there is even less prior work within the ontology engineering field, whereas NeOn is at the forefront of this development. Some methodological operations were however already presented in D2.5.1 [67]. The operations are mainly used for content patterns, but give an idea of what could be needed also for other pattern types, in terms of how patterns in general may be reused. The operations were the following:

- Covering is related to the requirements that are to be solved by a model, or a pattern. A pattern covers a set of requirements, if it is expressive enough to store the necessary knowledge to answer the full set of competency questions (CQs) representing the requirements.

- Cloning involves making a duplicate of an ontology element, in some cases including all axioms defining the element or in some cases only making a partial clone.
- Composition is the way of combining two or more patterns, where the result in case of content patterns is the union of the axioms of all the patterns together with any additional axioms used to link the elements of the different patterns.
- Specialization and generalization are relations among patterns, as well as among elements. Specialization is also a way of reusing a content pattern, when the elements are specialized and thereby form a new ontology tailored to some specific requirements.
- Expansion is when a pattern is extended with additional elements or axioms.
- Importing is a basic operator for content pattern reuse (since already available in OWL) and is the standard way of reusing such OPs.

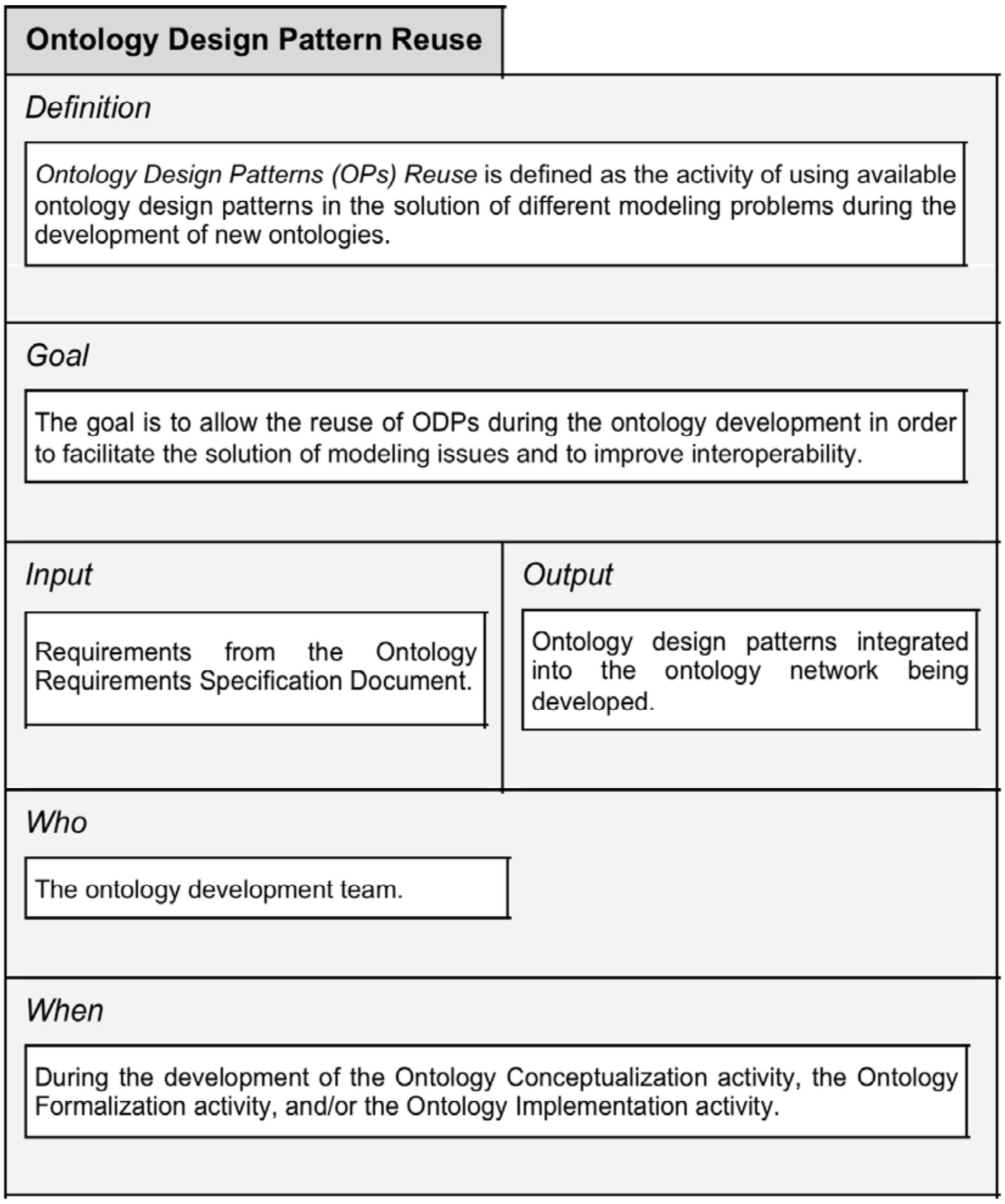
For more details, and formal definitions of the operators the reader is referred to D2.5.1 [67].

Additionally, in D2.5.1 [67] there is a brief and informal description of how content OPs can be reused, but this description is focussed only on the matching between current requirements and the intent of the OPs. In general, the process proposed within the NeOn methodology for reuse of OPs have so far been to somehow match the patterns (the solution space) to the current requirements (the problem space), select a set of patterns, and reuse them (possibly through importing, specializing and expanding them) to construct an ontology (or ontology network). A divide-and-conquer paradigm was inherent in these initial ideas, since the patterns are small solutions that solve small parts of the overall problem. Still, no more specific guidelines were proposed so far but are instead the topic of this deliverable.

4.2. Proposed Guidelines for Ontology Design Pattern Reuse

As was mentioned before, the goal of ontology design pattern reuse is to facilitate the solution of modeling issues and to improve interoperability through using well-proven solutions and best practices, in the form of patterns.

In the framework of the NeOn methodology we propose the ontology design pattern reuse filling card, presented in Table 3, for the ontology design pattern reuse activity. It is worth to mention that the filling card for the ontology design pattern reuse presented in deliverable D5.4.1 [80] has been modified to obtain the new one presented in Table 3. The main difference is that we redefined the activity to focus on the solution of modelling problems, while previously it also included pattern-based matching. This is in line with the latest revision of the NeOn glossary of terms.



Ontology Design Pattern Reuse

Definition

Ontology Design Patterns (OPs) Reuse is defined as the activity of using available ontology design patterns in the solution of different modeling problems during the development of new ontologies.

Goal

The goal is to allow the reuse of ODPs during the ontology development in order to facilitate the solution of modeling issues and to improve interoperability.

Input	*Output*
Requirements from the Ontology Requirements Specification Document.	Ontology design patterns integrated into the ontology network being developed.

Who

The ontology development team.

When

During the development of the Ontology Conceptualization activity, the Ontology Formalization activity, and/or the Ontology Implementation activity.

Table 3. Ontology Design Pattern Reuse Filling Card

The tasks for carrying out the ontology design patterns reuse activity can be seen in Figure 3. This is an updated version of what was provided in NeOn Deliverable D5.3.2 [77]. The activity starts by identifying the set of requirements to be addressed; possibly not the complete set of requirements in the ORSD will be addressed using OPs, and a set of patterns (catalogues of patterns) that are available for reuse. Next the “divide and conquer” paradigm is used, meaning that the set of requirements is divided into smaller parts and each part is first realised by itself and then integrated into the complete solution. For each smaller problem, this is matched to the patterns (a search of the solution space) and some appropriate patterns are selected and reused. 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.

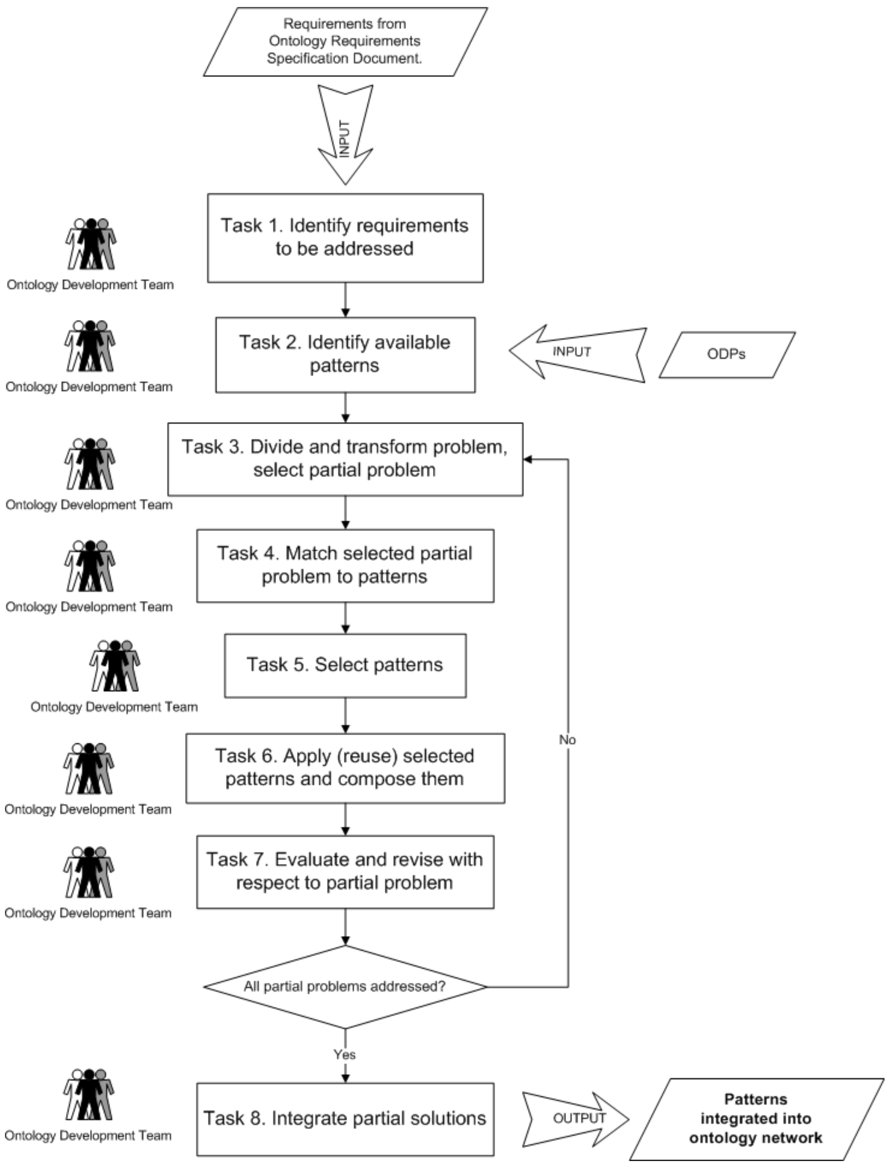

Requirements from Ontology Requirements Specification Document.
INPUT
Task 1. Identify requirements to be addressed
Ontology Development Team
Task 2. Identify available patterns
INPUT
ODPs
Task 3. Divide and transform problem, select partial problem
Task 4. Match selected partial problem to patterns
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 3. General Workflow of Ontology Design Pattern Reuse

The tasks for carrying out the ontology design pattern reuse activity are explained more in depth in the following:

Task 1. Identify requirements to be addressed.

The objective of this task is to identify which requirement(s), from the ORSD, can be addressed by ontology design pattern reuse. In many cases different methods will be used for realizing different parts of the requirements, and not all requirements may have suitable patterns available. The first task is to decide what specific requirements to include in the following steps. This task may be integrated with, or done iteratively together with, the next task, which is to select available pattern catalogues. In case no patterns are available to solve a specific type of problem, then those requirements are probably not likely to be amenable by design patterns; instead they should be realized using alternative activities within the NeOn methodology.

The selection of requirements may be based on the following (non-exhaustive) list of criteria:

- What requirements are not addressed within other activities yet?
- What requirements do not have trivial solutions?

- What requirements can be associated with existing pattern types?

The first question addresses the organizational aspects of the works, which requirements are not treated yet and require a solution to be developed. The second question addresses the fact that patterns do not usually provide solutions to trivial problems, but represent best practices for solving some commonly occurring, repetitive problems. For example creating a single OWL class is quite trivial, and although there may exist a logical pattern solving this issue it is commonly supported as a basic functionality in ontology design environment and tools, therefore applying a pattern-based methodology would in this case introduce overhead and give no added benefit (over using any graphical ontology editing tool). "Trivial" is a relative concept here, in addition to indicating very small problems, it can also be related to the experience and skills of the developer. If a person is highly experienced and skilled in a certain area, then he might find a problem trivial, i.e. he immediately knows the correct solution to the problem. In this case using patterns again introduces an unnecessary overhead. Requirements where it is not immediately obvious (subjectively) how to represent those in a "good" way are generally ideal candidates for design pattern reuse. For example, how to model an n-ary relation might not be trivial to an inexperienced ontology engineer, in this case a content pattern could help. Finally, in order for the requirement to be solvable but design pattern reuse there must at least be pattern types that solve the kind of problems posed by the requirements (the presence of actual suitable patterns is discussed in the next task).

Requirements are included within the ORSD expressed as competency questions (CQs). Different aspects of a CQ may be realized by different kinds of patterns. Below is a list of the available types of OPs and the kinds of problems they are intended to solve (for details see D2.5.1 [67]):

- Correspondence OP – used for either re-engineering or mapping between ontologies.
- Presentation OP – used for naming of elements or for annotating ontologies.
- Reasoning OP – used for introducing certain reasoning capabilities.
- Lexico-syntactic OP – used for linking natural language and ontological elements.
- Structural OP – used for designing the logical structure of ontologies.
- Content OP – used for designing the content of ontologies.

Structural and content OPs are generally useful for most kinds of modeling problems, since they address the realization of the CQs into ontology elements and axioms. Lexico-syntactic patterns may additionally be useful if the ontology design team includes novice users (see method in D5.4.1 [80]) or if a text corpus is used as the basis for building the ontology. Presentation OPs are useful if usability aspects of the ontology are deemed important, and correspondence OPs are relevant when building ontology networks by creating mapping between ontologies or when re-engineering other kinds of resources. Reasoning patterns define the kind of reasoning services needed to provide certain types of information. When choosing the requirements to address with ontology design pattern reuse, these available pattern types should be considered, and requirements covered by one or more of these types may be selected.

Task 2. Identify available patterns.

The goal of this task is to identify as many patterns, or rather pattern catalogues, as possible that could help in solving the modelling issues proposed by the requirements selected in the previous task. Catalogues of OPs may be found both in the form of written documents, like D5.1.1 [82] and D2.5.1 [67], and online catalogues like the ontology design pattern portal[5]. The relevant pattern types of the chosen requirements, identified in the previous task, should guide the search of available pattern catalogues. Aspects to consider when collecting patterns are for example the following:

- Relevant pattern types.

[5] www.ontologydesignpatterns.org

- Relevant problem domain.
- Pattern provenance and reliability.

An example illustrating the criteria above could be the problem of realising a CQ exposing an n-ary relation in the fisheries domain, such as connecting an aquatic resource observation to the observed resource, the time of the observation, and the measured parameters. Both logical and content pattern may be relevant, as well as presentation patterns for increasing usability aspects. The domain is fishery, whereas only general domain independent patterns and pattern for this specific domain should be considered (not for example patterns from the financial domain). If this ontology is to be part of a safety critical system, then only well-known patterns and proven best-practices would be considered, while in a less safety critical case even pattern candidates (not yet approved by the community) could be considered for inspiration.

Task 3. Divide and transform the problem, select a partial problem.

The goal of this task is to prepare the set of modeling problems posed by the requirements for matching to the set of available patterns. Pattern-based design is inherently a divide-and-conquer approach, since patterns are restricted and solve a specific problem. In order to match the problem to such small partial solutions and reuse them, the problem needs to be divided into manageable pieces. In addition the problem could be divided in order to let different groups or individuals solve different sub-problems, not all the design team may be working on the same parts of the problem throughout the development.

This task could include transformations like writing CQs to represent requirements stated only in example scenario sentences if not already present in the ORSD, and grouping of similar CQs that may be solved together by one or more pattern types if not already present in the ORSD, etc. It is recommended that pattern types that may affect the overall organization of the ontology are treated first, while detailed patterns are treated later. For example, presentation patterns such as naming conventions should be treated at the beginning since this will minimize the refactoring needed when applying the pattern to the solution. Similarly, reasoning patterns and structural patterns in the form of architecture patterns are recommended at an early stage, also in this case to minimize later refactoring.

Finally, when the problem has been transformed and divided, one such “manageable piece” (for example, a set of CQs treating a coherent part of the intended ontology and envisioned to be solvable by logical or content patterns) is selected as a starting point. The rest of the tasks may be carried our iteratively, so that each sub-problem is solved before the next one is addressed, or in parallel, so that the sub-problems are divided between groups of designers and all groups solve their specific sub-problems in parallel (all using the tasks specified).

Task 4. Match selected partial problem to patterns.

The goal of this task is to identify which patterns are able to solve which parts of the selected sub-problem, if any. How this task is solved is, of course, highly dependent on the type of patterns that are used. The matching procedure will differ a lot between, for example, matching naming pattern or reasoning pattern compared to matching content patterns. This task is identified as one of the hardest tasks of the process, and it is also a key task for the success of the complete process. This makes it the primary candidate for future tool support, but at the moment very little tool support is available. The ontology design pattern portal provides some search functions for finding patterns in the catalogues, but no support for the actual matching. Forthcoming versions of the planned XD plug-in (supporting a specific method for reuse of in the first version mainly content OPs described further lather in this chapter) for the NeOn toolkit will provide more support for pattern-based design and also the matching task.

Some simple guidelines may be provided for specific types of OPs (see typology in D2.5.1 [67]). In the previous deliverable, D5.4.1 [80], the support for selecting structural and logical patterns based on natural language and lexico-syntactic patterns was described. For matching naming and annotation patterns usually a manual reading of the guidelines accompanying the patterns is enough, since these patterns are more like guidelines and propose for example naming conventions (in these cases the selection and reuse of the patterns is the more challenging part).

For matching correspondence OPs, support is being developed in the field of ontology matching (see for example [71]), but so far no general guidelines exist. Finally, also in the field of matching of content patterns only a few supporting tools may be found, one being the OntoCase approach as presented in [8], where suggestions for suitable content patterns are proposed base on ontology matching, ranking and learning algorithms.

To summarize, there are no detailed general guidelines for matching a problem to a pattern at present, even though this is a crucial task within the overall process. More specific guidelines have to be tailored to each of the pattern types (as we shall do later in this chapter, for content patterns), we can only propose two brief general suggestions for performing this task in order to assist the matching:

- Identify the type of patterns that may be suitable for solving the problem (if not already done in previous tasks) and match problem primarily to these kinds of patterns.
- Depending on the type of pattern, use the description of the intent of the pattern to match it to the problem (for content patterns this would mean to match the competency questions it intends to solve to the competency questions of the problem).

Task 5. Select patterns.

The goal of this task is to select a set of patterns to reuse based on the results of the matching done in the previous task. These results may be of varying kind, if a formal method or tool was used then the matching results may be in the form of ranking values of a set of patterns. In that case the selection may be done by setting a threshold value. Still, other things than the individual matching results may be useful to take into account, since there may be overlapping or alternative patterns so that it is perhaps not suitable to select all of them.

If manual matching was performed this is a decision-making process, where the usefulness of the pattern is weighted against the overhead of reusing it (instead of for example constructing a new solution). In many cases however it will be sufficient to simply study which patterns cover some part of the problem area and decide to reuse all of them, then applying a manual conflict resolution method in case there are overlaps or other conflicts that arise later.

Task 6. Apply (reuse) selected patterns and compose them.

The goal of this task is to reuse the selected patterns and compose them. How a pattern can be reused is of course again highly dependent on the pattern type. For example to reuse a reasoning pattern may involve to adapt the complete ontology for supporting this particular way of reasoning, as well as selecting a reasoning engine to perform the actual reasoning task. While reusing a content pattern may be to import it into an ontology file and specialise its elements and axioms. Also partial reuse of a pattern is possible, if not the complete pattern is needed or even appropriate.

An important part of this task is the composition of patterns. It is rare that a single pattern can solve the complete problem we are trying to address, even if it is very small. In many cases two or more patterns will have to be combined, and this combination task is called pattern composition. Composition of content patterns would involve adding the union of the elements and axioms from the patterns to the resulting ontology, but additionally to connect the elements from different patterns using additional elements and axioms in order to ensure that the pattern really solves the problem (in case of content patterns this is to be able to answer the CQs posed).

A certain amount of conflict resolution may be needed if patterns involve contradicting parts, but as stated above partial pattern reuse is also possible. At this stage however the focus is still on the partial selected problem intended to be solved, not on the complete solution for all the selected requirements. Integration of the complete solution is performed later in the process.

Task 7. Evaluate and revise with respect to partial problem.

The goal of this task is to test the solution with respect to the selected partial problem at hand and to ensure that it really solves this problem in a correct way. As stated at the beginning a pattern-based approach is inherently a divide-and-conquer approach, and this also leads to the possibility to test small and manageable pieces of the solution before finally integrating them into the complete solution. It does not replace the evaluations and revision of the complete solution, but these small “unit tests” are an important part of the process.

If the result of the last step is a small ontology then the ontology may be evaluated and tested for example through adding instances and running unit tests and queries corresponding to the CQs of the problem. If the evaluation identifies some problems of the solution then these problems should be addressed in this task, before proceeding to the next. When the solution passed all tests, the iteration continues, by performing tasks 3-7 again on the next partial problem until all problems have been covered, or if the problems were solved in parallel by immediately integrating all solutions.

Task 8. Integrate partial solutions.

The goal of this task is to integrate the solutions to the partial problems solved by the previous iterations, or by the parallel teams working on different sub-problems. If the division of the problem resulted in a large number of partial tasks the number of partial solutions will also be large and this process may be quite challenging.

In practice this integration may be performed after each solution of a small partial problem, instead of at the end of the complete process. The choice of method may depend on the team organisation. If arger design teams work on sub-problems in parallel and integrate their solutions as soon as they finish, the task needs to be performed in a collaborative fashion (supported by collaboration tools like chats, message boards or argumentation tools like Cicero [27]). On the other hand, if only one team is working in an iterative fashion, this task is less collaborative and simply contains the task of integrating one more “piece” into the complete solution.

Either way, integration of the solutions is a challenging problem, and may involve refactoring of the whole solution. So far, detailed guidelines of this process are still future work, although technical guidelines for integrating ontologies are of course present.

4.3. Reuse of Content Patterns – the eXtreme Design Approach

In this section we describe a specific method, concerning content ontology design patterns for performing the tasks 3-7 of the general guidelines for ontology design pattern reuse proposed in Section 4.2. This specific method is intended, in the future versions, to cover the complete set of tasks but task 8. Integration is still a challenge for the future work, as well as the detailed support for the two selection tasks (1-2), thus they will not be described here.

The eXtreme Design (XD) method is inspired by eXtreme Programming (XP) in software engineering, whereby we start by describing the background and basics of this methodology. Next, some guidelines are proposed and finally an example is given.

4.3.1. Background

The methodology of eXtreme Programming [5] has evolved over the last decade, as part of the agile software development movement. The main idea of agile software development is to be able to incorporate changes easily, in 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. Also the customer is highly involved in all stages, whereas such big change requests can often also be avoided. One of the main objectives is to have a working software as early as possible, even if it has only a very small subset of the intended functionality, in order to let the customer start using it and thereby prevent any big surprises at the end of the project.

One of the main promoters of XP has been Kent Beck, who in his 1999 cover feature in IEEE Computer wrote a comprehensible summary of the XP method and its practices [5]. XP is based on a set of rules, or practices that should always be followed as closely as possible in order for the project to be successful. Beck lists the following rules in [5]:

- Planning. The customer will decide the timing of the next release, and what it should cover, based on estimates of development effort. The developers implement only this functionality and nothing more.
- Small releases. The system is launched already after a few months, new releases are made often.
- Metaphor. The shape of the system is described by a metaphor, understandable by both customer and developers.
- Simplicity. At any given time the system must be able to pass all tests, it must at the same time be as simple and clear as possible and contain no redundancy.
- Tests. Two types of tests are used, unit tests written by the developers and functional tests developed by the customer, all tests collected so far must run correctly at any time of the project.
- Refactoring. The design is evolved when necessary throughout the project, so that it is as simple as possible and all tests are still running.
- Pair programming. All code of the system is written by two people sitting at one computer.
- Continuous integration. New code is integrated daily; all tests must run after each integration.
- Collective ownership. Anyone can change anything in the system at any point in time, if it improves the system.
- On-site customer. The customer is physically present at all times during development.
- 40-hour week. Overtime should be avoided, and is not allowed two weeks in a row.
- Open workspace. The team works all together in a large room.
- Rules. These rules have to be followed by all members of the team, but may be re-negotiated at any point if it benefits the project.

The main idea is that the problem, the current customer requirements, are divided into small working releases, but only the next release is ever planned in detail. This means that the customer picks a set of requirements (in XP called "stories") that he feels are the most important at the moment and the release time is planned based on the resources available. Then only those requirements are treated during that iteration, nothing else. The requirements are further subdivided into pieces that can be realised by the developers, who then work in pairs to develop solutions for each little piece of the problem. The new solutions are first tested and then integrated into the current system, the system is refactored to improve the design and keep all tests running without problems. In this way the system is built incrementally by adding small pieces of new functionality.

Testing is the main driver behind improvements and the thing that keeps the system from developing in an uncontrolled way. Tests are collected continuously and all previous tests have to run without problems even when new functionality is added, this guarantees a system that is always stable and which is really growing towards the complete solution rather than implementing random functionalities that introduce inconsistencies.

4.3.2. Proposed guidelines for reusing content patterns: eXtreme Design

The eXtreme Design (XD) method is inspired by XP in many ways but its focus is a bit different, because where XP diminishes the value of careful design this is exactly what XD has at the 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 in the same way as XP and also applies the divide-and-conquer approach; this is one of the reasons why it is perfectly suited for use with content patterns. The general XD method may be illustrated as a workflow in Figure 4. The method is, as mentioned previously, focused on content design patterns, whereas this method should be used when/if a set of content patterns have been selected (see general workflow, task 2 in Figure 3) to realise some set of requirements.

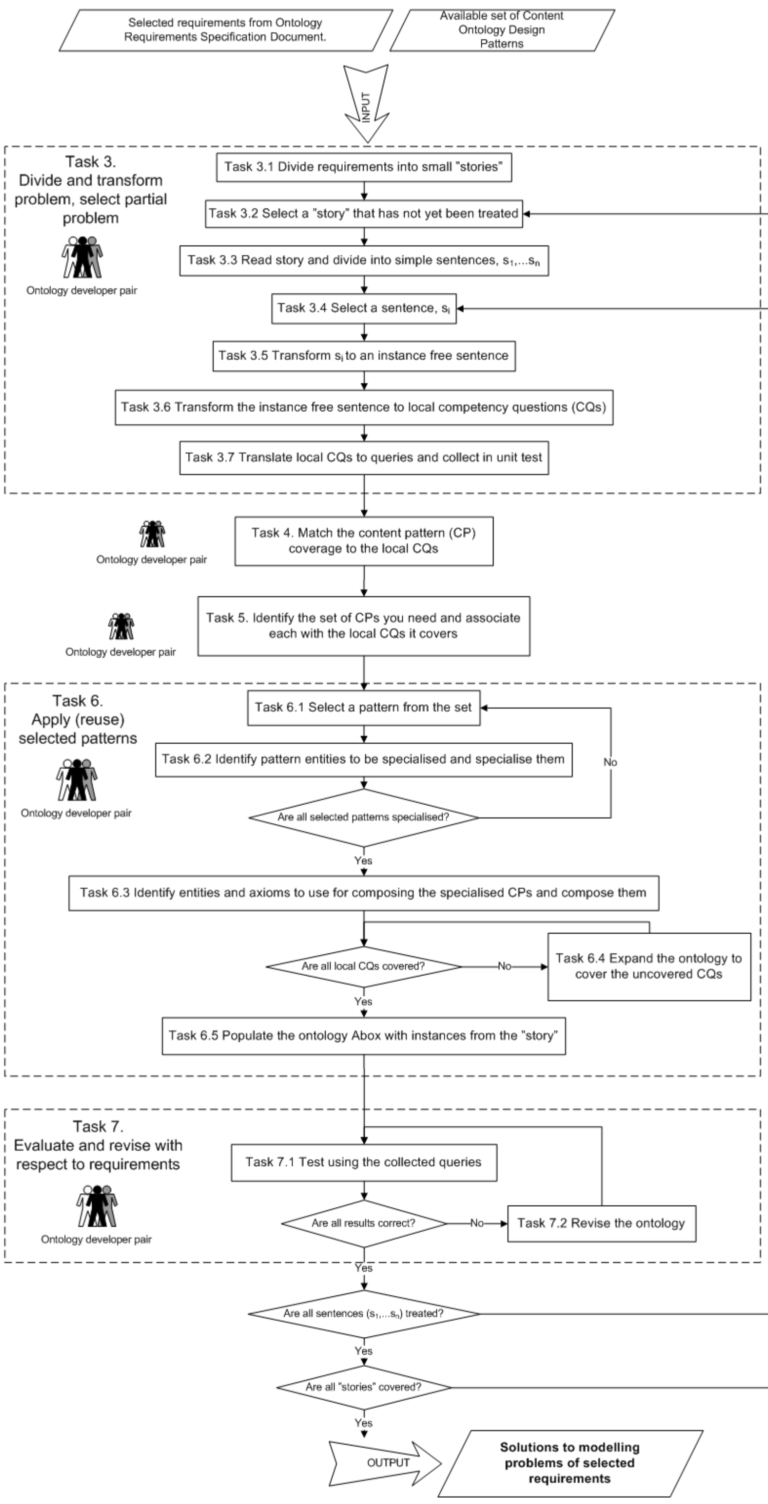

Selected requirements from Ontology Requirements Specification Document.
Available set of Content Ontology Design Patterns
INPUT
Task 3. Divide and transform problem, select partial problem
Ontology developer pair
Task 3.1 Divide requirements into small "stories"
Task 3.2 Select a "story" that has not yet been treated
Task 3.3 Read story and divide into simple sentences, s1,...sn
Task 3.4 Select a sentence, si
Task 3.5 Transform si to an instance free sentence
Task 3.6 Transform the instance free sentence to local competency questions (CQs)
Task 3.7 Translate local CQs to queries and collect in unit test
Task 4. Match the content pattern (CP) coverage to the local CQs
Ontology developer pair
Task 5. Identify the set of CPs you need and associate each with the local CQs it covers
Ontology developer pair
Task 6. Apply (reuse) selected patterns
Ontology developer pair
Task 6.1 Select a pattern from the set
Task 6.2 Identify pattern entities to be specialised and specialise them
Are all selected patterns specialised?
No
Yes
Task 6.3 Identify entities and axioms to use for composing the specialised CPs and compose them
Are all local CQs covered?
No
Task 6.4 Expand the ontology to cover the uncovered CQs
Yes
Task 6.5 Populate the ontology Abox with instances from the "story"
Task 7. Evaluate and revise with respect to requirements
Ontology developer pair
Task 7.1 Test using the collected queries
Are all results correct?
No
Task 7.2 Revise the ontology
Yes
Are all sentences (s1,...sn) treated?
No
Yes
Are all "stories" covered?
No
Yes
OUTPUT
Solutions to modelling problems of selected requirements

Figure 4. The overall Workflow of XD

Below, detailed guidelines are described for each of the tasks included in Figure 4. We remind the reader that task 1 and 2 are not specialised by this specific method, for guidelines to these tasks see the general guidelines for design pattern reuse presented in Section 4.2. Also, no detailed guidelines are present for task 8; therefore it is only presented as a future work in the description below.

Task 3. Divide and transform problem, select partial problem.

Task 3.1. Divide requirements into small "stories".

The objective of this sub-task is to transform and divide the requirements into small "stories" that address some more or less coherent subpart of the problem. A story could be a set of example sentences describing the type of information to be stored within the ontology (or ontology network), or a coherent set of CQs (if collected directly from the ORSD document as specified previously) that address a related part of the ontology network. The stories may include examples, instance-level information, since this can be used later for testing the requirements. If the requirements are already in the form of CQs, this sub-task should simply divide the CQs into groups that are somehow related and will be addressed together. CQ groups could be also directly extracted from the ORSD. This task may be performed by the development team together, a pair of developers or even preformed previously during the requirement specification process.

Task 3.2. Select a "story" that has not been treated.

The objective of this sub-task is to select a sub-problem, a small "story", and to start the development iteration. At this point the process may be branched so that different development pairs select different "stories" and work in parallel or the process is run iteratively with only one pair. In case several pairs are working in parallel their respective background and experience may be considered when selecting the "stories", and some negotiation process may be suitable.

Task 3.3. Read "story" and divide into simple sentences, s_1 ... s_n.

The objective of this sub-task is to further sub-divide the story into simple sentences, meaning that complex example sentences may be broken up into shorter sentences to increase clarity. In case the story is already in the form of CQs the CQs may be broken down into simpler CQs, in order to increase clarity and further divide the problem (this task may already have been performed during the ORSD creation, but is included since this method is more general and can handle different kinds of input). The sentences may be numbered in order to be able to refer to them and check which of them are related to certain CQs, tests and patterns in the coming steps.

Task 3.4. Select a sentence, s_i.

The objective of this sub-task is to pick one of the sentences (or CQs, in case this was already the requirement format present) that have not been treated yet. This may be viewed as an "inner loop" of the divide-and-conquer approach, where even the story is further sub-divided into its pieces and now each of the pieces are treated one by one.

Task 3.5. Transform s_i to an instance-free sentence.

The objective of this sub-task is to remove any references to instances from the sentence (or CQ), i.e., to abstract the sentence so that all the terms refer to what will be on the T-box level in the ontology. A name of a person would for example be replaced with a more generic term "person".

Task 3.6. Transform the instance-free sentence into local CQs.

The objective of this sub-task is to transform a sentence into a set of local CQs that covers all what is contained in the sentence. In case the requirements were already in the form of CQs, this task only concerns to check that the CQs are really local (see below) and that they cover all the information we intend to model. Using the CQs we should be able to retrieve all the information contained in the sentence from the ontology, so for example the instances that were removed from the sentence in sub-task 3.5. Still, the CQs should be local in the sense that they do not require any additional sentence or part of the ontology, only the parts represented by the sentence.

Task 3.7. Transform local CQs to queries and collect in unit tests.

The objective of this sub-task is to construct the tests that will be used to later check that the solution developed actually solves the problem in a correct way. Such queries could be expressed as SPARQL queries, but any other query language suitable may also be used. Unit test should be formed, i.e. collections of queries, that can be run in order to test the solution to this particular sub-problem against the local CQs.

Task 4. Match the content pattern (CP) coverage to the local CQs.

The objective of this task is to match the local CQs to the general CQs present in the descriptions of the content patterns (the intent of the pattern), in order to determine whether a certain pattern could solve some part of the problem. At the moment there is no tool support for search and matching based on CQs of the content patterns, but this is a future work issue that will be provided in future versions of the XD plug-in connecting the design pattern portal and the NeOn toolkit. Currently this, sometimes difficult, task has to be performed manually by abstracting from the local CQs and trying to match these to the pattern CQs. The result is a set of suitable content patterns.

Task 5. Identify the set of CPs you need and associate each with the local CQs it covers.

The objective of this task is to identify the set of CPs you need among the ones that matched some parts of the problem in the last task. There may be more than one pattern that can solve the problem at hand, and different combinations may be possible. As a general guideline, more specific patterns should be preferred over more general ones, since a more specific pattern will add more information to the solution than a more general pattern. Each pattern selected should also be connected to the local CQs it covers in order to identify in later steps the uncovered CQs and to know which pattern was used to realise what part of the solution.

Task 6. Apply (reuse) selected patterns.

Task 6.1. Select a pattern from the set.

The objective of this sub-task is to initiate an iteration over the selected patterns from the previous task and to thereby treat them one by one.

Task 6.2. Identify pattern entities to specialise and specialise them.

The objective of this sub-task is to specialise the pattern so that it contains the concepts and properties named in the instance-free sentence and in the CQs that should be covered, rather than merely the general elements and axioms of the pattern itself. For example, a pattern containing the concept of "agent" many be specialised by introducing the sub-concept "person" as a subclass of "agent" in order to support a CQ containing the concept "person". This is done for each of the selected patterns before proceeding, usually the patterns are also imported into the same ontology at this stage (even though they may not be connected, see next task).

Task 6.3. Identify entities and axioms for composing the specialised CPs and compose them

The objective of this sub-task is to compose the set of patterns solving the problem specified in the particular sentence treated. If a set of patterns were selected then they usually need to be connected in order to be able to correctly answer the CQs and corresponding queries associated to the sentence. This means to import all the patterns into one ontology, if not already done, and it could also mean to add new elements or axioms in order to connect entities from the different patterns. How this is done and what is added depends on the particular CQs to be supported, the CQs can be checked one by one until each composition operation needed is covered.

Task 6.4. Expand the ontology to cover uncovered CQs.

The objective of this sub-task is to make sure that all the CQs of this sentence is completely covered. It may be the case that there were not patterns suitable for all CQs, or that some pattern only partially solves a CQ. In this case the remaining parts have to be modelled "from scratch" in order for the ontology to solve the complete problem. This task aims at extending the composed patterns in order to cover all local CQs of the sentence currently treated.

Task 6.5. Populate the A-box with instances from the "story".

The objective of this sub-task is to populate the current small ontology with instances from the initial story, in order to prepare it for the unit test, containing a set of queries that was prepared previously. If instances are missing also some "test case instances" may be introduced, in order to support the testing of the ontology. These may be later removed.

Task 7. Evaluate and revise with respect to requirements.

Task 7.1. Test using the collected queries.

The objective of this sub-task is to test the constructed small ontology against the sentence it is supposed to represent, using the queries collected in sub-task 3.7. The queries are run on the populated ontology and the results are registered and compared to expected results.

Task 7.2. Revise the ontology.

The objective of this sub-task is to revise the ontology in case it did not pass all the tests in the previous sub-task. The revision and testing will continue until all tests are passed. When the queries run with correct results, the next sentence or story is picked and the iteration starts over from task 3.4 or task 3.2. Optionally the integration may also be performed directly after each iteration, or if all sub-problems were solved in parallel the next step may in any case be the integration of solutions.

[Future work] Task 8. Integrate partial solutions.

The objective of this sub-task is to integrate partial solutions constructed through iterations of the above tasks. Either the above tasks may be performed in parallel, when this task is then the problem of integrating the solutions for a set of *n* stories. In other cases the integration may be done after each iteration, whereas the problem is more similar to the integration and refactoring of the software XP process. In any case, no detailed guidelines for performing this process are available yet, this is still part of future work.

4.3.2. Example eXtreme Design iteration

Below, an example iteration of the XD method is presented, based on a use case from the fisheries domain. The scenario has been slightly simplified, compared to the original use case. In the ORSD the scenario would be additionally connected to a set of specific requirements in the form of CQs, but let us, for the sake of illustration, assume that the scenario is only described in the form of examples. Let us assume that the initial scenario to be modelled can be described through the following "story":

In 2004 the resource of species "Tuna" in water area 24 was observed to be fully exploited in the tropical zone at pelagic depth.

This is assumed to be the problem "story" we are working with, whereas tasks 3.1 and 3.2 are already performed. We mentioned that a catalogue of content patterns needs to be present. Such a catalogue can be found at the ontology design pattern portal[6], where we assume the following general (more or less domain independent) patterns to be present:

- AgentRole
- Classification
- CollectionEntity
- Constituency

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

- ❑ Description
- ❑ InformationRealization
- ❑ PartOf
- ❑ RoleTask
- ❑ Situation
- ❑ TimeInterval

Starting from **task 3.3** the goal is to divide the story into simpler sentences. A first division may result in the following:

Resource x was observed to be fully exploited in the tropical zone at pelagic depth in 2004. The resource x is species "Tuna" in water area 24.

Next **task 3.4** implies to select one of the two sentences to start modelling. Assume that we select the first sentence. **Task 3.5** then indicates that we should transform this sentence into an instance-free sentence. The sentence will look like:

A resource was observed to be in a certain exploitation state in a particular climatic zone at a certain vertical distance in a certain year.

Next **task 3.6** instructs us to transform the sentence into local competency questions. The following set of CQs can be derived from the sentence:

What resource was observed? In what year? In what exploitation state? In what climatic zone? At what vertical distance?

These questions may then be translated into queries, as proposed by **task 3.7**, and collected for later testing of the ontology. A set of SPARQL queries could be formed as follows:

SELECT ?x ?y WHERE {?x ?r ?y. ?x a :AquaticResource. ?y a :AquaticResourceObservation. }
SELECT ?x ?y WHERE {?x :hasObservationYear ?y. ?x a :AquaticResourceObservation. }
SELECT ?x ?y WHERE {?x :hasExploitationState ?y. ?x a :AquaticResourceObservation. }
SELECT ?x ?y WHERE {?x :hasClimaticZone ?y. ?x a :AquaticResourceObservation. }
SELECT ?x ?y WHERE {?x :hasVerticalDistance ?y. ?x a :AquaticResourceObservation. }

It is of course important to note that it is not necessary to formulate the queries exactly like this; there may be other ways to formulate them and languages other than SPARQL can be used. For this simple example however, this set of queries will be used.

Next is **task 4**, where the local CQs should be matched to the intent of the content patterns available (see the list above). This task is a hard problem, to be solved manually, and it requires some experience to be solved quickly. Still, if studied one by one, we will find that the two patterns that most closely resemble the case at hand are the Situation-pattern and the TimeInterval-pattern. The competency question of Situation "What entities are in the setting of a certain situation?" can be said to match the observation and the resource and the parameters that are in the setting of that observation. Additionally the TimeInterval pattern may be seen as partially matching the question of what year a certain observation was made, although this could also be solve with just a simple datatype property. The pattern 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 task 4 is then two matching patterns.

In **task 5** the objective is to select which of those patterns should be used for solving the modelling problem of the current sentence. For the sake of this example we may decide that the TimeInterval pattern adds too much extra effort, besides the needed year of observation, in which case we will only select the Situation-pattern. Thus, **task 6.1**, selecting a pattern to start with, means treating the only selected pattern.

Task 6.2 proposes to specialise the elements of the selected pattern. The situation pattern can be illustrated as in Figure 5. The particular situation is in our case the observation, and the thing being observed is the resource, but additionally the exploitation state, climatic zone, and vertical distance of the observation is also a part of 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 `situation:hasSetting`, for connecting the observations to the resources and the different parameters. The resulting specialisation is illustrated in Figure 6.

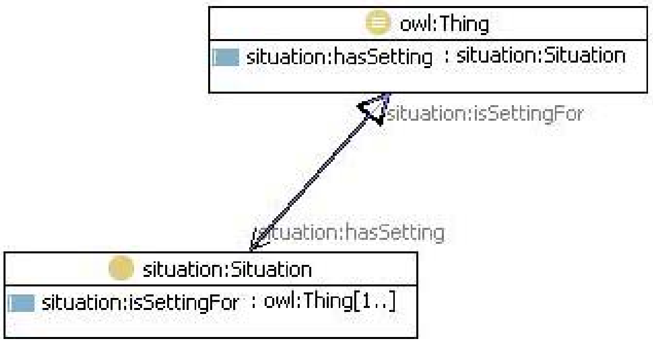

owl:Thing
situation:hasSetting : situation:Situation
situation:isSettingFor
situation:hasSetting
situation:Situation
situation:isSettingFor : owl:Thing[1..]

Figure 5. The Situation Pattern

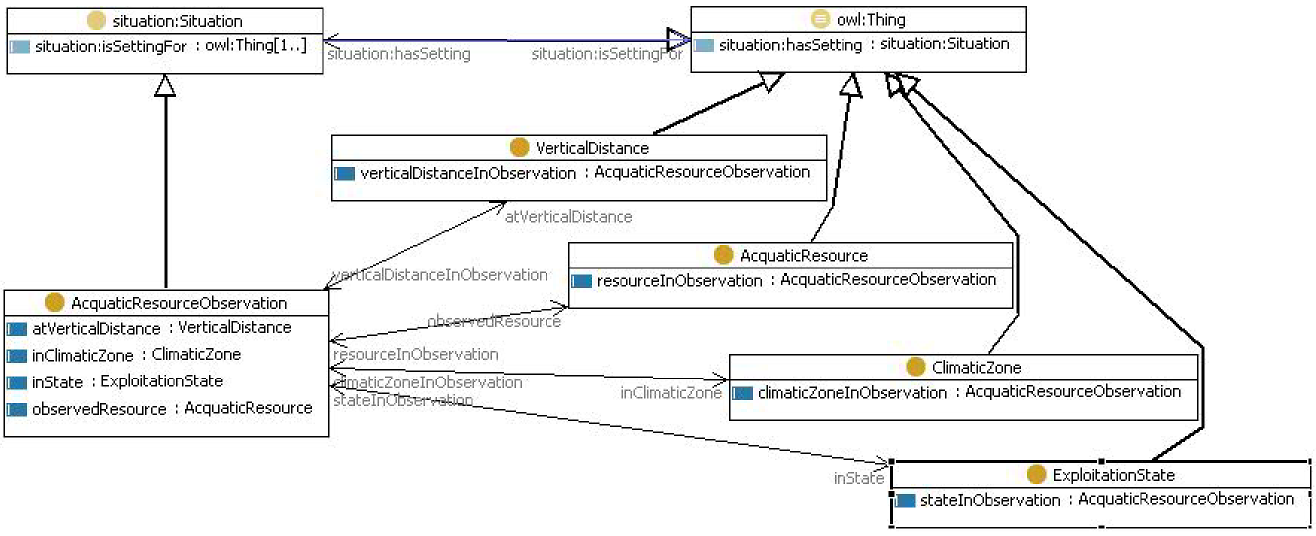

situation:Situation
situation:isSettingFor : owl:Thing[1..]
situation:hasSetting
situation:isSettingFor
owl:Thing
situation:hasSetting : situation:Situation
VerticalDistance
verticalDistanceInObservation : AcquaticResourceObservation
atVerticalDistance
verticalDistanceInObservation
AcquaticResource
resourceInObservation : AcquaticResourceObservation
AcquaticResourceObservation
atVerticalDistance : VerticalDistance
inClimaticZone : ClimaticZone
inState : ExploitationState
observedResource : AcquaticResource
observedResource
resourceInObservation
climaticZoneInObservation
stateInObservation
inClimaticZone
ClimaticZone
climaticZoneInObservation : AcquaticResourceObservation
inState
ExploitationState
stateInObservation : AcquaticResourceObservation

Figure 6. The Specialised Situation Pattern, including Resource Observations

Since this was the only pattern selected we proceed to **task 6.3**, but this is already prepared since there is no need for pattern composition with only one pattern. Next, we check if all local CQs are covered and discover that the year of the observation is still missing. We therefore extend the small ontology we have constructed with a datatype property called “yearOfObservation” of the AquaticResourceObservation class, as specified by **task 6.4**.

The final sub-task of task 6 is **task 6.5**, where we populate the small ontology with the instances from our story. We add an instance of the resource observation, and connect it to an example resource, add the year “2004”, and instances “tropical”, “pelagic” and “fully exploited” as instances of the other classes and connect them through the properties to the observation. Let us for the sake of the example say that we forget to add the connection between the observation and the ClimaticZone instance “tropical”.

Then the small ontology is tested, in the context of **task 7.1**, using the queries that were collected in task 3.7. All queries will give expected results except the fourth one, where not instances will be retrieved. Thus we discover our mistake of forgetting to instantiate the property, and we can proceed to **task 7.2** in order to correct the ontology. In this case the mistake was rather in the addition of example data and not in the modeling of the actual ontology, and the problem is easily corrected. We return to **task 7.1** and conclude that all tests now run with expected results.

The next step is to check if all sentences in the story were treated. In our case we realize that we have one sentence left and therefore proceed to the next sentence (**task 3.4**), which was the following:

> *The resource x is species "Tuna" in water area 24.*

Task 3.5 instructs us to transform this into an instance free sentence, which results in the following:

> *The resource x is a certain species in a certain water area.*

The local competency questions of this sentence (see **task 3.6**) are:

> *What are the resources of this species? What are the resources in this water area? What is the species and water area of this resource?*

These are then transformed into queries in **task 3.7**:

SELECT ?x ?y WHERE {?x ?r ?y. ?x a :AquaticResource. ?y a :AquaticSpecies. }
SELECT ?x ?y WHERE {?x ?r ?y. ?x a :AquaticResource. ?y a :WaterArea. }
SELECT ?x ?y ?z WHERE {?x :hasSpecies ?y. ?x a :AquaticResource. ?x :hasWaterArea ?z. }

When performing **task 4** we do not find any one of the 10 patterns to fit our case at hand. This is a very likely situation that will occur many times, since content patterns do not cover all possible situations that may arise in modelling but mainly the "problematic" ones where some specific modelling issues need to be solved. In this case the modelling is quite straightforward. Of course a specific "resource pattern" could be constructed where resources are connected to some specific thing that is the content of the resource and some location where the resource is located, but since our catalogue lacks such a resource pattern task 4 does not result in any patterns.

In this case sub-tasks 5 and 6.1-6.3 are not applicable, and we directly proceed to **task 6.4**. We model the sentence from scratch, by adding concepts for AquaticResource, AquaticSpecies and WaterArea. At this stage there are two possibilities, either this second sentence is modelled in a different file (separate from the modelling of the first sentence, we could, for example, have assigned the modelling of the two sentences to two separate design pairs to be performed in parallel) than the first sentence or the modelling is continued in the same physical ontology file as the first sentence. In case we are continuing within the same ontology file, we may immediately discover some refactoring possibilities. Since AquaticResource was a concept also used in the first sentence, we may not want to add a new concept representing this again, but instead to continue to use the already defined concept. In case we are starting a new ontology file for this sentence, this refactoring will take place in the integration step after we have finished the modelling of the sentence. We continue to add also properties between the resources and the species and water areas.

In **task 6.5** the ontology is again populated with instances of the story, whereas we add an example resource (if not already present) the species "Tuna" and a water area with number 24. The solution is then tested in **task 7.1**, through the queries developed previously, and if all queries run with expected results the **task 7.2** is not needed. Now, we have covered all sentences in the example story, and assuming that no more stories are to be covered we may proceed to the integration. Note that, as mentioned previously, the integration and refactoring may in practice be done partly during the development process, or at least between the treatment of different stories. This process has to be adapted to the organisational setting at hand, and thereby the workflow should not be viewed as strict and prescriptional, but rather descriptional indicating the possible steps and their dependencies. Still, as mentioned previously, integration is an important issue and deserves further research in order to develop detailed guidelines also for the collaborative integration of partial solutions.

4.4. Future Work

In this chapter we have presented preliminary methodological guidelines for carrying out the overall ontology design pattern reuse activity. It may be noted from the general guidelines of Section 3.2 that they are quite brief and very general. This is due to the great diversity of ontology design patterns that exist. It is not possible to present detailed guidelines for all possible patterns at once. Instead detailed methodological guidelines should be presented for each type of pattern (see typology in D2.5.1 [67]). In the previous version of this deliverable, D5.4.1 [80], a method tailored for naive users was presented, which introduced reuse of logical patterns by means of using lexico-syntactic patterns. In this deliverable we have presented a specialisation of the overall method for reuse of content patterns. The main further work needed for reusing OPs is thereby to cover all of the different types of patterns, so that specific methods may be chosen based on the pattern types available, rather than only relying only on the general guidelines of the overall process.

Additionally, the specific method proposed for content patterns still needs some further work, with respect to evaluation of the method and development of detailed guidelines also for the final step, i.e. result integration. The method has already been tried in many project use-cases, since it is based on long experience and work in ontology engineering, but previously it has not been described at this level of detail. In the particular form described above it has only been used in NeOn training events and during the experiments on patterns presented in NeOn deliverable D5.6.2 [30]. Next steps are to experiment specifically on this method, refine it and empirically determine its strengths and weaknesses.

Additionally, the need for tools supporting the general activity of ontology design pattern reuse has been observed. At the moment there exist no tools that explicitly support pattern reuse. Within the NeOn toolkit this will be supported through the XD plug-in being developed. Initially the plug-in will support mainly the XD method, as described in Section 4.3, but it is not in principle restricted to only content patterns for example. The first version of the plug-in will support basic search and retrieval of patterns, importing patterns, composing them etc. This will be supported through a graphical user interface intuitive to ontology engineers.

5. Ontology Modularization

Designing ontologies in a modular way is generally considered as a good practice [68]. However, there may be scenarios where ontology engineers need to reuse or exploit an ontology that has not been modularized at design time, or for which the criteria applied for distinguishing modules do not fit the particular requirements of his/her current application. Therefore, we consider the activity of **ontology modularization** as one responsible for creating modules from an existing ontology. This activity, and therefore the present methodological guidelines, is supported by several operators included in the NeOn toolkit for modularization, as described in [20]. The importance of such an activity is motivated by several different scenarios within the NeOn case studies and technical workpackages, as described in [21]. The proposed operators are designed to be generic, in order to be useful in the majority of the modularization scenarios. As such, they have to be used in an interactive process, where the user provides the relevant parameters and input. Proper methodological guidance is then required to support him in this task."

Within the NeOn Glossary of Processes and Activities [78], ontology modularization is described as the “the activity of identifying one or more modules in an ontology with the purpose of supporting reuse or maintenance.”

In this chapter we present a brief introduction to the existing techniques and tools for ontology modularization. We also propose the NeOn methodological guidelines for carrying out the activity.

5.1. State of the Art

5.1.1. Techniques and tools for modularization

We consider in this section techniques and tools that have been developed to help users in extracting or creating modules from existing, and potentially large scale ontologies. We start our analysis by briefly introducing notations and distinguishing two major types of modularization: ontology module extraction and ontology partitioning. We then describe different techniques in each of these categories.

We consider an ontology O as a set of axioms (subclass, equivalence, instantiation, etc.) and the signature Sig(O) of an ontology O as the set of entity names occurring in the axioms of O, i.e. its vocabulary. A module is considered to be a significant and self-contained sub-part of an ontology.

Ontology Partitioning. Ontology Partitioning refers to the activity of dividing an ontology into a set of (not necessary disjoint) modules that together form an ontology and that can be treated separately [78].

More precisely, it splits up the set of axioms into a set of modules $\{M_1 , \cdots , M_k \}$ such that each M_i is an ontology and the union of all modules is semantically equivalent to the original ontology O.

Note that some existing approaches being labelled as partitioning methods do not actually create partitions, as the resulting modules may overlap. There are several approaches for ontology partitioning that have been developed for different purposes.

- The approach presented in [50] aims at improving the efficiency of inference algorithms by localizing reasoning. For this purpose, this technique minimizes the shared language (i.e. the intersection of the signatures) of pairs of modules. A message passing algorithm for reasoning over the distributed ontology is proposed for implementing resolution-based inference in the separate modules. Completeness and correctness of some resolution strategies is preserved and others trade completeness for efficiency.

- The approach presented in [16] partitions an ontology into a set of modules connected by ε-connections. This approach aims at preserving the completeness of local reasoning within all created modules. This requirement is supposed to make the approach suitable for supporting selective use and reuse since ever y module can be exploited independently of the others.

- A tool that produces sparsely connected modules of reduced size was presented in [76]. The goal of this approach is to support maintenance and use of very large ontologies by providing the possibility to individually inspect smaller parts of the ontology. The algorithm operates with a number of parameters that can be used to tune the result to the requirements of a given application.

Module Extraction. Ontology Module Extraction refers to the activity of obtaining from an ontology concrete modules to be used for a particular purpose (to contain a particular sub-vocabulary SV of the original ontology) [78].

This activity has been called segmentation in [72] and traversal view extraction in [56]. More precisely, given an ontology O and a set SV □ Sig(O) of terms from the ontology, a module extraction mechanism returns a module M_{SV} , supposed to be the relevant par t of O that covers the sub-vocabulary SV (Sig(M) □ SV).

- Techniques for module extraction often rely on the so-called traversal approach: starting from the elements of the input sub-vocabulary, relations in the ontology are recursively “traversed” to gather relevant (i.e. related) elements to be included in the module. Such a technique has been integrated in the PROMPT tool [56], to be used in the Protégé environment. This approach recursively follows the proper ties around a selected class of the ontology, until a given distance is reached. The user can exclude certain proper ties in order to adapt the result to the needs of the application.

- The mechanism presented in [72] starts from a set of classes of the input ontology and extracts related elements on the basis of class subsumption and OWL restrictions. Some optional filters can also be activated to reduce the size of the resulting module. This technique has been implemented to be used in the Galen project and relies on the Galen Upper Ontology.

- In [75], the author defines a viewpoint as being a sub-par t of an ontology that only contains the knowledge concerning a given sub-vocabulary (a set of concept and property names). The computation of a viewpoint is based on the definition of a viewpoint dependent subsumption relation.

- Inspired from the previously described techniques, [24] defines an approach for the purpose of the dynamic selection of relevant modules from online ontologies. The input sub-vocabulary can contain classes, properties, or individuals. The mechanism is fully automatized and is designed to work with different kinds of ontologies (from simple taxonomies to rich and complex OWL ontologies) and relies on inferences during the modularization process.

- Finally, the technique described in [29] is focused on ontology module extraction for aiding an ontology engineer in reusing an ontology module. It takes a single class as input and extracts a module about this class. The approach it relies on is that, in most cases, elements that (directly or indirectly) make reference to the initial class should be included.

5.1.2. Integrated approaches

One important issue related to modularizing ontologies is that different scenarios and applications require different ways to modularize ontologies [23]. To facilitate the selection, combination, and adaptation the various existing module extraction techniques, [22] describes a parametric approach for this activity. The principle is to describe module extraction techniques under a common framework that can be parameterized according to the modularization technique that is more suited for the application. This framework relies on a graph transformation engine. Ontologies to be modularized are represented as graphs and modularization techniques re-formulated as graph transformation rules. In this way, existing modularization technique can implemented in the same tool, making it easier to compare, adapt and combine them, and new modularization techniques can easily be implemented in the form of modularization rules. The paper [22] described the reformulation of several existing techniques for modularization, but an operational implementation of the tool has not been made available.

Very similar ideas to the one described in [22] are at the basis of another approach for parametric modularization [28] which, instead of a graph transformation framework, employees a mechanism that recursively execute SPARQL queries over the ontology to build a sub-set of it. The parameters of this framework are the sets of SPARQL query that represent modularization techniques. In the same line of ideas, the SAIQL [46] query language has been applied to ontology customization [custom], which can be related to the task of extracting modules from ontologies.

5.1.3. Initial guidelines for modularization

A study for a particular scenario of various modularization techniques and of the different criteria for their evaluation has been described in [19]. For this study, modules were extracted from several ontologies, using several different techniques, for an application in knowledge selection, i.e., requiring to obtain only the relevant part of the ontologies according to given set of keywords. A number of criteria have been identified to characterize the modularization techniques and their results. These criteria were then use to evaluate the application of each technique to this particular scenario.

Amongst other conclusions, this study showed the need for clear and comprehensive guidelines for application developers to employ modularization, and in particular, to select the appropriate techniques and criteria according to the requirements of the application.

5.2. Proposed Guidelines for Ontology Modularization

As we mentioned before, the goal of ontology modularization is to obtain a module or a set of modules from an ontology, which fit the requirements of a particular application or a particular scenario. As already commented, there is a clear need for methodological guidelines, to help ontology developers in selecting and applying the appropriate techniques for modularization depending on the goal of modularization. For this purpose, in the NeOn methodology we start by detailing the filling card for the activity of ontology modularization, and by proposing a set of detailed tasks for this activity.

Note that, as opposed to a single, monolithic ontology, an ontology network is essentially a modular ontology, made of components (the individual ontologies) interacting with each other in a particular context. The approach presented here is applied on individual ontologies (possibly networked) to create either networks of ontologies or elements for networks of ontologies.

In the framework of the NeOn methodology for building ontology networks, we propose the ontology modularization filling card, presented in Table 4, which includes the definition, goal, input, output, who carries out the activity and when the activity should be carried out.

Ontology Modularization	
Definition *Ontology Modularization refers to the activity of identifying one or more modules in an ontology with the purpose of supporting reuse or maintenance.*	
Goal The modularization activity offers a way to cut-down potentially large ontologies into smaller, more manageable modules.	
Input An ontology.	*Output* A module or a set of modules from the input ontology. In practice, ontology modules are themselves ontologies.
Who Ontology engineer (ontology development team), curator of the ontology, preferably with the help of domain experts.	
When In scenario 3 for facilitating the reuse of ontological resources and in scenario 4, as part of the re-engineering process. Also in scenario 8, as part of the restructuring activity.	

Table 4. Ontology Modularization Filling Card

The tasks for carrying out the ontology modularization activity can be seen in Figure 7.

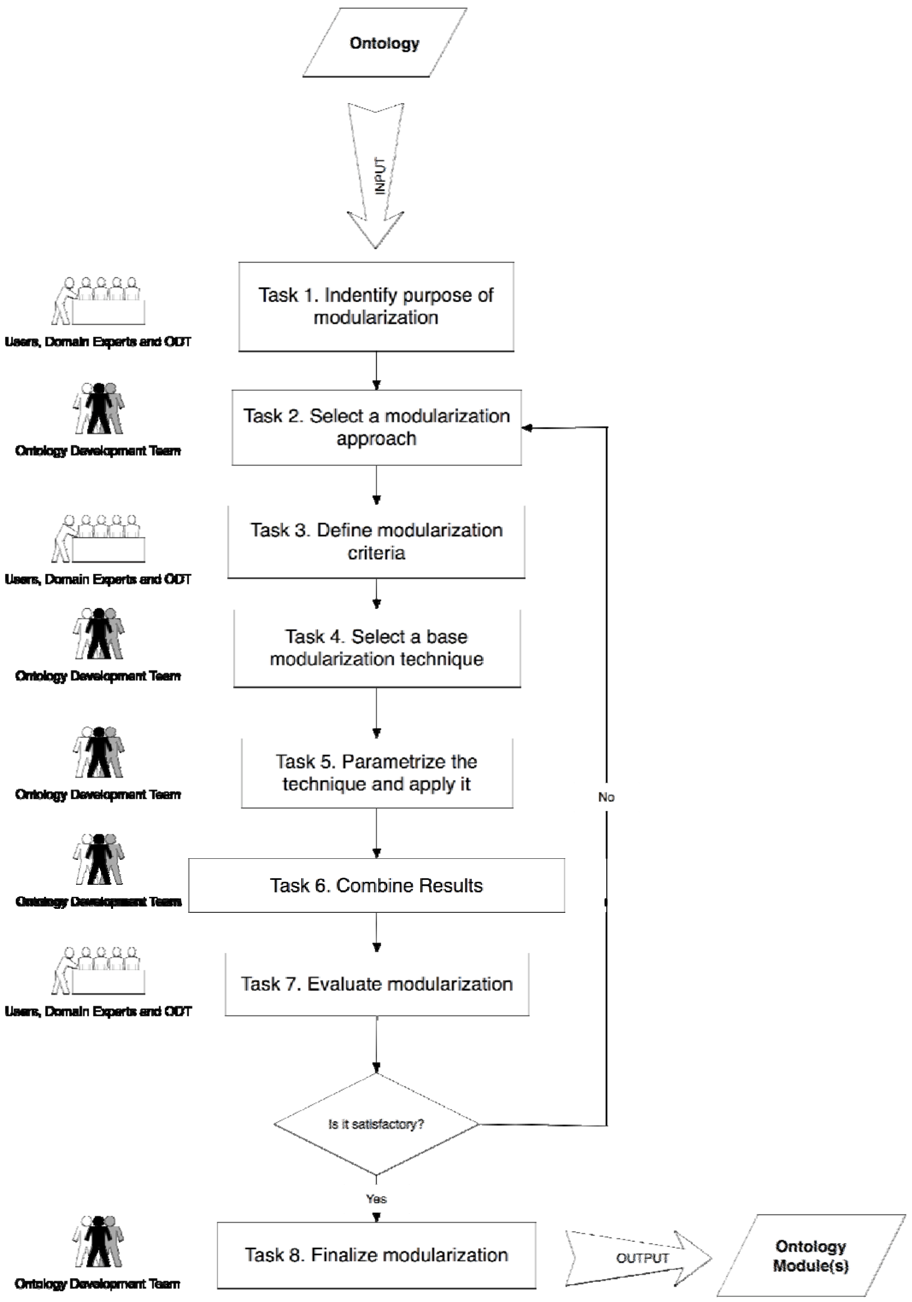

Ontology
INPUT
Users, Domain Experts and ODT
Task 1. Indentify purpose of modularization
Ontology Development Team
Task 2. Select a modularization approach
Users, Domain Experts and ODT
Task 3. Define modularization criteria
Ontology Development Team
Task 4. Select a base modularization technique
Ontology Development Team
Task 5. Parametrize the technique and apply it
No
Ontology Development Team
Task 6. Combine Results
Users, Domain Experts and ODT
Task 7. Evaluate modularization
Is it satisfactory?
Yes
Ontology Development Team
Task 8. Finalize modularization
OUTPUT
Ontology Module(s)

Figure 7. Tasks for the Ontology Modularization

The tasks for carrying out the ontology modularization activity are explained in the following:

Task 1. Identify purpose of modularization.

The goal of this task is to make explicit the reason why the considered ontology should be modularized. In other terms, the outcome of this task is a clear description of the application scenario in which modularization and ontology modules are used, as well as the expected benefit of the modular approach. D1.1.3 [21] describes some typical use case for modularization and concrete examples from NeOn.

Commonly considered benefits (and thus drivers) of ontology modularization are:

- **Improving performance** by enabling the distribution of reasoning or by exploiting only the relevant modules of a large ontology (see [20] for an example in inference justification).

- **Facilitating the development and maintenance of the ontology** by dividing it in loosely coupled, self-contained components, which can be managed separately.
- **Facilitating the reuse of (parts of) the ontology**, by extracting modules of the ontology that have a specific application or purpose for being reuse.
- **Customizing ontologies**, by application developers to flexibly extract and combine modules relevant to a particular application, or to provide different modules to different groups of users (see [48] for an example in managing access rights in a distributed question answering system).

Identifying the purpose of modularization is essential for the next tasks, in particular to select the appropriate modularization technique and criteria to maximize the expected benefit of modularization.

Task 2. Select a modularization approach.

We distinguish two different types of modularization: ontology partitioning and ontology module extraction. It is generally easy to decide which one to choose according the modularization purpose:

- Whenever the purpose relates to the entire ontology (i.e., improving maintenance, and in some cases performance), a partitioning approach should be considered.
- Whenever the purpose relates to extracting specific parts of an ontology (e.g., to customize it or reuse it partially), module extraction should be considered.

However, considering that the present methodological guideline favours an iterative approach, it can happen that the two approaches can be combined, extracting, for example, modules from the result of a partitioning technique.

Task 3. Define modularization criteria.

The modularization criteria define the basic characteristics that the resulting modules should have, i.e., what should go into a module. In [19] a set of criteria typically employed for modularization is given (e.g., logical completeness and correctness with respect to the original ontology, size, relation between modules, etc.) The criteria to emphasize should be decided depending on the purpose of modularization. For example, if the goal is to improve the reasoning procedure, logical criteria should be favoured. Authors in [23] show that with the great variety of techniques for modularization all implement different criteria, meaning that this task is essential for choosing the appropriate technique, or combination of techniques. Unfortunately, while work in [19] provides a list of common criteria, and insights on their importance in different scenarios, the choice of the right criteria to apply is highly dependent on a particular situation and has to be left to the ontology engineers to decide.

Task 4. Select a base modularization technique.

As mentioned in Section 5.1, there is a great variety of techniques and tools for ontology modularization. In [23, 19] we showed that these techniques implement a *different intuition* about what should be in a module, and so, there is no universal definition of what an ontology module should contain. In other words, it is necessary to select the most appropriate technique; depending on the criteria to apply. There is currently no comprehensive list of techniques that could be applied for modularization. However, authors in [19] provide a complete description of the major techniques and experiment demonstrating how they realize some possible criteria.

Task 5. Parameterize the technique and apply it.

Depending on the technique that has been selected by the previous task, there may be various parameters required to obtain interesting and useful results. For example, module extraction techniques generally require identifying a sub-vocabulary of the original ontology, defining a particular area of interest. Partitioning techniques may require indications, for example, about the minimal/maximal size of a module (like it is the case of the NeOn Toolkit plug-in for ontology partitioning [20]). In this case, the ontology engineer can only refer to guidelines and manual of the individual tool to establish the best parameters in his/her context. Most of the techniques would, in principle, be applied in the same way, taking the original ontology as input and creating modules in the form of smaller ontologies, allowing in this way to process the resulting modules iteratively, in the same way as the original ontology.

Task 6. Combine results.

As mentioned earlier, we favour an iterative process where the adequate modules are produced by refining and combining the results obtained with various parameters, techniques and approaches. Therefore, at every iteration, every time a new (set of) module(s) is produced, it is necessary to integrate it—i.e. to combine it—with the modules that were produced at previous iterations. The way to combine depends on the criteria for modularization and on the modules already produced. Two possibilities are:

- If some modules were too small or not logically complete and the current iteration produced complementary modules, then the results should merged.
- If modules from a previous iteration were too big because the employed technique didn't consider some of the criteria, and a new technique is applied that implements the missing criteria, then the common part from the results of both iteration should be considered.

Operators for combining modules have been included as a plug-in in the NeOn toolkit [20], in addition to plug-ins for ontology partitioning and module extraction, in order to facilitate this task. More precisely, tools are provided to compute the intersection (i.e., the common part), the union (i.e., the fusion) and the difference (i.e., the complement) of two ontology modules. These should be applied in the following situations:

- **Intersection:** when two or more modules have been produced that are complementary in the sense that they are too broad and should be reduced in relation with each other.
- **Union:** when two or more modules have been produced that are complementary in the sense that they are too narrow and should be integrated with each other
- **Difference:** when two of more modules have been produced that are complementary in the sense that they one should be narrowed down so that it does not overlap with the other.

Task 7. Evaluate the result.

The evaluation of the result of the modularization (meaning the complete set of modules generated) is a crucial part of the iterative and interactive process we promote. Indeed, it depends on this evaluation whether a new iteration is necessary, applying a new set of criteria and a new technique, or if the current (set of) modules is satisfactory considering the application scenario. There are two ways in which the modularization could be evaluated.

- **By checking the criteria:** Evaluating whether the criteria defined for modularization have been realized as expected by the modularization technique is useful both for checking if the results match the requirements of the application, and for establishing a new set of criteria in case another iteration is required.
- **By testing against the purpose of modularization:** If the defined criteria have all been realized, it is important to check whether or not the obtain modularization actually realize the expected improvement compared to the original ontology. For example, if the goal was to facilitate the maintenance of the ontology, the ontology engineers and domain experts

should check whether the structure of the new, modular ontology has been created in a sensible way according to this purpose.

There can be 3 outcomes for this task. It can establish by evaluation that:

- **The modularization is satisfactory**, so that the created modules can be finalized and deployed (Task 8).
- **The modularization is incomplete**, so that a new iteration should be carried on, using another set of criteria and another technique to produce complementary results.
- **The modularization is improper**, so that a new iteration is required, re-considering the set of criteria and the technique to employ in order to produce modules that better match the purpose of modularization.

Note that in different iterations, only the purpose of modularization cannot change. In particular, even if the approach (extraction or partitioning) generally does not change, it is not hard to imagine scenarios in which a partitioning technique is first applied, followed by extraction procedures on the previously created modules, as showed by the example in Section 5.3.

Task 8. Finalize modularization.

Once the produced modularization has judged satisfactory, an additional step can be required for it to be deployed and exploited in an application. For example, it usually necessary to revise the identifiers of each of the modules so that they follow the conventions employed in the target application, to re-establish links between modules, or simply to deploy the resulting modules in a way that it is made accessible in the target application and the editorial workflow.

5.3. Example

In this section we include a general example of how to use the proposed guidelines for the ontology modularization activity and what results are expected.

We consider the scenario where a large monolithic ontology has been developed in the past, and this needs to be modularized in order to facilitate its maintenance. The purpose of the modularization has therefore been clearly identified (Task 1). In this case, it is clear that what is required is to produce a set of modules that together cover the entire ontology. Thus, in Task 2, the partitioning approach is selected. Considering that the purpose is to facilitate maintenance, the major criteria (Task 3) to take into account are:

- the sizes of the modules, which should be small enough to be easily manageable, but not too small so that the ontology curator does not have to handle to many different modules for a particular management task; and
- the relations between modules, that should favour a well-structured organization in the dependency of the modules

Considering both criteria above, it is decided to apply the NeOn toolkit plug-in for ontology partitioning [20], which work on the dependency graph of modules and intend to provide good structures for this dependency graph. The only parameter for this technique is the minimum size of a module (Task 4), which is chosen according to the size of the initial ontology. The resulting partition is described in the figure below. Even if there are no previous results yet, some modules produced by the partitioning technique can already be combined together (Task 6). Indeed, small modules can be judged too small and might contain information that is considered relevant for other modules. Therefore, these modules can be merged using the NeOn toolkit plug-in for module combination, and employing the *Union* operator. This is depicted for our example in Figure 8.

Now that a first result has been produced, it can be evaluated (Task 7) by the ontology development team and the domain experts. In this example, there is one module that is considered too big and covering two major topics that could be separated. A second iteration is necessary.

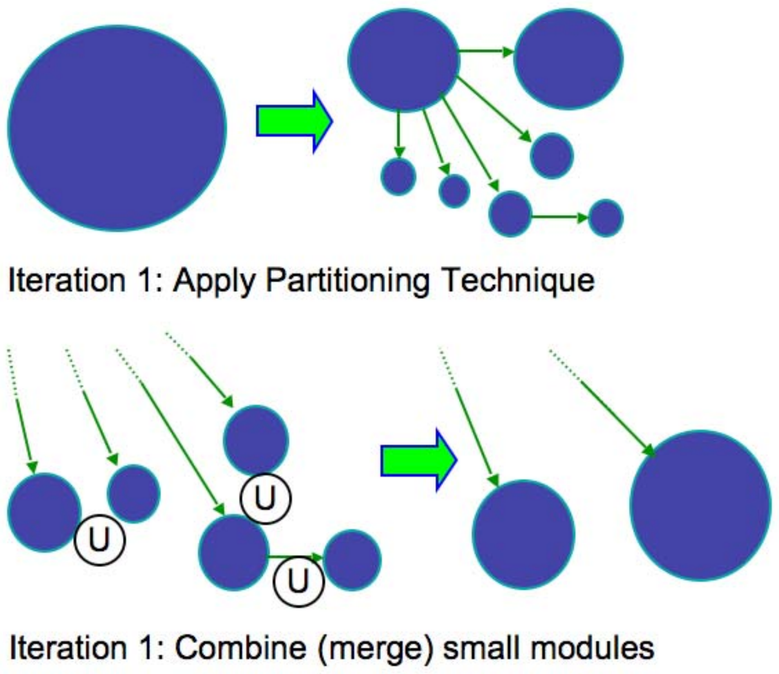
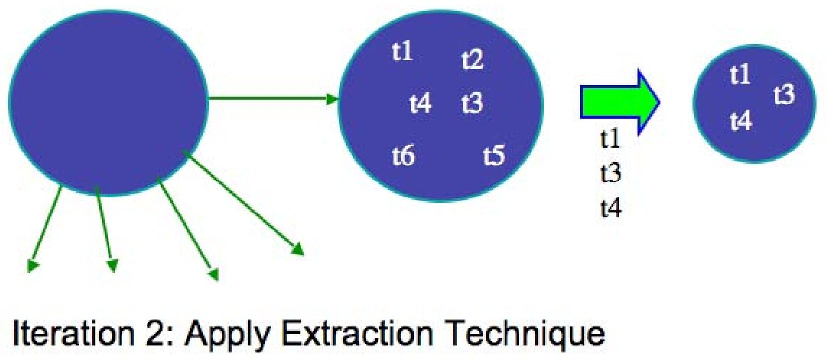

Iteration 1: Apply Partitioning Technique
U
U
U
Iteration 1: Combine (merge) small modules

Figure 8. Ontology Modularization: Result of the First Iteration

The goal of the second iteration is to extract from one of the modules produced previously, the elements related to one particular topic. Thus, we chose to follow the extraction approach (Task 2). The criteria here are mainly that the extracted module should contain ontological elements relevant to one this particular topic (Task 3). The NeOn Toolkit plug-in for module extraction [20] proposes a number of operators that can be used interactively and combine to extract modules, following the general idea of a parametric modularization as described in [22]. This tool is used (Task 4), following the rules described in [24] to generate relevant modules on the basis of a set of core terms defining the topic (Task 5). The result is depicted in the figure below. Now that one module has been extracted for one of the topic covered by the original module, the one for the second topic has to be created in the combination task (Task 6). This is achieved by using the *Difference* operator in the module combination plug-in of the NeOn Toolkit (see Figure 9). In this way, the original module has then been divided into two modules, one being the complement of the other. We then obtain a new set of modules that can be evaluated, and, if judged adequate, can replace the original, monolithic ontology.



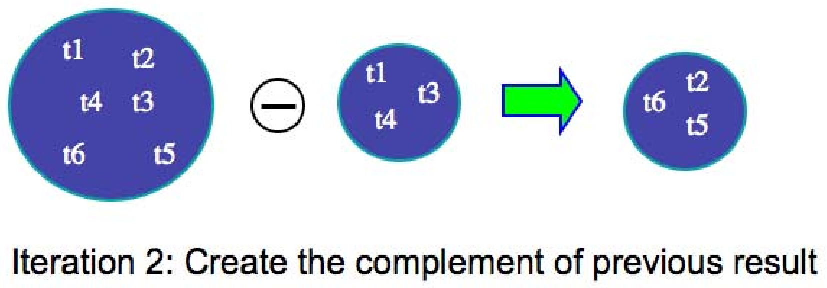


Figure 9. Ontology Modularization: Result of the Second Iteration

This abstract example provide an illustration of the overall activity of modularizing an existing ontology, using the iterative and interactive method we promote, using different modularization approaches, and combining results from different techniques. However, even with the provided tool and methodological support, modularizing an ontology is still a very time consuming task, not only because of the expensive computation it requires, but also because of the expertise and experience needed from the ontology engineer to obtain the desired result (which is very often very hard to establish). We applied the proposed guidelines and the corresponding tools successfully on a number of either small-scale or artificial examples (see e.g. [20]). In the future, we plan to realize this activity in more realistic scenarios that may require adapting the tools and the guideline to become more robust and more flexible.

6. Ontology (Network) Evaluation

Ontology Evaluation is defined in [81, 78] as the activity of checking the technical quality of an ontology against a frame of reference. We can distinguish two different types of ontology evaluation:

- ***Ontology Validation*** is the ontology evaluation activity that compares the meaning of the ontology definitions against the intended model of the world aiming to conceptualize. This activity answers the question: *are you producing the right ontology?*
- ***Ontology Verification*** is the ontology evaluation activity which compares the ontology against the ontology specification document (ontology requirements and competency questions), thus ensuring that the ontology is built correctly (in compliance with the ontology specification). This activity answers the question: *are you producing the ontology in the right way?*

In this chapter we present a brief summary of the state of the art in ontology evaluation. We also propose the NeOn methodological guidelines for carrying out the ontology network evaluation activity.

6.1. State of the Art

Ontology evaluation is an important part of the overall NeOn methodology for constructing ontology networks, and during recent years ontology evaluation has attracted a considerable amount of attention within the research community. This also indicates that there is a large amount of related work in the area. A thorough account on the state of the art in ontology evaluation, related to the NeOn project, has already been presented in NeOn deliverable D2.2.1 [70], hence this section is to be viewed more as a summary of that deliverable, adding some aspects related to methodologies (since this is the topic of this deliverable). For a complete coverage of ontology evaluation methods the reader is referred to either D2.2.1 [70] and its referenced literature or directly to collective frameworks like the O^2 ontology and oQual and the method and measurement overview in [37], or state of the art overviews like [10] and [38] as already referenced in D2.2.1 [70].

As already mentioned in D2.2.1 [70] within the NeOn project the work in [36] has been used to describe the general notion of evaluation and the specific methods relevant to the NeOn project. The work proposes several levels of formal description for characterizing and describing ontology evaluation as such and specific evaluation instances. A meta-ontology called O^2 describes ontologies as semiotic objects and describes the elements and features of the ontology that may be evaluated. Next, oQual models ontology evaluation as a diagnostics task, based on the formal description of ontologies in O^2. Based on this Quality-oriented Ontology Descriptions can be formulated, which together with descriptions of principles and parameters etc. will constitute the basis of the ontology evaluation.

One of the main merits of the above mentioned work is the overall formal framework describing ontology evaluation and the division of evaluation methods into three main categories:

- *Structural evaluations* consider the logical structure of the ontology, usually depicted as a graph of elements. Evaluating this means evaluating the ontology on a context-free syntactic and semantic level, where general quality principles may be stated depending on the aim/goal of the evaluation. At the structural level the ontology is simply regarded as an information object.
- *Functional evaluations*. Evaluating the functional dimension means to evaluate the ontology in its usage context, so to evaluate the conceptualization against its requirements and goals

to be achieved by the ontology. At the functional level the ontology is viewed as an information object and its intended conceptualization.

- *Usability evaluations* focus on the pragmatics of the ontology and the context in which it is communicated. At this level the ontology is viewed as a semiotic object, trying to convey some meaning in a context. These dimensions are not methods in themselves, but merely categories into which specific methods may be classified.

6.1.1. Structural evaluations

Structural evaluations focus as mentioned above on syntax and formal semantics of the ontology. In [37] a comprehensive overview is given with respect to currently available structural measures applicable when evaluating ontologies. Such measures for example include: depth, breadth, tangledness, fan-outness, ratio of shared differentiating notion, density, modularity, logical adequacy, and degree distribution.

Some are quite intuitive, like studying different variants of the taxonomic depth, breadth, tangledness and fan-outness of the ontology structure. Density, modularity and degree distribution concern the distribution of relations within the ontology, while logical adequacy is concerned with the formal definitions of the ontology and can measure for example the generic language complexity, inverse relations ratio or axiom class ratio. The ratio of differentiating notions is concerned with the rationale behind the taxonomy, and how the classes were divided into sets of siblings.

6.1.2. Functional evaluations

Functional evaluations are the main focus of D2.2.1 [70], whereby only a very brief summary is presented here. Functional evaluations focus as mentioned above on the usage of the ontology, how well it matches the intended conceptualization. Again we may find a good overview in [37]. Such measures for example include: expert agreement, user satisfaction, task assessment, and topic assessment.

Expert agreement can be the basis of applying precision and recall-like measures for ontology evaluation. Task assessment evaluated the ontology with respect to its appropriateness for the intended task, for example using competency questions. Topic based assessments measures the fitness of the ontology with respect to some existing repository of knowledge on the same topic. User satisfaction can be measures in terms of for example user ratings or polls. In D2.2.1 [70] two specific methods are discussed, first gold standard-based evaluations where the ontology is compared to an “ideal” ontology (the gold standard of that intended conceptualization) and then Open Rating Systems for collecting user satisfaction.

6.1.3. Usability evaluations

Usability evaluations focus on the pragmatics of the ontologies, how well they convey the intended conceptualization. These measures are related to the ontology profile, which in terms of [37] is the metadata of the ontology, the information about the ontology itself and its elements. Three levels of usability are defined in [37]:

- *Recognition* is mainly about the annotations of the ontology that should convey the structure, functions and state of the ontology.
- *Efficiency* of the ontology implies to study how efficiently the ontology supports the user needs.
- *Interfacing* level is concerned with the connection to ontology-based user interfaces.

6.1.4. Conclusion

Ontology evaluation is a topic that has been treated by researchers for more than a decade; still it is not until recently that some general frameworks and categories for ontology evaluation methods have emerged. Still, numerous techniques exist for all different kinds of ontology evaluation methods (categorised as described above). The challenge is rather to select the methods that are suitable for the case at hand and the data and resources available.

Since ontology evaluation methods have already been treated in other deliverables (i.e. D2.2.1 [70]) and there exist several comprehensible state of the art overviews (see [37], [10], and [38]) we did not aim to present a detailed account of all techniques present today. Instead we have presented a general overview of the kinds of measures applicable within ontology evaluation, in order to set the stage for the proposed general guidelines for evaluation of networked ontologies presented in the rest of this chapter. Some specific methods are also presented in detail further on.

6.2. Proposed Guidelines for Ontology (Network) Evaluation

As already mentioned the goal of the Ontology Network Evaluation activity is to evaluate and compare the ontology network within a useful frame of reference and using appropriate evaluation criteria.

An ontology network is a complex structure of more or less loosely connected ontologies. These connections can have a form of meta-relations between ontologies (e.g., versions of ontologies) or mappings and alignments between pairs of individual ontology elements (concepts, properties, instances of related ontologies). Therefore, it is often practical to focus the evaluation on different aspects of the ontology network and perform the evaluation of the constituent parts of the ontology network first:

- Evaluation of constituent ontologies.
- Evaluation of mappings/alignments between pairs of ontologies.

After the relevant individual parts of the network are evaluated, the evaluation results can be combined into the overall evaluation. Alternatively, the ontology network can be evaluated as a whole within particular application scenarios for which it was designed in the first place.

In the framework of the NeOn methodology, we proposes the filling card for the ontology network evaluation activity presented in Table 5, which includes the definition, goal, input, output, who carries out the activity and when the activity should be carried out.

Ontology Network Evaluation	
Definition	
Evaluation of Ontology Networks refers to the activity of checking the technical quality of the ontology network against a frame of reference.	
Goal	
The goal is to compare the ontology network with the specification requirements and golden standards (if available) by taking into account evaluation criteria and applying various evaluation approaches (manual, automatic), yielding evaluation results and advices on how to improve the ontology network.	
Input	*Output*
A set of ontologies with interconnection links (network), evaluation criteria and a frame of reference.	Evaluation results in the form of quantitative and qualitative measures, and informal advices on the possible ontology network modifications.
Who	
Domain experts and users, also ontology developers and practitioners, who form the network ontology development team.	
When	
This activity should be carried out in parallel with the ontology network development and evolution, and after parts of the ontology network are (at least partially, as prototypes) implemented.	

Table 5. Ontology Network Evaluation Filling Card

The tasks for carrying out the ontology network evaluation activity can be seen in Figure 10.

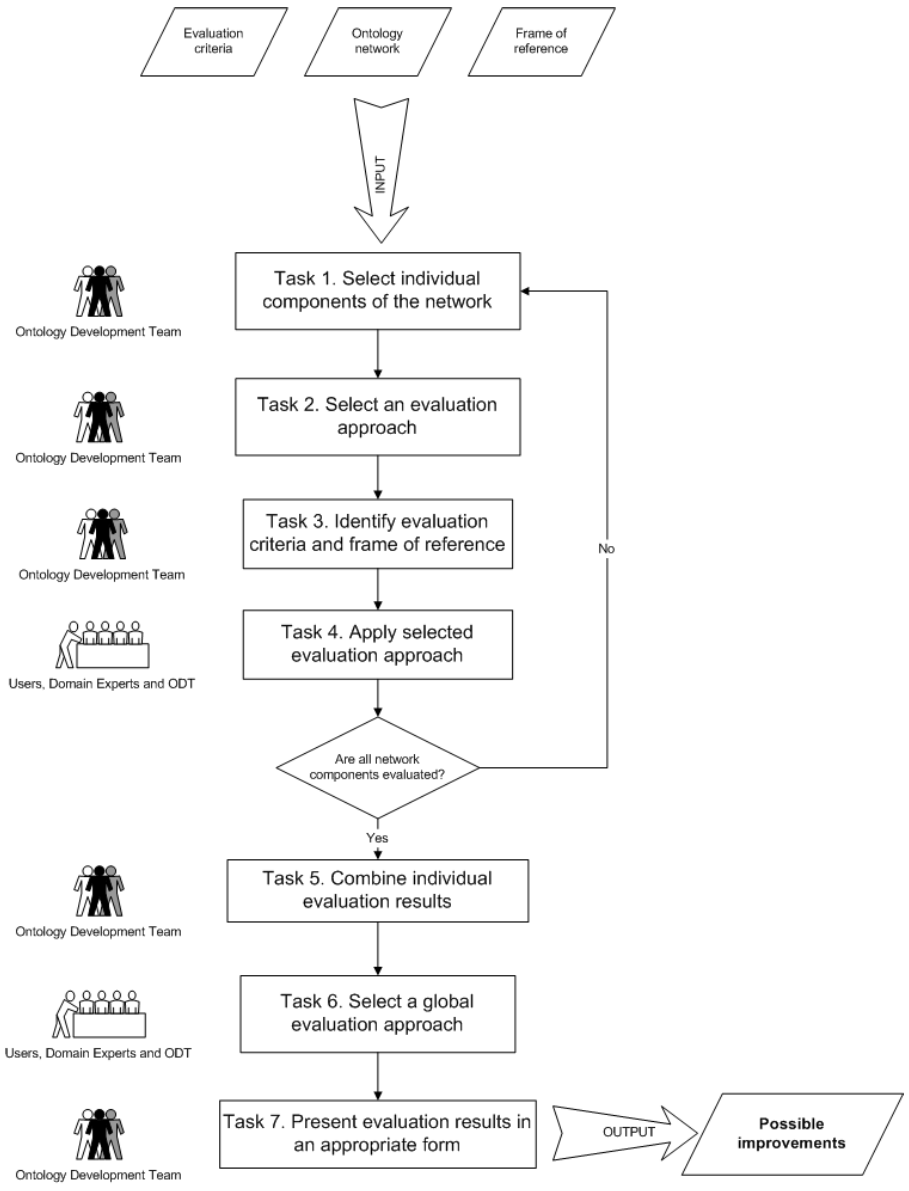

Evaluation criteria
Ontology network
Frame of reference
INPUT
Ontology Development Team
Task 1. Select individual components of the network
Ontology Development Team
Task 2. Select an evaluation approach
Ontology Development Team
Task 3. Identify evaluation criteria and frame of reference
Users, Domain Experts and ODT
Task 4. Apply selected evaluation approach
Are all network components evaluated?
No
Yes
Ontology Development Team
Task 5. Combine individual evaluation results
Users, Domain Experts and ODT
Task 6. Select a global evaluation approach
Ontology Development Team
Task 7. Present evaluation results in an appropriate form
OUTPUT
Possible improvements

Figure 10. Workflow for the Ontology Network Evaluation Activity

The tasks for carrying out the ontology network evaluation activity are explained in the following:

Task 1. Select individual components of the ontology network.

The objective of this task is to identify individual ontologies within the network, or pairs of related ontologies and the mappings/alignments between them.

The individual components of the ontology network are selected by the ontology development team based on two criteria: (1) which ontologies and mappings are critical for the overall network, and (2) which can actually be evaluated. The latter means that there must exist some frame of reference against which these individual components can be, at least in principle, evaluated.

Task 2. Select an evaluation approach.

Once the components of the ontology network are selected, one has to specify the corresponding evaluation approach.

In the case of evaluating individual ontologies, the most common evaluation approaches are [10]:

- Comparison of the ontology to the "gold standard" [51].
- Using the ontology in an application and evaluating the results [66].
- Comparison of the ontology with a source of data about the domain to be covered (e.g., a set of documents) [12].
- Evaluation by human experts who assess how the ontology meets the requirements [49].

The selection really depends on the availability of corresponding frame of reference and/or human experts.

In addition to these, existing common methods, more recently also methods for pattern-based evaluations, have emerged. Ontology design patterns (see D2.5.1 [67] for details and definitions) are encoded best practices, ranging from detailed solutions of modeling issues to general guidelines such as naming conventions. In this sense, ontology design patterns (OPs) can also be used for ontology evaluation. The three evaluation dimensions (as mentioned in Section 6.1) are covered by different kinds of patterns. Pattern-based evaluation is then carried out through checking the ontology to be evaluated against the issues and solutions represented by the patterns.

In the case of evaluating mappings and alignments between pairs of ontologies, there are typically three evaluation approaches [13]:

- Open evaluation is made with already published reference alignments.
- Blind evaluation is made by evaluators from reference alignments, unknown to the methods used to compute the alignments.
- Consensual evaluation, when there is no "golden standard" reference alignment, is obtained by reaching consensus over the results found by different methods.

Again, the selection of the evaluation approach depends on the availability of the reference alignments. No guidelines can replace the insight that the ontology network developers and domain experts have with respect to the "gold standard" of intended model of the world. The guidelines just offer different techniques that the evaluators can use for the evaluation task and some examples where different selections were made.

The examples in Section 6.3 show some typical applications of the above evaluation approaches in two practical cases.

Task 3. Identify evaluation criteria and frame of reference.

Depending on the evaluation approach selected, a frame of reference (or "golden standard") has to be specified and corresponding evaluation measures defined. A frame of reference can be other, existing resources available (reference ontologies and reference alignments), sources of data, from which the ontologies and mappings were derived (e.g., documents corpora), or human experts with the implicit understanding of the domain.

Evaluation criteria are various metrics which can be applied to the ontologies and mappings to be evaluated. The most common metrics are:

- Precision and recall measure, for an ontology or a mapping, (a) how many identified items are correct, and (b) how many items, that should have been identified are actually identified.
- Cost-based evaluation metrics measure the performance in terms of the costs of errors or the utility of correct identifications.
- Measure of "fit" between an ontology or a mapping, and a corpus (domain knowledge) by using vector space model of instances.
- Lexical measures compare the contents of two ontologies or mappings without considering their conceptual structure.

Task 4. Apply selected evaluation approach.

The goal of this task is to collect the selected network components, reference ontologies and mappings and to apply the evaluation approach selected. This requires proper setup for the evaluation experiments and implementation of software tools to compute the evaluation measures, and/or engage the human experts in stimulating sessions to collect their evaluations. The result is the computation of evaluation metrics, until all the selected components of the ontology network are evaluated.

Task 5. Combine individual evaluation results.

The goal of this task is to highlight the weakest spots in the ontology network by considering individual evaluation results and how they affect the rest of the network.

Task 6. Select a global evaluation approach.

The ontology network is designed with several typical application scenarios in mind. Rather then evaluating individual ontologies or mappings between pairs of ontologies, it may be more practical to see how the results of the application are affected by the use of the ontology network in question. Instead of focusing on an individual application, one may also focus on an evaluation from the point of view of the individual users or the organization that will use the ontology network [35].

Task 7. Present evaluation results.

The final task is to present the results of the evaluation in an appropriate form for possible repair (corrections and additions), improvements and future evolution of the ontology network.

6.3. Examples

In this section we include examples of how the proposed guidelines for the ontology network evaluation activity were actually used in practice and what were the results. The first example follows the guidelines for the evaluation of single ontology; the second example shows how patterns-based evaluation was performed in an individual ontology; and the third example illustrates the guidelines for evaluation of ontology mappings. These evaluation cases did not follow proposed guidelines (since they were completed earlier) but show that the proposed guidelines are general enough that current best practices are covered by them.

6.3.1. Evaluation of an individual ontology

In this example we show how one of the largest ontologies, which was derived mostly automatically, was evaluated. This should provide an example guideline on how to evaluate an individual, non-trivial ontology by selecting an evaluation approach, evaluation criteria and frame of reference, and present the evaluation results.

YAGO [83] is a large, lightweight, general-purpose ontology, automatically derived from Wikipedia and WordNet. It consists of over 1.7 million entities (individuals and concepts) and 15 million facts (ground binary relations between entities). The relations include the taxonomic hierarchy as well as around 100 semantic relations between entities. The empirically evaluated precision of YAGO is around 95%, which makes it one of the most accurate and largest (second only to DBpedia) generic ontologies available.

Individuals in YAGO are derived from the Wikipedia page titles. The class of each individual is derived from the Wikipedia category system. This defines the lower level type relation in the taxonomic hierarchy. The Wikipedia categories are linked to the WordNet synsets (each synset of WordNet is a class in YAGO). For the subClassOf relation, the hyponymy relation between the WordNet synsets is taken. Other, non-taxonomic relations are extracted from Wikipedia by exploiting redirects (means), relational categories (locatedIn, bornOnDate, etc.), some parsing (familyNameOf, givenNameOf), Wikipedia infoboxes (hasChild, hasBDP, etc.), and other techniques. There are altogether about 100 relations in YAGO.

The YAGO evaluation follows our guidelines in Tasks 2, 3, 4 and 7. The authors selected an evaluation approach by manual comparison of the ontology to the “gold standard” (Task 2). They identified precision as the evaluation criteria, and the Wikipedia source as a frame of reference (Task 3). The selected evaluation approach was applied by engaging human judges (Task 4), and eventually, the evaluation results were presented and discussed (Task 7).

Specifically, the evaluation proceeded as follows. They authors stated that they were interested in the precision of YAGO [83]. To evaluate the precision of an ontology, its facts have to be compared to some ground truths. Since there is no computer-processable ground truth of suitable extent, they had to rely on manual evaluation. They presented randomly selected facts of the ontology to human judges and asked them to assess whether the facts were correct. For each fact, judges could click "correct", "incorrect" or "don't know". Since common sense often does not suffice to judge the correctness of the YAGO facts, a snippet of the corresponding Wikipedia page was also presented to the judges. Thus, the evaluation compared YAGO against the ground truth of Wikipedia (i.e., it does not deal with the problem of Wikipedia containing some false information). On the other hand, the authors claim that it would be pointless to evaluate the portion of YAGO that stems from WordNet, because one can assume human accuracy there. 13 judges participated in the evaluation and evaluated a total number of 5200 facts (ground relations between YAGO entities).

The precision of the most precise and least precise relations is presented in the following Table. To make sure that the findings are significant, the Wilson confidence interval for α = 5% was computed. A confidence interval of 0% means that the facts have been evaluated exhaustively. The evaluation shows very high quality results. 74 heuristics have a precision of over 95%. Especially the crucial link between WordNet and Wikipedia (WordNetLinker) turned out to be very accurate. Also, the use of conceptual categories (ConceptualCategory) and infobox types (InfoboxType) to establish the type relation proved very fruitful.

Their algorithms for extraction of facts even in principle cannot achieve a precision of 100%. One reason for this is statistical: even if all of the assessed sample facts are correct (as they were indeed for many heuristics), the center of the confidence interval is lower than 100% to account for the uncertainty that is inherent in a confidence estimation.

Relation	#Eval	Precision
1 hasExpenses	46	100.0 % ± 0.0 %
2 hasInflation	25	100.0 % ± 0.0 %
3 hasLaborForce	43	97.67441% ± 0.0 %
4 during	232	97.48950% ± 1.838 %
5 ConceptualCategory	59	96.94342% ± 3.056 %
6 participatedIn	59	96.94342% ± 3.056 %
7 plays	59	96.94342% ± 3.056 %
8 establishedInYear	57	96.84294% ± 3.157 %
9 createdOn	57	96.84294% ± 3.157 %
10 originatesFrom	57	96.84294% ± 3.157 %
...		
72WordNetLinker	56	95.11911% ± 4.564 %
...		
74 InfoboxType	76	95.08927% ± 4.186 %
75 hasSuccessor	53	94.86150% ± 4.804 %

...		
88 hasGDPPPP	75	91.22189% ± 5.897 %
89 hasGini	62	91.00750% ± 6.455 %
90 discovered	84	90.98286% ± 5.702 %

Table 6. Precision of YAGO Facts

Another source of error are inconsistencies of the underlying sources. For example, for the relation *bornOnDate*, most false facts stem from erroneous Wikipedia categories (e.g. some person born in 1802 is in the Wikipedia category 1805 births). For facts with literals (such as *hasHeight*), many errors stem from a non-standard format of the numbers (giving, e.g., one movie actor the height of 1.6km, just be- cause the infobox says 1,632m instead of 1.632m). Occasionally, the data in Wikipedia was updated between the time of our extraction and the time of the evaluation. This explains many errors in *hasGDPPPP* and *hasGini*. In addition, the evaluation of an ontology is sometimes a philosophical issue, because even simple relations suffer from vagueness. For example, is Lake Victoria *locatedIn* Tanzania, if Tanzania borders the lake? Is an economist who works in France a French Economist, even if he was born in Ireland? These cases of disputability are inherent even to human-made ontologies. Thus, they authors state that one can be extremely satisfied with the evaluation results. They claim that it is difficult to compare YAGO to other information extraction approaches, because the approaches usually differ in the choice of relations and in the choice of the sources. Finally, precision can usually be varied at the cost of recall.

Evaluation of an individual ontology using a pattern-based approach

As already mention in Section 6.2, in addition to existing common methods for evaluating ontologies, now there are also methods for pattern-based evaluations. In this subsection, we show how ontology design patterns, more specifically content design patterns (CPs), are used to evaluate an ontology. The example does not cover the complete evaluation of the ontology, but presented one specific case where a CP assisted in finding potential problems and additionally suggests a way to improve the solution. The example is set within the fishery domain, and is taken from the actual evaluation of ontologies performed in the WP7 case study, and in this particular case the ontology evaluation and revision reported in NeOn D7.2.3 [98].

The ontology being evaluated is version 0.3 of the ontology referred to as 'Fishing areas' in NeOn D7.2.2 [99], modeling the division of water areas into divisions and subdivisions. Marine and inland waters are divided into regions, or 'FAO division areas'. The FAO division areas consist of major areas, divided into sub-areas, each divided into divisions and these finally into sub-divisions. Water areas have names in natural language only at the area level, while internal divisions are identified by numeric values. An example from [2] is the major area South East Pacific, which has code 87, and then one of its subdivisions has the code: 87.2.1.1. Another example is the FAO major Fishing area 51, i.e. Western Indian Ocean, and its subareas that are numbered from 1 to 8, where 1 corresponds to the Red Sea and 2 to the Persian Gulf, but where the subdivisions of these sub-areas (1-8) do not have names, only numbers to indentify them.

The ontology was examined and evaluated manually, by an ontology patterns expert. Using the available pattern catalogue within the NeOn project, more specifically as represented in the ontology design pattern portal[7], as a 'gold standard' of modeling to which the solutions were compared. CPs introduce best practices for solving particular modeling problems, but by introducing those solutions the pattern catalogue can also be seen as a catalogue of modeling issues. Hence, the patterns can be used as a list of possible issues, when trying to identify potential problems in the ontology to be evaluated.

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

In the ontology in this example a locally defined, transitive, ‘part-of’ relation was used to model the division of sub-areas and further levels of divisions and sub-divisions. In this case the available content pattern, the ‘part-of pattern’, introducing the exact same relation could have been imported and used. When performing a pattern-based evaluation this is seen as a quality indication, that a best practices solution was used (disregarding whether the actual pattern implementation was imported or not). Nevertheless, the use of this particular relation can also cause problems in certain contexts. In this case there are several steps in the division of the areas, i.e. areas have sub-areas, then divisions that in turn have sub-divisions. When using only this one transitive relation, ‘part-of’, and then reasoning with this ontology, it will not be possible to distinguish between the direct sub-parts of an area and the ones that are subparts of that area due to the transitivity of the relation. Although transitivity is a desirable property for certain reasoning applications, i.e. answering questions such as ‘is this sub-division a part of the sub-area representing the Red Sea?’, it may not be desirable in all cases. For example, if the hierarchical structure of the partitioning of the areas should be reconstructed, e.g. for browsing the ontology in a graphical interface, or when answering questions such as ‘what are the divisions of the Red Sea?’, we are perhaps not interested in all inferable parts of the Red Sea but only the direct components. For providing this option something more is needed.

Such problems are addressed by the ‘componency pattern’, also available at the ontology design pattern portal. The pattern introduces the two object properties ‘hasComponent’ and ‘isComponentOf’, which are each other’s inverses. These are non-transitive properties that can be used in combination with the ‘part-of pattern’ to both register general partitioning but in this case also the non-transitive property of a ‘proper part’, i.e. a direct component of something. When using these two patterns as ‘gold standards’ when manually examining the ‘Fishing areas’ ontology, the ontology evaluator can discover the potential problem of a missing non-transitive property to distinguish the different ‘levels’ of area decomposition. Since it is likely that applications need to be able to answer questions such as exemplified above, this is considered as a problem in the ontology. However, since this was discovered using the best practice of how to solve such a problem, i.e. the ‘componency pattern’ a solution can also be proposed, i.e. to add an additional non-transitive property possibly by importing the ‘componency pattern’.

6.3.2. Evaluation of ontology mappings

The guidelines for evaluating ontology mappings are best illustrated by an international initiative that organizes systematic evaluation of software systems which compute ontology mappings and alignments. These guidelines closely match our proposed guidelines, since they follow Tasks 2, 3 and 4, and finally present and analyse results as in Task 7.

The evaluation of mappings and alignments between ontologies within the framework of the NeOn project is described in details in deliverable D3.4.1 [45]. The main objective is to evaluate different ontology mapping algorithms on the project case studies (fisheries ontologies and pharmaceutical cases). The basis for the alignments evaluation is an ongoing Ontology Alignment Evaluation Initiative (OAEI), specifically the last one in the series, OAEI-2008 [13]. Here we give some details on the alignments evaluation in the FAO case study, and just list the rest of the test cases.

6.3.2.1. OAEI guidelines

OAEI is an international coordinated initiative that organizes the evaluation of ontology mapping and alignment systems. The main goal is to compare systems and algorithms on the same basis and to draw conclusions about the best alignment strategies. The comparison and evaluation is performed on consensus test cases. The ambition is that from such unbiased evaluations, developers can learn and improve their alignment systems.

The main phases of the OAEI evaluations are the following:

- Organizers provide pairs of ontologies.

- Participants return the alignments, computed from matching theses ontologies.
- Organizers evaluate the results with regard to reference alignments (which can be hidden or disclosed). Usually, the evaluation criteria are precision and recall, adapted to alignments.

Preparatory phase: ontologies to be matched and relevant alignments (where applicable) are provided in advance. This gives potential participants chance to send bugs and remarks to the organizers, and to propose additional test cases.

Execution phase: participants apply their systems to automatically compute alignments between the ontologies from the test cases. In most cases, the ontologies are given in OWL-DL, and the computed alignments are expected in the Alignment format, specified by the organizers.

Evaluation phase: the organizers evaluate the alignments provided by the participants. In the case of blind tests, only the organizers can compare the results with respect to the withheld reference alignments. The standard evaluation criteria are precision and recall, computed against the reference alignments, and aggregated by using weighted harmonic means (weights are the numbers of true positives). This phase corresponds to Task 2, selection of evaluation approach, and Task 3, identification of evaluation criteria and frame of reference.

6.3.2.2. Fishery case study

FAO has large amounts of data, taxonomies, thesauri and ontologies related to food production and consumptions, and also specific to fisheries. The goal of the FAO case study was to explore the possibilities of establishing alignments between some of the resources which are closely related. Specifically, the following ontologies were used:

- AGROVOC – a general thesaurus, in OWL, related to all matters of interest to FAO. For the evaluation purposes, only fragments with an overlap with other ontologies were considered (organisms, vessels, fishing gears).
- ASFA – a thesaurus dedicated to aquatic sciences and fisheries, in OWL.
- Ontologies for modelling coding systems for commodities and species - two specific ontologies in OWL with instances in three languages.

The evaluation task was to establish alignments between these ontologies:

- AGROVOC and ASFA alignments (agrasfa)
- AGROVOC and fisheries ontology about biological species (agrobio)
- The two ontologies about biological species and commodities (fishbio)

Several alignment algorithms participated in the FAO case study evaluation: Aroma, ASMOV, DSSim, Lily, RiMOM, SAMBO. These systems are described in detail in the proceedings of the Ontology matching workshop [73].

Reference alignments were obtained by randomly selecting a specific number of correspondences from each algorithm and pooling them together (this corresponds to Task 3, selection of frame of reference). This provided sample alignments, which were evaluated by FAO experts for correctness. There were two assessors: one specialized in thesauri, and another specialized in fisheries data. This corresponds to Task 4, an application of selected evaluation approach by engaging human experts.

The detailed results of evaluation are in NeOn deliverable D3.4.1 [45]. Here we show just the results for the two systems which returned usable results for agrobio and fishbio alignments. The evaluation criteria used were precision and recall, relative to the sample that has been extracted (this corresponds to Task 3, identification of evaluation criteria).

System	subtrack	retrieved |A|	Evaluated |A ∩ A0|	Correct |A ∩ R0|	RPrecision P0 (A , R0)	RRecall R0 (A, R0)
DSSim	agrasfa	218	129	70	0.54	0.31
	agrobio	339	214	151	0.71	0.97
	fishbio	243	166	79	0.48	0.60
RiMOM	agrasfa	743	194	159	0.81	0.70
	agrobio	395	219	149	0.68	0.95
	fishbio	738	217	118	0.54	0.90

From the above results it seems that fishbio is the most difficult task in terms of precision, while agrasfa is the most difficult in terms of recall. It was concluded that the RiMOM system provides the best results. The system seems especially interesting for this real-life case study, as it performed well when aligning between classes as well as when aligning between ontology instances. This is the final stage of evaluation, corresponding to Task 7, presentation and discussion of the evaluation results.

6.3.2.3. Other OAEI-2008 testing data

Apart to the FAO testing cases, there was a number of testing sets used in the OAEI 2008 evaluation:

- Anatomy – ontologies are the NCI thesaurus and the Adult Mouse Anatomical Dictionary.
- Directory – directories matching sets were constructed from Google, Yahoo and Looksmart web directories.
- Multilingual directories – provides alignment problems for multilingual (English and Japan) internet directories.
- Library – two large Dutch thesauri were to be aligned: the Scientific and Deposit collection of books.
- Very large crosslingual resources – three resources were used: GTAA – a Dutch thesaurus to index TV programs, WordNet – lexical database of English, and DBpedia – resources tied to Wikipedia articles.
- Conference – fifteen ontologies in the domain of organising conferences were to be aligned.

All the details of data used, algorithms evaluated and evaluation results are in [13].

6.3.2.4. Results

Experiments with finding alignments and mappings between ontologies indicated that no automatic procedure, i.e., algorithm can find results as good as human experts. Nevertheless, some results are very promising. If the ontologies are clearly defined and the labels for concepts and properties are sensible, some alignments algorithms can detect almost all alignments correctly.

7. Ontology Evolution

Ontology Evolution is defined in the NeOn Glossary of Processes and Activities [78] as follows: “Ontology evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency. Ontology evolution can be seen as a consequence of different activities during the development of the ontology.”

Ontologies are fundamental building blocks of the Semantic Web and are often used as the knowledge backbones of advanced information systems. As such, they need to be kept up to date in order to reflect the changes that affect the life-cycle of such systems (e.g. changes in the underlying data sets, need for new functionalities, etc).

Within their lifetime, ontologies undergo changes. They evolve, for example, to correct errors or adapt to new knowledge about the world, or changed circumstances. Moreover, during the ontology development process sometimes errors are not spotted and have to be corrected later, i.e., after initial deployment. However, as ontologies may depend on several other ontologies and may also be related to other elements (e.g. instances, mappings, applications, metadata, etc.), one has to be careful with making changes. Dealing with ontology changes usually involves the execution of many related tasks identified in the context of the ontology evolution activity. For instance, among these tasks are the capturing and formal representation of ontology changes, the verification of the ontology consistency after the changes are performed and the propagation of those changes to the ontology related entities. The definition of this activity and the study of those tasks is the subject of ontology evolution research.

Ontology evolution is described as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes [40]. While it seems necessary to apply such an activity consistently for most ontology-based systems, it is often a time-consuming and knowledge intensive activity, as it requires a knowledge engineer to identify the need for change, perform appropriate changes on the base ontology and manage its various versions.

In this chapter we present a brief introduction to the existing methods, techniques and tools for ontology evolution. We also propose the NeOn methodological guidelines for carrying out the activity. Note that for this first version, we propose high level guidelines instead of providing detailed ones. Additionally, we provide here methodological guidelines for supporting ontology engineers and domain experts in exploiting tools to facilitate the evolution of their ontologies.

7.1. State of the Art

Although the evolution of conceptual models such as schemas in databases or XML schemas has been thoroughly investigated in the computer science field, the evolution of ontologies is still under continuous research. An extensive overview on the current state of the art can be found in [65]; therefore in the following we only introduce briefly the most relevant work for this section.

We start in Section 7.1.1 by presenting some of the most well known methods for ontology evolution, followed by the presentation of typical ontology evolution techniques in Section 7.1.2. Next in Section 7.1.3 we describe some existing tools supporting ontology evolution aspects and we conclude with a brief analysis in Section 7.1.4.

7.1.1. Methods

Ontology evolution is defined by [74] as the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artefacts. The author proposes a six phase process for ontology evolution:

- The *change capturing* phase is in charge of the continual improvement of an ontology. It distinguishes two types of changes depending on the source of the change: top-down changes are those made by humans and bottom-up changes are those derived from machine learning techniques, generated to repair or evolve knowledge automatically.
- The *change representation* phase is in charge of representing requests for changes formally and explicitly. The representation of ontology changes makes them machine-understandable, usable by other ontology evolution systems as well as exploitable for supplementary functionality of an ontology evolution system.
- The *semantics of change* phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into another consistent state.
- In the *change propagation* phase all of the ontology dependent artefacts are updated.
- The role of the *change implementation* phase is to inform the ontology engineer about all consequences of a change request, to apply all the (required and derived) changes to the ontology in a transactional manner and to keep track about performed changes.
- Finally, the *change validation* phase enables justification of performed changes and undoing them at user's request.

Another approach for modelling the ontology evolution process is presented in [25]. It is based on the work proposed in [6] which distinguishes four generic activities in the process of making a change:

- *Requesting a change* initiates the change process and includes activities related with the representation of changes, prioritization of multiple changes and the discovery of changes.
- *Planning the change* deals with understanding *why* the change needs to be made, and *where* the change needs to be made. As part of this activity an analysis of the change impact is performed that identifies all the potential consequences (side effects) of a change along with an estimation of what needs to be modified to accomplish a change. Additionally, during this activity the engineer is presented with an estimated cost of evolution to allow him to decide whether or not to implement the change.
- The *implementation of the change* refers to the activities of change propagation, restructuring the ontology before the implementation of the change and inconsistency management.
- Finally, the *verification and validation* phase addresses the issues of building the right ontology and build it in a right way (i.e. quality assurance assessment).

7.1.2. Techniques

Even though the previous ontology evolution approaches model the process in a different way (i.e. using different names and number of steps), they identify very similar activities. We can summarize that change representation is fundamental, as well as the activities related to the discovery of ontology changes, the potential side effects of those changes and the verification of the ontology consistency after applying them. Finally, besides the activity of the actual implementation of the changes, some of the approaches introduce the activity of the propagation of changes. For each activity, specific methods and techniques have been proposed in the past. For instance, there are some approaches for the formal and explicit representation of ontology changes. Much of the current work focuses on devising taxonomies of elementary change operators that are sound and complete. In [74], the author describes an evolution ontology that is used as a backbone for creating evolution logs. This ontology models what changes are performed in an ontology, along with information such as why, when, by whom and how. The structure of the hierarchy of ontology changes is based on the KAON ontology model. The author introduces three levels of changes: elementary, composite and complex. In [42], the author describes an ontology for representing ontology changes for the OWL ontology language. Changes are classified as atomic or composite. They also define a class that represents a set of change operations that has a source and target ontology. Finally, the ontology includes properties that specify arguments for a change operation as well as a property *effect* to annotate the class of operations with the effect of the change.

According to [74] the discovery of changes can be done using top-down or bottom-up techniques. In the first, experts explicitly requests changes while in the second, the changes are discovered using machine learning techniques. Similarly, user-driven changes are discovered from certain usage patterns emerged over a period of time. In [43] examples of such patterns include querying and browsing behavior.

The management of consistency has also been addressed in the past. For instance, [74] and [26] adopted or extended the work in data schema evolution presented in [4], where semantics for each schema change is determined by first identifying a set of invariant properties intrinsic to the model, ensuring semantic integrity. The invariants strictly depend on the underlying model. So for each change, different alternatives are evaluated to preserve the invariants properties. In a related work, [40] the authors present an approach to localize inconsistencies based on the notion of a minimal inconsistent sub-ontology.

Finally, regarding the propagation of changes, in [74], the author discusses the propagation of ontology changes to three different types of ontology related artefacts: related ontologies, ontology instances and applications. The author further elaborates only the propagation of ontology changes to related ontologies. A related work is presented in [59] where the authors present an approach for the management of changes that comprises the ability to make copied (and possibly changed) versions of controlled vocabularies (e.g. ontologies) up to date with a remotely changed controlled vocabulary. Another approach is presented in [84], where the authors describe the management of the dependency between component ontologies in an ontology as a whole (i.e. an ontology is divided into several components ontologies). They consider two kind of dependencies between component ontologies (i.e. is-a relation and referring-to relation), the influence of a change of one ontology to others through its dependencies and finally they design a function to suggest a few candidate modifications of the influenced ontology for keeping the consistency (i.e. propagate changes). The approach considers that components ontologies can be distributed and changed locally, but the management of the dependencies is done centralized.

7.1.3. Tools

There are several tools for ontologies that address some aspects of ontology evolution. One example is the Karlsruhe Ontology and Semantic Web framework (KAON)[8] [74, 97]. The KAON ontology evolution support is realized at the API level and includes the following functionalities:

- ❑ Evolution logging is responsible for keeping track of the ontology changes in an evolution log.
- ❑ Change reversibility enables undoing and redoing changes made in an ontology.
- ❑ Evolution strategy is responsible for ensuring that all changes applied to the ontology leave the ontology in a consistent state and for preventing illegal changes. Also, the evolution strategy allows the user to customize the evolution process.

[8] http://kaon.semanticweb.org/

- Evolution graph enables ontology engineers to enhance a set of changes with their own changes and to resolve them.
- Ontology inclusion facilities, together with the dependent evolution, are responsible for managing multiple ontologies within one node (i.e. change propagation).
- Ontology replication facilities, together with the distributed evolution, are responsible for enabling the reuse and the management of distributed ontologies (i.e. change propagation).
- Change discovery includes the means for the discovery of problems in an ontology and for making recommendation for their resolution.
- Usage logging is responsible for keeping track of the end-users interactions with ontology-based applications in order to adapt ontologies to the users' needs.

In [42] the author introduces the tool OntoView, a web-based change management system for ontologies. The system aims at helping users to manage changes in ontologies and keeping ontology versions as much interoperable as possible. OntoView, inspired by the Concurrent Versioning System CVS, provides an interface to different versions of ontologies, by maintaining not only the transformations between them, but also the conceptual relation between concepts in different versions.

A similar tool support called PROMPTdiff was presented by the same authors in [58]. PROMPTdiff is an ontology-versioning system implemented as a Protégé plug-in that compares two different versions of an ontology (similar also to the SemVersion system presented in [87]) producing a structural diff between ontologies (i.e. mappings).

Additionally, in [52] the Evolva ontology evolution framework is proposed covering a complete ontology evolution cycle. Evolva intends to be a *blueprint* for ontology evolution systems, relying on the hypothesis that various forms of data sources (texts, folksonomies, etc.) can be used to detect the need for an evolution, and initiate it (see Figure 11). Evolva also relies on the idea that, in order to integrate new pieces of information extracted from the exploited sources into the current ontology, evolution systems can rely on the automated use of external background knowledge, such as other ontologies, lexical resources (e.g., WordNet [32]) or the Web.

While the goal of the Evolva framework is to reduce, as much as possible, human intervention within the evolution process, user input is required at the level of evolution management, and for fine-tuning various parts of the framework. The role of the user is needed, to properly parameterize the components, select the right sources of information and of background knowledge, validate the results of various steps and, generally, guide the evolution process to obtain high quality results. These tasks are not trivial, as they depend a lot on the particular ontology to be evolved, the domain covered, the applications relying on the ontology and the reasons for its evolution. The experience of the knowledge engineer and his/her knowledge of the ontology and of the exploitable sources of information are therefore essential.

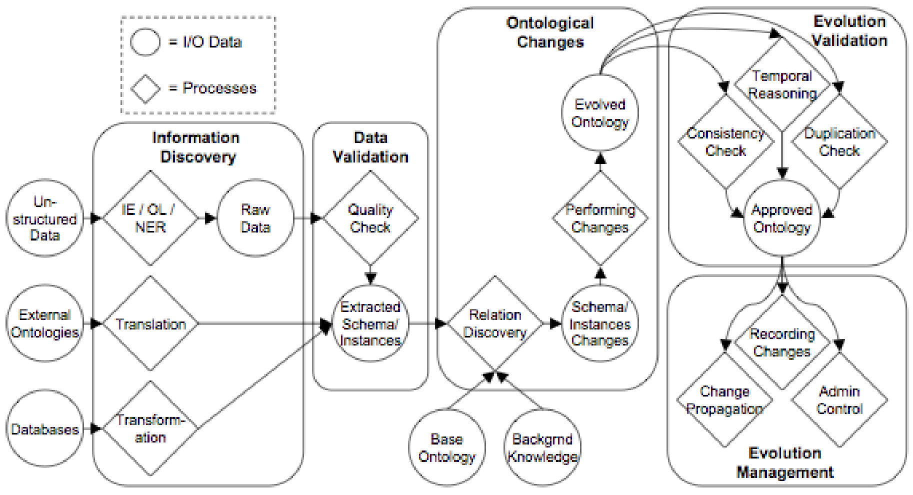


Figure 11. Evolva Ontology Evolution Framework

7.1.4. Conclusion

Summarizing, several research efforts have addressed various phases of the ontology evolution process. A first category of approaches [42, 56, 74, 89] are concerned with formalisms for representing changes and for facilitating the versioning process. Another category of work [1, 7, 55, 60] aims to identify potential novel information that should be added to the ontology. They do this primarily by exploiting the changes occurring in the various data sources underlying an information system (e.g. databases, text corpora, etc), or by interpreting trends in the behaviour of the users of the system [1, 7]. A few systems from this category also investigate methods that propose appropriate changes to an ontology given a piece of novel information. These methods typically rely on agent negotiation/multi-agent systems [55, 60] to propose concrete changes that then are verified by the ontology curator. However, the management side of evolution does not occupy a substantial part of their process. Thus, there is no existing approach covering a complete cycle of ontology evolution, ranging from integrating new knowledge to managing changes.

A large number of the ontology evolution activities (e.g. change propagation, change representation, versioning approaches, etc.) have been thoroughly investigated in many other computer science fields, however for our domain (i.e. ontologies), they are still under active research. For instance, regarding the representation of changes, we can summarize that much of the current work focuses on devising taxonomies of change operations that classify them as elementary (atomic) or composite, and some approaches also consider complex type of changes. Even though the proposed elementary (atomic) changes are introduced as operations that cannot be subdivided into smaller operations, they are in all cases considering changes at the entity level (i.e. concepts, properties, individuals) , and in some cases they are not even minimal (e.g. include modify operations). Furthermore, existing approaches are dependent on the underlying ontology model (i.e. they are based on proprietary models (e.g. KAON) or for specific languages (e.g. OWL) and consequently they have different set of elementary (atomic) changes. With respect to the propagation of ontology changes still much can be learned from other related fields. For instance, the propagation of changes has been investigated in particular to dependent ontologies, ontology instances and in a few cases to dependent applications. In the case that the dependent ontologies are distributed, in general they address one of the following cases:

- An ontology is locally edited by a single user and changes to that ontology are propagated to physically distributed replicas of the ontology
- A shared-central copy of an ontology is shared by several users. Each user has a local copy of this ontology (maybe even modified), that may (or not) be updated (synchronized) whenever then shared-central copy is changed centralized by a single user.

- An ontology is divided into several (distributed) component ontologies related to each other. Each component ontology is then treated as a normal ontology that is locally edited by a single user and changes are propagated to related ontology components in a centralized manner when the user publish the component ontology into a central server.

Hence, in all cases, they are considering a central (main) copy of the ontology that is either replicated or divided into several component ontologies. Moreover, in most of the cases, changes are propagated only in one direction: from the main copy to its replicas. The only exception is when the ontology is divided into several component ontologies, but in this case the management of changes is also centralized.

Similarly, the existing tools support different aspects of the change management. Many of them are focused on the management of ontology versions rather than in support the tasks of the ontology evolution process that are our main interest. Nevertheless, some tools address evolution aspects such as the effect of changes in dependent applications or, the tracking of changes and their formal representation and/or the propagation task. However in those cases, the management of ontologies and their changes is either centralized or changes are propagated from a main copy of the ontology to its distributed non-editable replicas.

Finally, there are several aspects regarding ontology evolution that are still not fully addressed in the existing models/tools. For instance, in practice the evolution of the ontologies is no longer driven by a single user. Currently a whole team of ontology editors is in charge of the development and maintenance of the ontology. Each member of the team may have different role and associated permissions and usually they follow a well defined process to control when and how the ontology can change (evolve). Furthermore, the members of the team are usually geographically distributed.

The aspects mentioned above are the main reason why methodological guidelines for ontology evolution are needed in the context of the NeOn methodology for building ontology networks. While existing evolution frameworks normally include a description of the life cycle, this description is neither meant nor suited to replace guidelines. Therefore we propose here methodological guidelines for supporting ontology developers during the evolution of the ontologies and for supporting them in exploiting tools to facilitate the evolution of their ontologies.

7.2. Proposed Preliminary Guidelines for Ontology Evolution

As we mentioned before, ontology evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency. Ontology evolution can be seen as a consequence of different activities during the development of the ontology.

In the framework of the NeOn methodology we propose the filling card for the ontology evolution, presented in Table 7, which includes the definition, goal, input, output, who carries out the activity and when the activity should be carried out.

Ontology Evolution	
Definition	
Ontology evolution refers to the activity of facilitating the modification of an ontology by preserving its consistency. Ontology evolution can be seen as a consequence of different activities during the development of the ontology.	
Goal	
The goal of ontology evolution is to provide a defined process (potentially with tool support) to perform updates and changes to one or multiple ontologies.	
Input	*Output*
An ontology in a consistent state.	An ontology in a consistent state with proposed changes implemented.
Who	
All ontology engineers that have to perform changes/updates to a deployed ontology.	
When	
Ontology evolution normally occurs after the ontology is deployed. Changes during the initial creation would be part of the initial ontology engineering process. Once an ontology has been deployed and needs to be updated/changes, ontology evolution is needed.	

Table 7. Ontology Evolution Filling Card

The tasks for carrying out the ontology evolution activity can be seen in Figure 9.

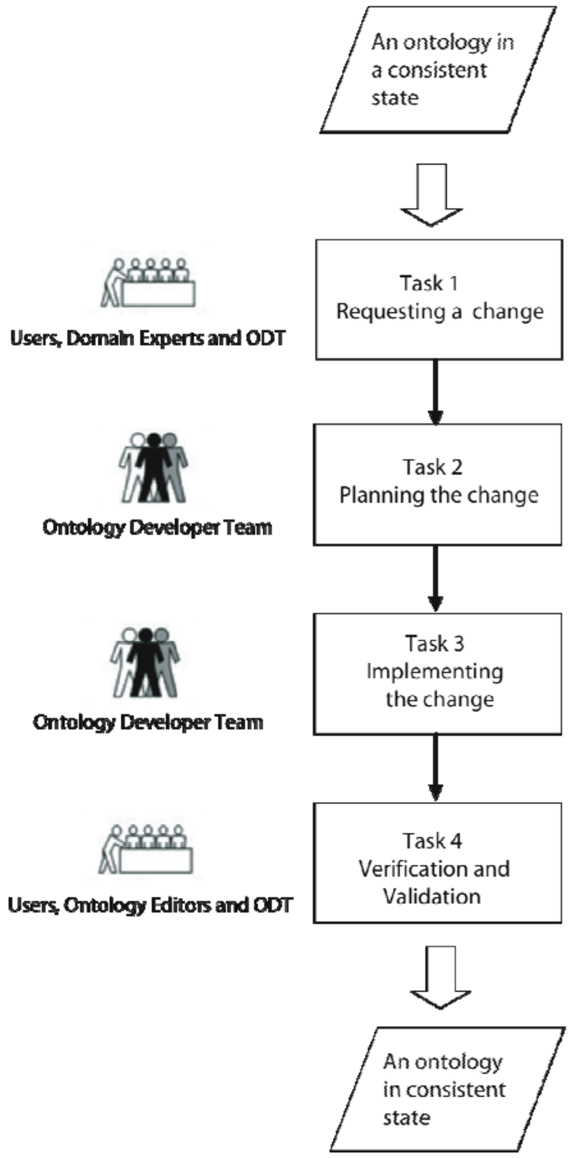

An ontology in a consistent state
Users, Domain Experts and ODT
Task 1 Requesting a change
Ontology Developer Team
Task 2 Planning the change
Ontology Developer Team
Task 3 Implementing the change
Users, Ontology Editors and ODT
Task 4 Verification and Validation
An ontology in consistent state

Figure 12. Workflow for the Ontology Evolution Activity

The tasks for carrying out the ontology evolution activity are explained in detail in the following:

Task 1. Requesting a change.

This is the initial task in the evolution of an ontology. In order for ontology evolution to have the desired outcome, it is important that the input ontology is in a consistent state. If the ontology is not in a consistent state, it has to be repaired first, using one of the different ontology diagnosis and repair tools (e.g. RADON[9]) or techniques before starting the evolution process. Note that we require the input ontology to be in a consistent state because dealing with an inconsistent ontology may produce unexpected results. For instance the propagation of changes may produce inconsistencies in related artificats. This requirement is also in accordance to existing ontology evolution approaches (e.g. [74]). Besides, the main goal of ontology evolution is to adapt an ontology to arisen needs (e.g. changes in the domain, changes in the experts knowledge, etc.), not to repair an inconsistent ontology. Therefore, the input of the evolution process is an ontology that

[9] http://radon.ontoware.org/index.html

correctly modelled a particular domain/task, before new needs arise. However, the repairing of an inconsistent ontology before starting the ontology evolution process can be seen as a one pre-processing task. The first step of this task is basically initiating the change process. Changes can either be requested from users or developers, who feel that the ontology is not adequate in its current form, or changes can be discovered. In literature [74] change discovery is distinguished into top-down and bottom-up change discovery. Top-down (deductive/explicit) changes are usually the results from knowledge elicitation techniques that are used to acquire knowledge directly from human experts (e.g. domain experts or end users). Bottom-up changes are the result from machine learning techniques, which use different methods to infer patterns from the sets of examples (e.g. Structure/Data/Usage-driven Change Discovery).

Once changes are discovered or requested, they have to be represented in a formal and explicit way. Typically, a change ontology is used to model proposed/requested changes (e.g. [74, 42, 56]). This formal representation of ontology changes makes them machine-understandable, which supports and facilitates many evolution activities: their propagation to ontology related entities, the synchronization of distributed copies of the same ontology, their integration with information related to the process of the ontology development (e.g. accept/reject changes), the identification of conflicts, etc. Moreover, having changes formally represented makes them usable by other ontology evolution systems as well as exploitable for supplementary functionality of an ontology evolution system such as learnability. Finally, it allows to keep track of the ontology changes by generating a log that maintains the history (and order) of applied changes as a sequence of individuals of the proposed model.

In contrast to previous approaches in the literature, in NeOn Deliverables D1.3.1 [65] and D1.3.2 [64], a layered approach for the representation of ontology changes was presented, which consists of a generic ontology, independent of the underlying ontology model that models generic operations in a taxonomy of changes that are expected to be supported by any ontology language. Furthermore, the model can be specialized for different ontology languages, allowing the reuse and refinement for specific needs. Also, the model extends previous taxonomies of changes with a more granular classification that considers the actual “atomic changes” that can be performed in an ontology.

In case there are multiple change requests for an ontology, the requested changes have to be prioritized. In order to determine which change should be implemented first, one can rely on the status of the person requesting the change, or have an ontology engineer review the requested changes and rank them according to urgency. It is also important that dependencies are considered when ranking the requested changes. It could be that changes are dependent on each other or even contradict each other.

Finally, according to NeOn Deliverables D1.3.1 [65] and D1.3.2 [64] (see also [62]), this task includes (in some scenarios) the use of a well defined process (i.e. a workflow) for coordinating change proposals. This process is responsible for determining who (depending on the user permissions) can do what (i.e. what kind of actions) and when (depending on the state of the ontology element (e.g. classes, properties and individuals) and the permissions of the user).

As NeOn technology supporting this task we can mention the following:

- RADON Plug-in[10] is an ontology diagnosis and repair tool that can be used before starting the evolution activity.
- Evolva Plug-in[11] supports the discovery of changes from external data sources (e.g. text or folksonomies) and relies on various background knowledge sources to automatically link new knowledge to the ontology.

[10] http://www.neon-toolkit.org/wiki/index.php/RaDON

[11] http://evolva.kmi.open.ac.uk/

- The workflow Feature[12] supports the process to coordinate the proposal of changes.
- Finally, the change ontology used within NeOn for the representation of changes is described in NeOn Deliverable D1.3.1 [65].

Task 2. Planning the change.

In this task the change request is analyzed and it is determined why the change needs to be made and which part of the ontology is affected by the change.

For that purpose one uses Impact Analysis, where all potential consequences (side effects) of a change are identified along with an estimation of what needs to be modified to accomplish a change [91]. As we noted in the introduction, ontologies may depend on several others and may also be related to other artefacts (e.g. instances, mappings, applications, metadata, etc.). Hence, for the analysis of the impact of a change, a complete list of all implications to the ontology and its dependent artefacts should be presented to the ontology engineer [92].

The previous analysis is also helpful to estimate the cost of evolution. Based on this cost the ontology engineer can decide whether or not to propagate a change to a dependent artefact [92].

As a result of the analysis performed during this task, the ontology engineer may decide to implement the change, or if the change has many side-effects or if the cost of implementation is too high, he may defer the change request to a later time or not implement it at all.

Once the ontology developer team has decided which changes will be implemented and how they have to be implemented, the next phase of ontology evolution, namely change implementation is entered.

As NeOn technology supporting this task we can mention that the NeOn toolkit[13] provides a simple support for the decision of making a change or not. In particular, when a user wants to delete an ontology element the list of related axioms (the side-effect) is shown to the editor, allowing him to verify the cost of implementing the change.

Task 3. Implementing the change.

Implementing the changes is of varying difficulty, depending on the impact of the requested change. While some change can be as easy as adding or removing a subclass, other changes can require complex operations and restructuring of the ontology.

One of the first and foremost important features is change logging, which allows to track which changes have been made, and also allows for an easy undo, in case something goes wrong. The change log can also be published to inform people using the ontology on the updates.

If the requested change turns out to be too difficult to be implemented, the ontology may need to be restructured first, before the actual desired change can be implemented [93]. Depending on the complexity of the task, an ontology engineer can be chosen to perform the restructuring and the subsequent implementation of the changes. For instance, in [94] the authors distinguish three reasons to apply transformation: (i) to select an alternative conceptual schema which is regarded as a better representation of the domain, (ii) to enrich the schema with derivable parts creating diverse alternative views on the same conceptual schema as a part of the original schema, (iii) to optimise a finished conceptual schema before mapping it to a logical design.

One important issue to take into consideration when implementing a change is the management of inconsistencies that this change may introduce in the ontology. In case an inconsistency occurs, it has to be decided how to address it. While some approaches try to keep the ontology in consistent state at all cost by even disallowing changes introducing inconsistencies, others claim that the inconsistencies are inevitable and hence we have to deal with them. Regardless of the approach,

[12] http://www.neon-toolkit.org/wiki/index.pjp

[13] www.neon-toolkit.org

the inconsistencies have to be identified and resolved, maybe using some tools as it was mentioned in the introduction. In the literature, this activity has been introduced in [74] as the *semantics of the change* (originally proposed in the area of data schema evolution in [95]), and includes the computation of additional changes that guarantee the transition of the ontology into another consistent state. It enables the resolution of induced changes in a systematic manner, ensuring the consistency of the whole ontology. In particular, the author focuses on the structural inconsistencies that arise when the ontology model constraints are invalidated after a change request. Additionally, they author introduce evolution strategies to choose how a change should be resolved based on the structure of the ontology, the complexity of the process, the frequency of the strategy use or on an explicitly given state of the instances to be achieved (given by the ontology engineer).

Furthermore, another important issue that has to be addressed during the implementation of the change(s) is the management of the ontology version. After the ontology changes, the ontology engineer should decide whether the resulting ontology constitutes a new version of the ontology and hence it should have a different version information. Some recommendations on the use of URIs[14] can be found. For instance, in [44] the authors propose to use an URI for ontology identification with a two level numbering scheme: mayor and minor. Minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology). Major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies). In practice, however, it is common that ontologies do not include any version information at all. As a consequence, usually is not easy to identify different versions of an ontology. The problem of identifying ontologies in the semantic web is not a trivial issue (see [44]). For instance, in [63], a composite identification consisting of the URI plus version (if available) plus the location of the ontology is used to identify an ontology.

Finally, as aforementioned, the change(s) have to be propagated to all the ontology related artefacts (if the ontology engineer decided to do it in the previous task based on the analysis of the cost and impact). In [74] the author discusses the propagation of changes to dependent ontologies, instances, and applications and elaborates on the propagation to dependent ontologies using a combination of push and pull mechanism. For the propagation to ontology instances several mechanisms can be applied from the research in the area of databases. For instance, in [96] the authors discuss how changes can be propagated to the instances of the database by using four possible mechanisms: immediate conversion (propagate changes as they happen), deferred conversion (propagate changes at specific points in time), explicit deletion (when referenced concepts are dropped) or filtering (for using different versions of the schema).

In NeOn Deliverables D1.3.1 [65] and D1.3.2 [64] (see also [62]), the propagation of changes has also been considered to (i) distributed copies of the same ontology and (ii) ontology metadata.

As NeOn technology supporting this task we can mention that the change capturing plug-in[15] supports the logging of changes automatically from the NeOn ontology editor. Additionally it is also in charge of the propagation of changes to distributed copies of the same ontology.

As aforementioned, the RADON plug-in can be used for the management of inconsistencies.

Task 4. Verification and validation.

Before the ontology is considered evolved completely, the last step deals with assessing questions whether the right ontology is built, and whether it is built in the right way. During this assessment, usually not only the ontology originally modified is verified in isolation, but in general this activity can include the verification of other artefacts related to the ontology (as mentioned above) to ensure they were not changed in a wrong way or they have an unexpected behaviour. The verification and validation step can include the following activities:

[14] http://www.w3.org/Provider/Style/URI

[15] http://www.neon-toolkit.org/wiki/index.pjp

- Formal verification, such as state machines and temporal logics to derive useful properties of the system under study.
- Testing by users or automatically to verify whether the system behaves as expected.
- Debugging for localising and repairing errors found during the verification or testing (usually performed by an ontology engineer) e.g. [39].
- Quality assurance, which typically concerns non-functional qualities, like reusability, adaptability, interoperability, etc.
- Justification of the changes [74].

In case problems are detected, these have to be fixed by moving back into task 3, and then returning to task 4 to verify the corrected outcome.

Additionally, this task may include curation (e.g. approve/reject) activities derived from the well-defined process (i.e. workflow) that coordinate the change proposals (see NeOn Deliverables D1.3.1 [65] and D1.3.2 [64] and also [62]). In this case, ontology engineers usually have different roles and only those with the required authority can accept or reject the change proposals. If a change is rejected, the original author can modify the change and start all over again since task 1 or he can decide to discard it completely.

As NeOn technology supporting this task we can mention the following:

- The Cicero Plug-in[16] supports the justification of changes.
- The workflow Feature supports the curation activities.

7.2.1. Particularities when working with networked ontologies

Since the NeOn project deals mainly with networks of ontologies and networked ontologies [39] defined as a collection of ontologies related together via a variety of different relationships such as mapping, modularization, version, and dependency relationships, it is important to note that the process described above can also be applied to networked ontologies since many of those dependencies are taken into account as we explained. The advantage in the NeOn context is that when an ontology that is evolved is known to be part of an ontology network, that information is considered during the analysis of the impact and cost in task 2. Furthermore, during the propagation of the changes in task 3, all the ontology related artefacts are updated (if necessary), ensuring the consistency of the networked ontologies. Finally, when assessing the correctness of the evolved ontology in task 4, the verification also takes into consideration the ontology related artefacts to ensure that the whole network of ontologies is behaving as expected (i.e. is consistent).

So arising conflicts can be caught at an earlier stage and this can affect the decision whether a change should be implemented as we explained in task 2.

In the case of evolving a single ontology (which is not know to be part of a network of ontologies), it is often unknown which other ontologies will later import the ontology or even use the ontology. So interoperability cannot be considered.

7.2.2. Example

To describe the proposed guidelines for the ontology evolution activity in a more practical way, in this section we illustrate how to perform this activity by describing an experiment conducted in collaboration with a team of FAO ontology editors in charge of the maintenance of ontologies in the fishery domain. The editors performed collaboratively a set of typical changes and actions to a

[16] [16] http://www.neon-toolkit.org/wiki/index.pjp

stable version of a fishery ontology in order to reach a new stable version. In this scenario a central server kept a shared copy of the ontology and the related changes. In the remainder of this section we describe only the most relevant points. A detailed and complete description of the experiment is presented in [64].

Task 1. Requesting a change.

Initially, FAO experts in the fishery domain requested a set of changes to be applied to the current version of the species ontology[17]. That is, changes were discovered using a top-down/explicit method. A total of 31 changes were requested as shown in Table 8.

#	Description
1	Add Individual 31005_10000 (Species)
2	Add Individual 31005_10001 (Species)
3	Add Individual 31005_10000 DataProperty hasCodeAlpha3 value: DCR. Type: string
4	Add Individual 31005_10000 DataProperty hasID value: 10000. Type: string
5	Add Individual 31005_10000 DataProperty hasMeta value: 31005. Type: string
6	Add Individual 31005_10000 DataProperty hasNameEN value: Yellow-nosed albat. Type: string
7	Add Individual 31005_10000 DataProperty hasNameScientific value: Diomedea chlororhynchos.
8	Add Individual 31005_10001 DataProperty hasCodeAlpha3 value: PDM. Type: string
9	Add Individual 31005_10001 DataProperty hasID value: 10001. Type: string
10	Add Individual 31005_10000 DataProperty hasMeta value: 31005. Type: string
11	Add Individual 31005_10001 DataProperty hasNameEN value: Great-winged petre. Type: string
12	Add Individual 31005_10001 DataProperty hasNameScientific value: Pterodroma wrong macroptera.
13	Add Root Class Speciation
14	Add Individual Allopatric
15	Add Individual Peripatric
16	Add Individual ParapatricWrong
17	Add DataProperty hasSpeciation (Species)
18	Add SubClass genus of biological_entity. Right Click biological_entity and add Class genus.
19	Add Root Class Person
20	Add DataProperty name (Person)
21	Add ObjectProperty hasScientificNameAuthor
22	Add ObjectProperty domain (hasScientificNameAuthor,Species)
23	Add ObjectProperty range (hasScientificNameAuthor,Person)
24	Add Root Class Category
25	Add DataProperty description (Category)
26	Add Individual Extinct (Category)
27	Add Individual Endangered (Category)
28	Add Individual Extint DataPropertyValue (description, the last remaining member of the species has
28	Add ObjectProperty hasCategory.
30	Add ObjectPropertyRange (hasCategory,Category)
31	Add ObjectPropertyDomain (hasCategory,Species)

Table 8. Ontology Changes in FAO Experiment

In this scenario, different ontology editors were working collaboratively in the implementation of the changes and hence it was not necessary to prioritize them (prioritization of multiple changes).

[17] Available at http://www.fao.org/aims/neon.jsp

Each of the proposed changes was represented as instances of a change ontology (see [65]) (representation of changes). For this experiment, ontology editors were using the NeOn Toolkit with the collaborative infrastructure. Hence, the representation of the changes was performed automatically whenever a new change was captured by the system change capturing Plug-in of NeOn Toolkit).

Furthermore, in this scenario, the ontology editors were following a well defined process (i.e. workflow) for the coordination of the change proposals. As a consequence during this task, the system created for any new change proposal, the appropriate workflow action automatically (i.e. insert, update, delete).

Task 2. Planning the change.

For this experiment, it was necessary to implement the requested changes regardless of the side-effects. Therefore, it was not performed any analysis of the impact or cost. In fact, the idea of the experiment was to assess the efficiency of the system to support the development of an ontology in a collaborative scenario, not the time or cost of implementing a change.

Task 3. Implementing the change.

For this task, it was not necessary any restructuring of the change(s) because on the one hand the changes were not too difficult to implement due to the ontology structure and on the other hand, the cost of implementing was not an issue.

Additionally, for this task, the system (change capturing plug-in of NeOn Toolkit) took care of logging automatically all of the proposed changes (change logging), maintaining the chronological history of the events.

In this experiment, the change(s) did not introduce any inconsistencies in the ontology. However, in case it would be necessary to manage inconsistencies, the RADON Plug-in for NeOn Toolkit, could have been used to detect and fix them.

As we introduced at the beginning of this section, for this experiment, the ontology and related changes were centralized in a server. Furthermore, the ontology used for the experiment was not related to other artifacts at the moment. Hence, it was not necessary any propagation of changes.

Task 4. Verification and validation.

During this task, the ontology editors analyzed every change to ensure that the resulting ontology was as expected using the visualization plug-ins of the NeOn Toolkit.

Additionally, this task was one of the most important of the experiment as it included all the curation activities derived from the workflow that coordinates the proposal of changes. Hence, in this task, an ontology validator was in charge of accepting and rejecting changes as necessary by using the appropriate workflow Plug-ins of the NeOn Toolkit. Finally, at the moment of the experiment, there was no support for the justification of changes.

7.3. Proposed Guidelines for Exploiting Tools in Ontology Evolution

In Section 7.2 we discussed a generic approach for ontology evolution. In this section we focus on evolving ontologies through the use of semi-automatic tools, which assist the user throughout the evolution process.

Here, we propose a methodological guideline for supporting ontology engineers and domain experts in exploiting tools to facilitate the evolution of their ontologies. The goal of this guideline is to complement the tool-based support provided by the proposed Evolva framework, with concrete guidance on how to realize the various tasks of the evolution activity, using semi-automatic techniques in an efficient way.

The tasks for performing the semi-automatic ontology evolution can be seen in the workflow shown in Figure 13.

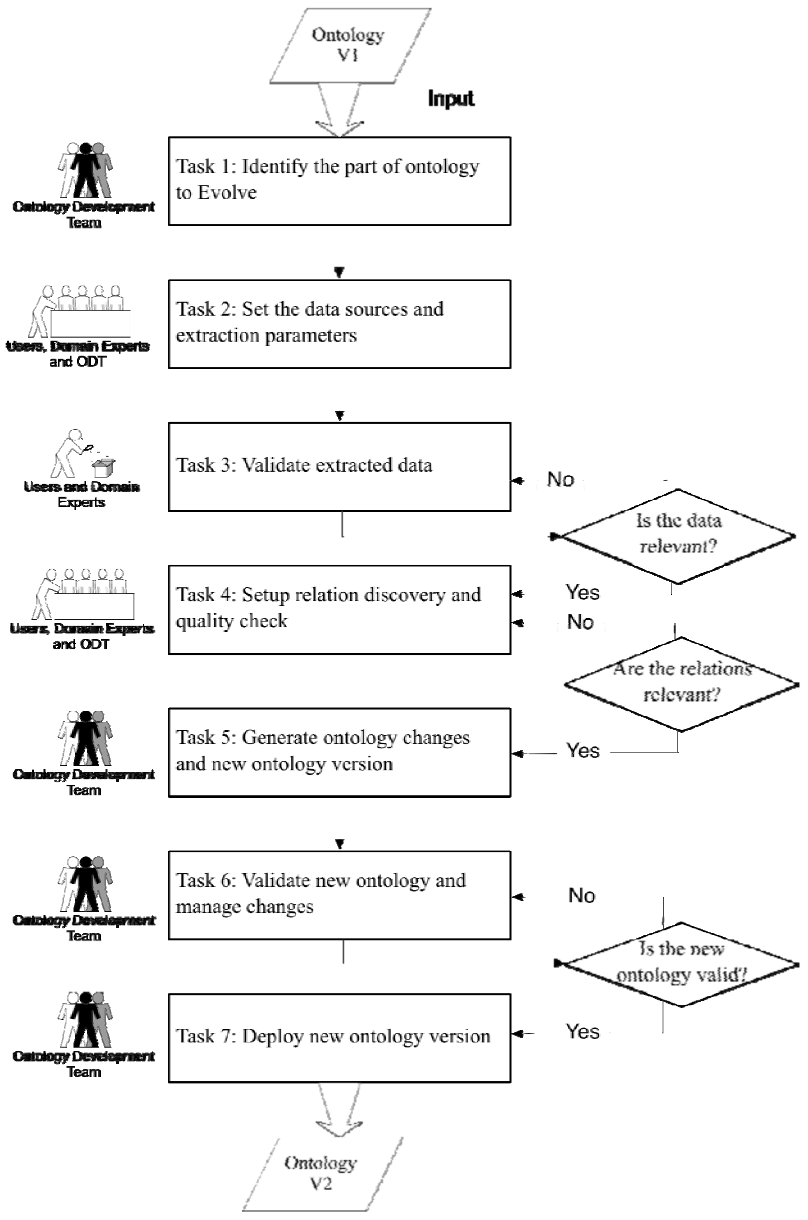

Ontology V1
Input
Ontology Development Team
Task 1: Identify the part of ontology to Evolve
Users, Domain Experts and ODT
Task 2: Set the data sources and extraction parameters
Users and Domain Experts
Task 3: Validate extracted data
No
Is the data relevant?
Users, Domain Experts and ODT
Task 4: Setup relation discovery and quality check
Yes
No
Are the relations relevant?
Ontology Development Team
Task 5: Generate ontology changes and new ontology version
Yes
Ontology Development Team
Task 6: Validate new ontology and manage changes
No
Is the new ontology valid?
Ontology Development Team
Task 7: Deploy new ontology version
Yes
Ontology V2

Figure 13. Tasks for Ontology Evolution Supported by Semi-Automatic Tools

The tasks for carrying out the semi-automatic ontology evolution are explained in detail in the following:

Task 1. Identify the part of ontology to evolve.

The first task required by the ontology development team is to select the part of the ontology to evolve. The evolution can be applied either on the entire ontology, or on a certain part of it. In many cases, ontologies may include a significant amount of statements, causing the evolution to take a long processing time. In such cases, after specifying the evolution purpose, the user may choose the part of the ontology to evolve through selecting the set of concepts to be handled by the process.

Task 2. Set the data sources and extraction parameters.

Depending on the domain, domain experts should prepare the data sources that contain relevant information to the ontology context. Such data sources could be in the form of text documents, folksonomies, databases or even other ontologies. Based on the decision of the ontology development team to evolve the ontology either in terms of schema, instances or both, the extraction should be customized accordingly. For example, in the case of schema evolution, the user may choose to extract concepts from the data sources, without dealing with instances. While in the case of instances, the evolution process could omit the extraction of schema elements. Choosing between schema and instances evolution could be biased by the ontology functionalities and domain nature. For example, when many ontology dependent components exist (e.g. various applications or other aligned ontologies), evolving the ontology schema may be costly and the ontology development team may choose to perform this operation less frequently. While in environments where ontology components are easily controllable, and where a lot of new information is generated leading to a frequent generation of new concepts, schema evolution would be required.

Task 3. Validate extracted data.

After extracting knowledge elements from the data sources, noise and irrelevant entities should be excluded. The user is supported by manual and automated validation techniques with customizable parameters. For the manual validation, the domain expert would serve as one of the best quality checkers as he/she is the most knowledgeable about the ontology context. This task is completed after checking that all the data are valid to be processed further by the system.

Task 4. Setup relation discovery and quality check.

The role of the user, after the data validation task, is to prepare the automated relation discovery process. The relation discovery process links the validated data to the ontology. This requires the user to select the various types of background knowledge sources to be used. The choice of background knowledge is directly dependent on the type of domain the ontology represents. If the domain were specialized, the user would choose domain specific background knowledge sources (e.g. specialized thesauri). This would improve the quality of relations and increase the system precision. While if the domain were generic, using online ontologies or generic thesauri would perform well. In addition to the selection of sources, the user should fine-tune the parameters of the relation discovery process, such as the selection of relations with a high-ranking measure, or specify the maximum depth to explore. Domain experts should check the quality of relations, before using them later in the system.

Task 5. Generate ontology changes and new ontology version.

Based on the approved relations in the previous task, ontology changes are generated and applied on the new ontology version. Users should specify where to apply the changes, i.e. directly on the initial ontology or on a detached copy. The choice of where to perform changes depends on the environment and the ontology development team approach. The team should be aware that applying changes on the initial ontology would directly affect the dependent components. If this is not feasible, or designers prefer to keep the initial ontology intact while reviewing the changes, creating a detached ontology version would be more appropriate.

Task 6. Validate new ontology and manage changes.

The user should control the changes performed on the new ontology version. With the new evolved ontology, problems such as inconsistencies and duplication are likely to emerge. Users in this task specify the checking methods to be applied on the new ontology version using reasoners for example, in addition to manually control the recorded ontology changes.

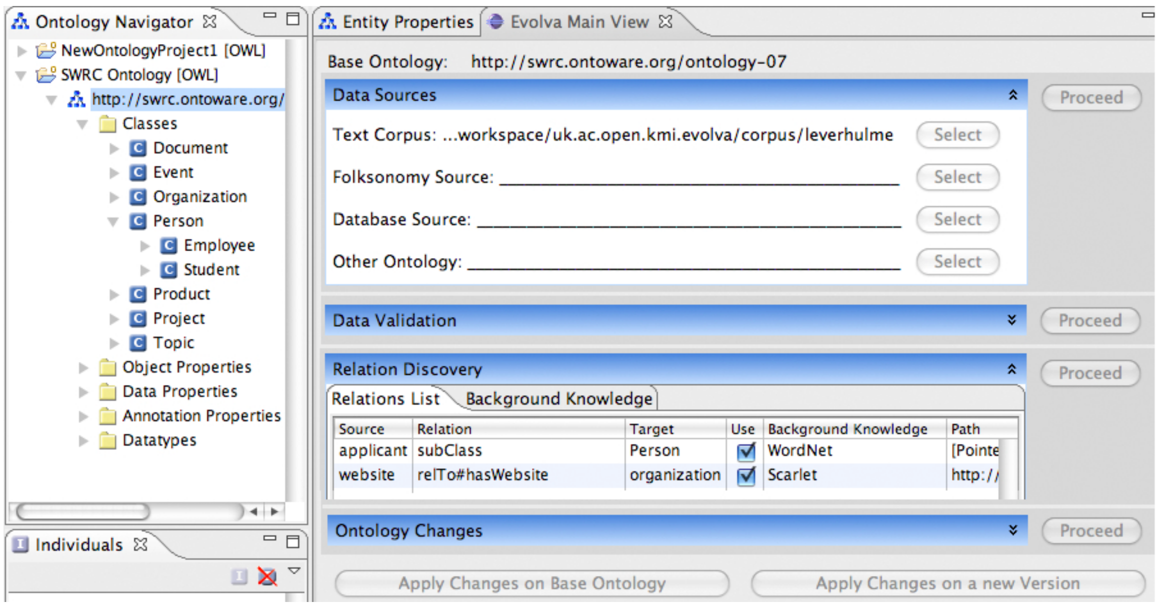
Task 7. Deploy new ontology version.

Once the new version is approved, users should control the propagation of the new ontology version to the dependent components. Links to the previous ontology version should be checked and whether the new ontology has been successfully saved and accessible.

7.3.1. Example

As described in deliverable D1.5.3 [52], we implemented Evolva as a plugin within the NeOn toolkit. Evolva provides ontology evolution through a complete framework, applied as a plugin in the NeOn ontology toolkit [52]. The various currently implemented steps in the plugin are visualized in Figure 3, covering the first three components of our ontology framework in Figure 1. In this section we highlight an example of ontology evolution, following the guidelines presented in Section 7.3. We run our example in an environment where the NeOn toolkit and Evolva plugin are operational[18].

Consider the case of evolving the latest version of the SWRC ontology[19] in the academic context. We first load the ontology in the NeOn toolkit, and start Evolva. In our simple case where the ontology has a limited number of concepts and time it not an issue, we choose to evolve all the ontology[20] (Task 1).



Base Ontology: http://swrc.ontoware.org/ontology-07

Text Corpus: ...workspace/uk.ac.open.kmi.evolva/corpus/leverhulme

Folksonomy Source: ______

Database Source: ______

Other Ontology: ______

Source	Relation	Target	Use	Background Knowledge	Path
applicant	subClass	Person	✓	WordNet	[Pointe
website	relTo#hasWebsite	organization	✓	Scarlet	http://

Figure 14. Screenshot of the Evolva plug-in

After preparing the ontology and identifying the part of ontology to evolve, we move to select the data sources containing relevant information with a potentially added value to our ontology (Task 2). A relevant source of information we found was on the Leverhulme website that contains text documents about research project, and information about people in the academic domain. We locate and download the relevant text documents, then select the sources in Evolva for performing data extraction and validation. Having no ontology dependent components, a schema evolution won't have any side effects on applications or other dependent elements. Thus, as ontology developers in this use case, we test the extraction of concepts from the data sources and integrate them in the ontology. At the time of writing this document, Evolva includes the extraction of concepts from text documents. Additional extraction features are to be implemented soon. The

[18] Details on how to install and run Evolva can be found at: http://evolva.kmi.open.ac.uk/

[19] The SWRC ontology can be downloaded at: http://ontoware.org/frs/download.php/354/swrc_updated_v0.7.1.owl

[20] Note that selecting part of the ontology to evolve is not yet available in the current version of Evolva.

validation parameters incorporate term existence checking feature (based on a similarity value), and a term length checker for removing terms under a specific length.

We load the Leverhulme text documents, and run the extraction and validation process. A list of extracted concepts is returned, with Evolva automatically identifying existing terms in the ontology, and terms that fall below a length threshold. If the automatic validation performs poorly overall, it is possible for users to fine-tune the parameters, and re-run the validation process again. In addition to the automatic validation, users have the ability to go through the list of concepts, and manually select terms they find irrelevant (Task 3). Domain experts would play here a major role as they are the most aware of the relevance of concepts with respect to the ontology.

After the data validation process and approving relevant data, we move to Task 5 of setting up the relation discovery process with the right background knowledge sources and parameters. The SWRC ontology domain is, to some extend, a generic academic purpose ontology. Thus related information can be easily found through online ontologies in which a lot of academic domain ontologies can be found, as well as through WordNet, the generic thesaurus. Thus we choose to perform the relation discovery process through exploiting Scarlet [69] (a Semantic Web based relation discovery engine) and WordNet.

Evolva automatically harvests the chosen background knowledge, and identifies how extracted concepts should be integrated in the ontology. For example, *Applicant* and *Website* are two concepts extracted from the Leverhulm text document. WorNet links *Applicant* as a *subClass* of *Person,* an existing concept in the SWRC ontology; while Scarlet links *Website* to *Organization* through a *hasWebsite* relation. The length of relations to discover is customizable. Thus if the users find that the process is taking long, or lengthy relations prove to be overall irrelevant, they can decrease the relation length threshold and re-run the process again. The user should validate such identified relations, as they will be used in the next step for deducing ontology changes.

Once all relations are approved and relevant, they are used to generate the ontology changes. If the user spots any unwanted changes, it is possible to go back to the relation validation, remove the source relations, and regenerate the ontology changes. Based on the ontology changes provided, we apply them both directly on the SWRC ontology, and on a separate ontology copy that includes all the new changes applied.

Our next task is to validate the new ontology version, and manage the new changes that the ontology has been subject to (Task 6). Evolva relies on the change logging plugin [65] based on the NeOn toolkit. The user is given all the functionalities to review changes after being applied on the ontology. Inappropriate changes can be rolled back, or sent for further review, until reaching a reliable new ontology version.

After approving the final ontology version, we deploy it by double-checking the links to the previous ontology version that are automatically created by Evolva (Task 7). We also check that the ontology has been saved correctly, and it is still accessible by doing some random checks such as running queries and validating the results.

7.4. Future Work

In this chapter we have presented detailed methodological guidelines for carrying out the ontology evolution activity. Further work related with this activity includes:

- ❑ Evaluation of the proposed guidelines with additional experiments.
- ❑ Refinement of the guidelines with the received feedback.
- ❑ Integration with other activities of the ontology engineering methodology.

8. Ontology Localization

Ontology Localization is defined in [78] as the activity that consists in adapting an ontology to a particular language and culture. The output of this activity is an ontology (multilingual ontology) whose ontology terms expressed in a source natural language have been translated to the target natural language. The resulting translations are added to available labels of the original ontology already in one or several languages.

In this chapter we present a brief introduction to the existing methods, techniques and tools for ontology localization. We also propose the NeOn methodological guidelines for carrying out the activity.

8.1. State of the Art

In this section we present some of the methods, techniques and tools that we found in the literature related with the localization of an ontology to different natural languages. First, we introduce existing general methods for carrying out the ontology localization activity. Then, we describe some techniques for ontology localization which has been guided by the observation of how the localization process is performed by a human expert. We also present some tools which support the localization activity to create a multilingual ontology. And finally we include some conclusions.

8.1.1. Methods

As we mentioned before, the goal of the ontology localization is to build a multilingual ontology. Two main streams of research are emerging as alternative to build a multilingual ontology: the first is based on the integration of language-specific ontologies via ontology merging techniques, while the second assumes the localization of an existing ontology.

1. *Reconciliation and Merging of Existing Monolingual Ontologies.* According to the first approach, people speaking different languages will model a given domain area by defining different ontologies that will be merged to provide a multilingual semantic environment. As many aspects of the predefined knowledge area will be common for all languages, many redundant “synonymous" relationships between language specific ontologies will be defined, increasing resource wasting and complexity. Developing a multilingual semantic framework using the above exposed methodology, involves then the risk of getting an unmanageable entity as an outcome, in which great care is required to define relationships between “equivalent ontologies" and to track changes and coherently update those relations [9].
2. *Localization of an Existing Monolingual Ontology.* Within this second trend, we have to differentiate between two approaches: 1) the inclusion of multilingual information in the ontology (what we refer to as *non-modular approach*), and 2) the association of external multilingual information with the ontology (*modular approach*).In both approaches there exists a monolingual ontology covering the subject domain of the proposed multilingual ontology, and serving as the source language from which is possible to obtain the corresponding translations in different languages. In the *non-modular approach*, multilinguality is obtained by including ontology labels in different languages by making use of the *rdfs:label* property (see RDF(S)[21]). In the second approach, *the modular approach*, an external linguistic model is associated to the ontology. Different models have been proposed to associate linguistic data to ontologies: the Linguistic Information Repository (LIR) [54] specially designed to link multilingual

[21] www.w3.org/TR/rdf-schema/

information to ontologies, or LingInfo [11] and LexOnto [15], whose focus is on the association of further linguistic data to ontology labels such as morphosyntactic decomposition of labels or subcategorization frames of properties, respectively.

In this deliverable we focus on the creation of guidelines that supporting the localization of an existing monolingual ontology for supporting the building of a multilingual ontology. We believe that this approach is inherently scalable, as new languages can be easily supported by integrating new lexical entities and definitions, and possibly slightly restructuring a small number of ontology terms.

8.1.2. Techniques

To our knowledge, no other study has focused on the techniques for ontology localization. In our opinion, the starting point in the design of an ontology localization system and the techniques it relies on should be guided by the observation of how the localization process is performed by a human expert. As the localization activity involves the use of translation techniques, we study the different attempts to model the human translation process. In particular, we adopt the model organized in four steps: 1) meaning discovering, 2) finding receptor (target) language equivalents, 3) checking the meaning of the receptor (target) language item, and 4) formulation of the final translation. Figure 15 illustrates the steps above described.

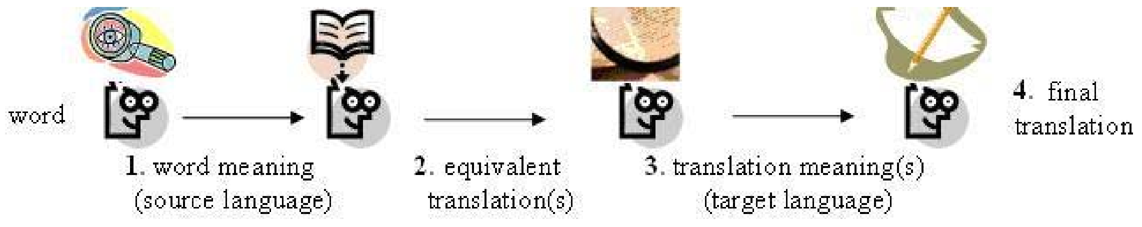

word
1. word meaning (source language)
2. equivalent translation(s)
3. translation meaning(s) (target language)
4. final translation

Figure 15. Human Translator Steps

In order for a human translator to be able to discern among the different meanings a word may have (homonymic or polysemic words), (s)he needs to analyse the context. Depending on the context in which the word is used, a certain meaning will be selected, whereas the rest of potential meanings of the word will be discarded. This process is performed almost unconsciously in the translator's mind if (s)he has a good command of the subject and the terminology used in it. For example, if the word to be translated is ‘bank', and the text is about finances, the translator will undoubtedly assign the meaning “financial institution" to the word ‘bank'.

The next step in the translator's mind is to look for possible equivalents of the word ‘bank' with the meaning of “financial institution" in the target language. Assuming the translator's proficiency on the subject in both, the source and target culture, the translator will look for an equivalent concept in the target culture. If ‘bank' is going to be translated into Spanish, the translator has to find out if the English word is referring to a “savings bank" or to an “investment bank", for example, since in the first case, ‘bank' would be translated into ‘caja' and in the second into ‘banco'. Here again the context is essential for the translator to make the right choice. At this stage, it is difficult to separate this action into two steps, (finding language equivalents and checking their meaning) because concepts are represented by lexicalizations, and they come together as indivisible items in the translator's mind.

In order to take the final decision on which the most appropriate translation for a certain word is, the translator will have to take into account two additional aspects: 1) which is the concept in the target language that better matches the concept in the source language?, and 2) which is the purpose of the translation? Once the purpose and context have been checked again, the translator is able to select the most appropriate translation for the source word.

In the following, we describe how each phase may be automated in some way.

1. Meaning Discovering. A way of discovering the meaning of an ontology label (name of an ontology term) to be localized is to extract small excerpts that describe the specific entity (here context[22]). The context describes the meaning of a specific term in the ontology. In a broader perspective in the ontology localization process, we can look at ontology label context from two different dimensions, considering: 1) the size of the context, and 2) the depth of processing.

When looking at the size of the context, it is useful to divide it into three categories [47]: i) without context, ii) with local context, and iii) with global context. The first alternative is sometimes disregarded, but it contains important information about the internal structure of the word, e.g., capitalization, type of ontology term (class, object property, instance, etc.), which often relates the word to other words with similar meaning in the ontology. The local context can be defined as a narrow window of 3-5 words centred on the ontology label itself. A narrow window shows a set of the semantically related terms, such as hyperonyms, hyponyms, etc. The global context can be a window of 25-1000 words centred on the ontology label itself, which fairly well approximates contexts starting from the immediately surrounding direct relationship terms to the whole ontology.

As an orthogonal dimension of context we have the depth of pre-processing. There are different points of view that can be applied for ontology context pre-processing: syntactic, and structure-level. For example, a syntactic context processing interprets the context information as a “bag of words". A structure-level context interprets the information as the entities that appear hierarchically organized in an ontological structure.

Obtaining the context of each term in the ontology could be executed using techniques of ontology module extraction [57, 72].

2. Finding target language equivalents. Regarding finding language equivalent terms, cross-language term extraction methods can be used by the system to perform this task. The goal of automatic term extraction is to identify candidate terms in (more or less) unstructured text. The most common way of discovering candidate translations is to use bilingual dictionaries [17]. Dictionary based approaches are fairly easy to implement and they offer quite effective tools for retrieving translations for ontology labels. However, significant performance degradation is often observed when ontology labels contain words that do not appear in the dictionary. If there is no word in the dictionary for an unknown label, the label is usually left untranslated. The most important categories of untranslatable labels are compound labels, proper names and their spelling variants and special terms.

A complementary method for discovering translation equivalents is Web mining [18, 90]. This method is based on the observation that new terms, foreign terms, or proper nouns are used in a source language web text (e.g., Spanish), they are sometimes accompanied by the target language translations (e.g., English) in the vicinity of the source text. For example, ‘Marinos de Seattle’ (Seattle Mariners). After mining the Web to collect a sufficient number of such instances for any proper noun or a name entity and applying statistical techniques, we are able to infer the appropriate translation with reasonable confidence. For those languages with no word boundaries (e.g., Chinese), the text should be correctly detected first, then the mining system has to perform segmentation of the sentences to find the candidate words. The quality of the segmentation greatly influences the quality of term extraction because incorrect segmentation of the source text may break the correct translation of the target term into two or more words thus losing the correct word.

3. Checking the meaning of the target language lexical unit. Regarding the translator's task of checking the label meaning in the target language, we believe that this task may be automated using word sense discovery approaches. Word sense discovery is defined as the task of learning

[22] In NLP, context it is the environment in which a word is used, and context, viz. word usage, provides the only information we have for figuring out the meaning of a new or a polysemous word

what senses a word may have in different contexts [47]. This task can be seen achieved in three steps 1) to determine the groups of related words in context, 2) to determine a suitable inventory of word sense labels, and 3) to learn how to associate a word sense label with a word cluster using either machine learning or manually created rules or metrics.

To the best of our knowledge, in the ontology localization activity, not only unstructured resources (e.g. dictionaries) but also structured resources that already encode some kind of semantic knowledge (e.g., thesauri, lexicons, wordnets) are used to discover word meanings. However, these kinds of approaches usually rely on static knowledge and data. Therefore, they cannot effectively reflect the quickly shifting interests of ontology users. Additionally, there are several disadvantages associated with these word senses. First, manually created lexicons often contain rare senses. For example, WordNet (version 1.5) included a sense of computer that means “the person who computes". Another problem with these lexicons is that they miss many domain specific senses. For example, WordNet misses the user-interface-object sense of the word dialog (as often used in software manuals).

Other resources may also be considered to avoid these problems, for instance the semantic knowledge available on online ontologies. These approaches rely on mechanisms that not only do dynamically the selection of appropriate ontologies from the Web, but also extract from these ontologies the relevant and useful parts [61, 86].

4. Formulation of the final translation. At this stage, we consider that a way of selecting the most appropriate translation is to use word sense disambiguation techniques. Word sense disambiguation involves the association of a given word in a text or discourse with a definition or meaning which is distinguishable from other meanings potentially attributable to that word. According to [41], this task necessarily involves two steps. The first step is to determine all the different senses for every word relevant to the text or discourse under consideration (see previous step). The second step involves a means to assign the appropriate sense to each occurrence of a word in context.

8.1.3. Tools

In this section we present tools which support the localization activity to create a multilingual ontology. This survey is not complete, but we present the most representative tools.

TermTranslator. In [31] authors propose a system for supporting the multilingual extension of ontologies existing in just one natural language. This tool is used to support “the supervised translation of ontology labels" and, at the same time, to allow for the semantic annotation of multilingual web documents using the resulting multilingual labels of ontologies. Therefore, the tool offers a semi-automatic strategy.

Ontoling. In [61] authors propose a framework for a semi-automatic linguistic enrichment of ontologies. They use two resources for the linguistic or multilingual enrichment, WordNet, for the linguistic enrichment of ontologies with English labels, and DICT dictionaries, for the linguistic and multilingual enrichment of ontologies. The tool also uses an enrichment component to exploit the taxonomical structure of the glosses of the linguistic resource to judge which linguistic information can be used to enrich the ontology. The main limitations of this work are that it only supports OWL ontologies, the loaded linguistic resource must be a taxonomical lexical resource and/or a linguistic resource with glosses, and the linguistic and multilingual information obtained from the resources is stored in the ontology itself.

LabelTranslator system. In [32, 33] authors propose a system for supporting the ontology localization activity. The tool is called **LabelTranslator** [32, 33], and takes as input an ontology whose labels are described in a source natural language and obtains the most probable translation of each ontology label in a target natural language. LabelTranslator [32, 33] has been designed with the aim of automating ontology localization, and has been implemented in the ontology editor NeOn Toolkit[23] as a plug-in. In its current version, it can localize ontologies in English, German and Spanish. First, the system uses a translation service which obtains automatic translations of each ontology label (name of an ontology term) in English, German, or Spanish by consulting different linguistic resources such as lexical databases, bilingual dictionaries, and terminologies. This service also uses a compositional method in order to perform the translations of composed labels. The compositional method first searches for translation candidates of each lexical component and then builds the translations for the candidates using lexical templates. Second, a ranking method sorts the different translations according to the similarity with its lexical and structural context. The method relies on a relatedness measure based on glosses to disambiguate the translations. This is done on the basis of the comparison of the senses associated to the translation and their context.

8.1.4. Conclusion

As a conclusion we can mention that there is no universal way to localize an ontology and that the choice of a specific method and tool to build an ontology localization system should be guided by the final purpose of the localized ontology and by the techniques deemed to be more efficient for each task of the activity. Most of the analyzed methodologies and techniques propose simple methods for carrying out the ontology localization activity inspired in the human translation process. The methods consist of high level steps that can be summarized as follows: discover the meaning of the ontology term, find target language equivalents, check the meaning in the target language and formulate the final translation of each localized term. However, these methodologies do not provide guidelines explaining how to carry out each step.

Finally, we can mention that some tools exist to enrich an ontology with linguistic information. However, only one tool, called LabelTranslator, has been designed and implemented from its creation for supporting all tasks of the activity of localization.

8.2. Proposed Guidelines for Ontology Localization

As we mentioned before, the goal of the ontology localization is to adapt an ontology to a particular language and culture.

In the context of the NeOn methodology for building ontology networks we propose the ontology localization filling card, presented in Table 9, which includes the definition, goal, inputs and outputs, who carries out the activity and when the activity should be carried out.

[23] http://www.neon-toolkit.org/

Ontology Localization	
Definition	
Ontology localization refers to the adaptation of an ontology to particular language and culture.	
Goal	
To translate an ontology expressed in a source natural language into a target natural language.	
Input	*Output*
An ontology whose ontology terms are expressed in one or several natural languages, from which one is selected as source natural language.	An ontology whose ontology terms have been translated to the target natural language. The resulting translations are added to available labels of the original ontology already in one or several languages.
Who	
Software developers and ontology practitioners, who form part of the ontology development team, in collaboration with domain and linguistic experts.	
When	
Once the conceptual model of the ontology is stable, with the aim of avoiding spending time and resources in a model that is not definitive.	

Table 9. Ontology Localization Filling Card

The tasks for carrying out the ontology localization activity can be seen in Figure 16. The result of this activity is an enriched ontology (multilingual) with linguistic information (into target language) associated to each localized term.

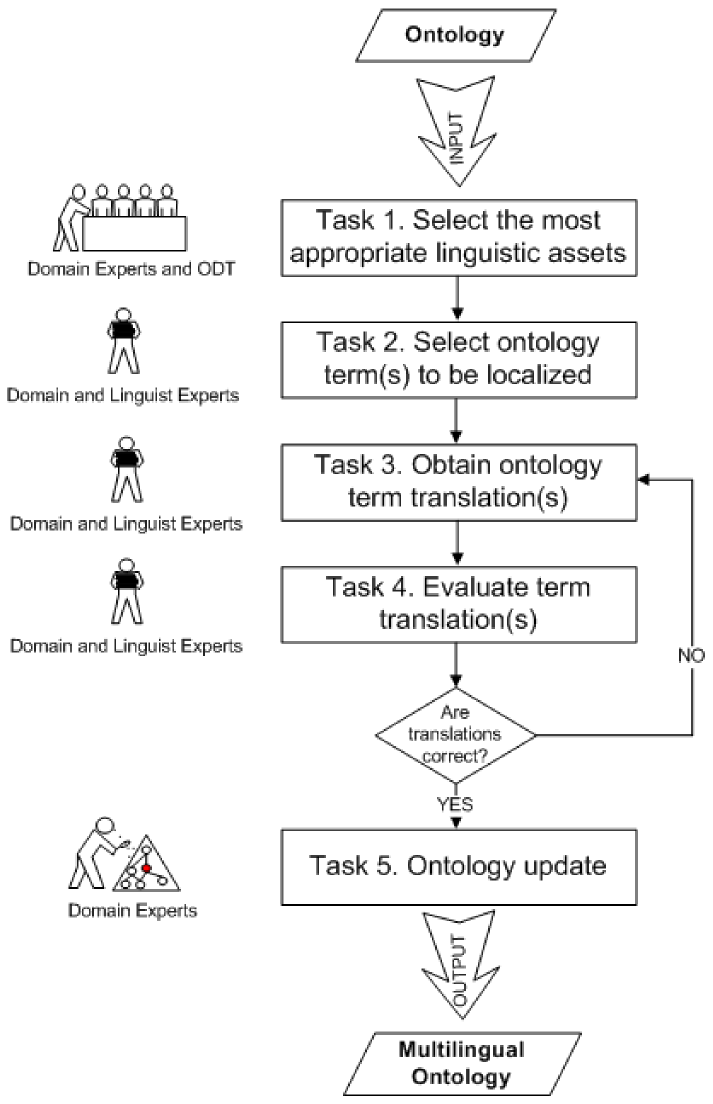

Ontology
INPUT
Domain Experts and ODT
Task 1. Select the most appropriate linguistic assets
Domain and Linguist Experts
Task 2. Select ontology term(s) to be localized
Domain and Linguist Experts
Task 3. Obtain ontology term translation(s)
Domain and Linguist Experts
Task 4. Evaluate term translation(s)
NO
Are translations correct?
YES
Domain Experts
Task 5. Ontology update
OUTPUT
Multilingual Ontology

Figure 16. Tasks for Ontology Localization

The tasks for carrying out the ontology localization activity are explained in detail in the following:

Task 1. Select the most appropriate linguistic assets.

The goal of this activity is to select the most appropriate linguistic assets that help in the localization activity. User and domain experts carry out this activity taking as input the ontology to be localized. The activity output is a set of linguistic assets that can help to reduce the cost, improve the quality and increase the consistency of the localization activity.

Linguistic assets may include computer aided translation (CAT) tools or machine translation (MT) tools. It is important to make a distinction between MT tools and CAT tools. Where the purpose of a machine translation tool is to assume and perform many of the tasks normally completed by a translator, computer aided translation tools are used to support the translator, by eliminating repetitive work, automating terminology lookup activities, and recycling previously translated text.

Computer aided translation tools, also called computer assisted translations tools, can be categorized as follows:

- Translation Memory tools. A key technology that enables to user to store translated phrases or sentences in a special database for local re-use or shared use over a network. Translation memory systems work by matching terms in the database with those in the source text. If a match is found the system proposes the ready-made translation in the target language.

- Terminology tools. A terminology management tool is much more that just creating and maintaining bilingual list of translated terms. These tools enables to translators to maintain term databases that include additional information such as definitions, context, gender, source, synonyms, etc. A type of terminology tool that is often used in localization is the electronic dictionary; however other electronic resources as specialized encyclopaedias or multilingual corpora can be used too.
- Ontology Localization tools. These tools are designed to help translators localize ontology terms. Most of these tools combine, translation memory, leveraging (re-using), and validation.

As a rule, the first two, translation memory and terminology tools, are generally combined in a tool set for the translation of sentences commonly found in the annotation of classes, properties or instances, (e.g. consider rdfs:description). On the other hand, ontology localization tools can be used to translate ontology terms, i.e. ontology labels, ontology identifiers, instances, etc.

Other option to support the localization of ontologies is to use machine translation tools. These tools are applied to any levels, from processing a huge amount of sentence strings to online “gisting” offered on many web sites. Currently, more and more translation memory (TM) systems offer support for MT. In a typical setup where TM and MT are used, the computer first searches the translation memory for a match of the term to be translated. If no match is found, the translator can ask the MT system to translate the term, edit the result and store the translation in the translation memory.

The choice of a specific tool usually depends on one o more of the following criteria:

- Supported languages. Each tool support different feature sets and language sets. Make sure all target languages for current and possible future ontology localization projects are supported by the tool.
- Support. How much support will be required for the translators using the tool?
- Alignment. If previous translations do exist, but not translation memories, the translated ontologies need to be aligned in order to create a translation memory. Check whether the tool contains an alignment feature.
- Network support. Does the tool allow translation memories to be shared over a network?. Which networks protocols are supported?
- Customization. Does the tool contain features that enable the user to customize the localization of specific terms?

To select the appropriate translation tool for performing the ontology localization activity, follow the basic preliminary guidelines presented in Table 10.

Type of Ontology Term	Translation Tool	Comments
Ontology labels and instances.	Ontology Localization tool, Machine translation tool or computer aided translation tool.	Machine Translation could be used effectively to translate ontology labels and instances only if the ontology terms are very controlled.
Ontology term annotations.	Translation Memory tool.	The major cost involved at this level is the difficulty to translate correctly long pieces of text.
Culture specific terms.	Native creation tool.	The tool requires a thorough analysis of the cultural conventions of source and target communities.

Table 10. Criteria to select the most appropriate Translation Tool

Task 2. Select ontology term(s) to be localized. The goal of this task is to select the ontology term(s) to be localized. Domain experts and the ontology development team carry out this task taking as input an ontology whose terms expressed in a source natural language need to be localized to a target natural language. The task output is a set of ontology terms with information of the text to be translated and its context.

Since there are no methodological guidelines for guiding in the selection of the ontology terms, we believe that the user is the one who has to choose the space of candidates to be localized. At this stage, the user may choose to localize the complete ontology or certain terms only.

Task 3. Obtain ontology term translation(s). For each ontology term, the goal of this task is to obtain the most appropriate translations in the target language. Domain and linguist experts carry out this task taking as input the text of the ontology term(s) to be localized.

Different techniques can be used to automatize this task (see Section 8.1.2 for more details):

1) cross-language term extraction to discover translation equivalents,
2) word sense discovery for discovering the possible senses or definitions of the translations, and
3) word sense disambiguation for selecting the most appropriate translation for each ontology term(s).

The task output is a ranked set of translations for each ontology term(s).

Task 4. Evaluate term translation(s). The goal of this task is to evaluate the translations of each ontology term. Domain and linguist experts, carry out this activity taking as input the translations of each ontology term(s) localized. The output of this task is a set of ontology terms with its corresponding evaluation.

Different linguistic criteria can be used for the evaluation of the translations of each ontology term(s). We propose three levels of evaluation criteria and for each level we propose a set of tests, which can be automated as far as possible.

1. Terminology evaluation. The aim of terminology evaluation is to control that the obtained translations do not alter the knowledge represented in the original ontology. In the following we describe the tests that should be checked:
   - *Unchanged test* checks whether a translation is basically identical to the original string. This test checks to see if the translation is not just a copy of the source language. Sometimes, this is what you want, but other times you will detect words that should have been translated.
2. Semantic fidelity evaluation. The aim of a semantic fidelity evaluation is to control that the translation be conceptually equivalent to the ontology term in the source language. Additionally is necessary to check if the translation is clear and easy to understand. A way of evaluating the semantic fidelity is described in the following:
   - *Backward Translation test* provides a quality-control step demonstrating that the quality of the translation is such that the same meaning is derived when the translation is moved back into the source language.
3. Stylistic evaluation. The aim of stylistic evaluation is to control the clarity and beauty of language, which depend on the style of the source language and on the peculiarities of the individual idiolect. The following are descriptions of the tests that should be checked:
   - *Acronym test* checks that acronyms that appear are unchanged. If the acronym term appears in an ontology term, this test will check that it appears in the translation. Translating acronyms is a language decision but many languages leave them unchanged in that case this test is useful for tracking down translations of the acronym and correcting them.

- *Blank test* checks whether a translation is totally blank. This will check to see if a translation has inadvertently been translated as blank i.e. as spaces. This is different from untranslated which is completely empty.
- *Double words test* checks for repeated words in the translation. Words that have been repeated in a translation will be highlighted with this test e.g. “the the”, “a a”. These are generally typos that need correcting. Some languages may have valid repeated words in their structure, in that case either ignore those instances or switch this test.
- *Long test* checks whether a translation is much longer than the original string. This is most useful in the special case where the translation is multiple words long while the source text is only one word long. Be aware that this test may create a number of false positives.
- *Short test* checks whether a translation is much shorter than the original string. This is most useful in the special case where the translation is one word long while the source text is multiple words long. Be aware that this test may create a number of false positives.
- *Simple caps test* checks if the capitalization of two strings is not largely different. This test is useful for identifying translations that do not start with a capital letter (upper-case letter) when they should, or those that do when they shouldn't. It will also highlight sentences that have extra capitals; depending on the capitalization convention of your language, you might want to change these to Title Case, or change them to normal sentence case.

Task 5. Ontology update. The goal of this task is to update the ontology with the linguistic information obtained for each ontology term(s) in the target language. Domain experts carry out this task taking as input the selected translations for each ontology term expressed in a source natural language. The activity output is an enriched ontology with linguistic information (into target language) associated to each localized term. In this case, we consider two strategies: non-modular and modular.

- ❑ Non-modular strategy. In the non-modular strategy the association between linguistic content and ontological data is projected over standard RDFS/OWL predicates. Different features included in RDFS (and therefore in OWL) support linguistic tagging of ontology terms by using: (i) rdfs:label and rdfs:comment, (ii) sub-properties of rdfs:label, (iii) language identifiers (as specified by RFC 3066) and xml:lang, (iv) embedded XML, etc.

  For example, the rdfs:label property is used for addressing short lexical objects like terms or words (used both to provide synonymic expressions as well as to provide translations for different languages) while rdfs:comment is commonly associated to wider descriptions like those which could be extracted from word glosses and term definitions. These options, though guaranteeing a complete adherence to widely accepted standards, offer poor representation primitives.
- ❑ Modular strategy. The modular strategy relies on the combination of two independent modules, the ontological and the linguistic one. Different models have been proposed to associate linguistic data to ontologies: the Linguistic Information Repository (LIR) [54], specially designed to link multilingual information to ontologies, or LingInfo [11] and LexOnto [15], whose focus is on the association of further linguistic data to ontology labels such as morphosyntactic decomposition of labels or subcategorization frames of properties, respectively.

8.3. Examples

In this section we include two different examples of how to use the proposed guidelines for the ontology localization activity and the obtained results.

The first example refers to the localization of the Pest control ontology, by means of using the guidelines proposed in this deliverable. Basically, the ontology localization activity is carried out by FAO Information Management specialist with the contribution of domain and linguistic experts. In particular we will refer to ISMEA, the Italian Institut “di Servizi per il Mercato Agricolo Alimentare” and internal FAO translators and systems.

The second example instantiate the guidelines for the localization of the ontology proposed in the first sample (Pest control ontology), but using an automatic localization tool (named LabelTranslator). It is important to mention that the work done within the LabelTranslator system in the localization activity has been one of the inputs to get preliminary guidelines for this activity. Such preliminary guidelines have been extended, improved, and proposed in this deliverable. Using the proposed guidelines presented in this deliverable, we described the localization activity with the Pest control ontology.

8.3.1. Pest control ontology localization

The objective in this FAO example is to localize the Pest control ontology from English to French and Italian. The input ontology is a module of the AGROVOC Concept Server containing English terms identifying one or more concepts.

Next, we show the tasks that we have carried out to localize the Pest control ontology to French and Italian.

Task 1. Select the most appropriate linguistic assets.

In general FAO experts make use of several computer aided translation tools to perform the localization of their ontologies. In this particular use case, the following criteria were used to select the linguistic assets for performing the ontology localization activity:

- Supported languages. The selected tools supported at least one of the target languages.
- Support. FAO experts have experience using the selected tools, therefore they do not require additional support.

Basically, the FAO experts use mainly the FAOTERM, the institutional multilingual terminological system[24]. FAOTERM provides a terminology database, with the correct language equivalents, and standardize terminology within the Organization and within the United Nations system as a whole. This system also provide a query service for all terminology and reference request in FAO, from external translators and editors as well as for international organizations.

FAOTERM may be useful only for a limited number of languages, mostly the 5 official FAO languages. Russian sometimes may also be available because of recent addition, and Italian may sometimes be available because of the location of the FAO HQ premises.

In addition to the one mentioned above, another important assets used to help in the localization activity is to make use of the Google define functionality. Finally for this use case FAO experts may also make use of cataloguing systems such as AGRIS or FAODOC.

Task 2. Select ontology term(s) to be localized.

From the Pest control ontology, we manually extracted the concept term “pest control” and their related terms that will be localized into French and Italian.

The related terms are:

- Postharvest sparring,
- Product protection,

[24] http://www.fao.org/faoterm/

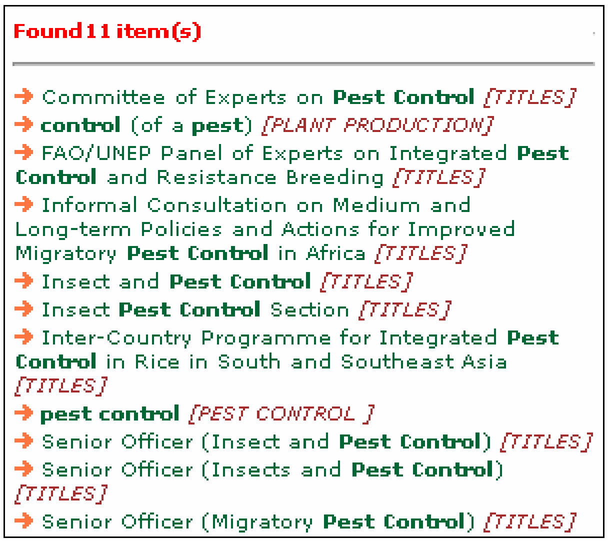
- Postharvest control, and
- Postharvest treatment.

Task 3. Obtain ontology term translation(s).

For each ontology term we use a manual process for discovering translation equivalents, discovering the possible senses or definitions of the translations, and to disambiguate the translation senses.

Discovering translation equivalents

For example for the term "pest control", we find 11 entries (using FAOTERM). Most of them are titles of conferences or journals. However, two entries refers to terminology in the area of Plant production ("control (of a pest)") and in the area of pest control ("pest control"). Figure 17 shows some related items for the term "pest control".


Found 11 item(s)

- Committee of Experts on **Pest Control** *[TITLES]*
- **control** (of a **pest**) *[PLANT PRODUCTION]*
- FAO/UNEP Panel of Experts on Integrated **Pest Control** and Resistance Breeding *[TITLES]*
- Informal Consultation on Medium and Long-term Policies and Actions for Improved Migratory **Pest Control** in Africa *[TITLES]*
- Insect and **Pest Control** *[TITLES]*
- Insect **Pest Control** Section *[TITLES]*
- Inter-Country Programme for Integrated **Pest Control** in Rice in South and Southeast Asia *[TITLES]*
- **pest control** *[PEST CONTROL]*
- Senior Officer (Insect and **Pest Control**) *[TITLES]*
- Senior Officer (Insects and **Pest Control**) *[TITLES]*
- Senior Officer (Migratory **Pest Control**) *[TITLES]*

Figure 17. Related Items for the Term "pest control" extracted from FAOTERM

The linguistic information related to the result “**pest control**” is shown in Table 11.

Arabic	
تاف _ ةحفاكم لا	
Source: ATG/KCCM, FAO, 2007.	
English	
pest control	
Spanish	
lucha contra las plagas	
Remarks: FAOTERM-Subjects; TRG(A-Sys1)/KCCM, FAO, 2007.	
lucha antiparasitaria	
control de plagas	
French	
lutte contre les ravageurs	
Remarks: FAOTERM-Subjects; TRG(A-Sys1)/KCCM, FAO, 2007.	
lutte phytosanitaire	
Russian	
●●●●●●●●●●●●●●●●●● ●●●●●●●●●	
Chinese	
有害生物防治	
Source: CTG/KCCM, FAO, 2007.	
Record info	
Entry Number:75416	
Category	**Terminology**
Status	Validated
Reliability	2
Source Language	en
Source	FAOTERM-Subjects; TRG(A-Sys1)/KCCM, FAO, 2007.
Subject	PEST CONTROL

Table 11. Linguistic Information related to Area of “pest control”

On the other hand, the linguistic information related to the result “**control (of a pest)**” is shown in Table 12.

Arabic	
	ةحفاكم **Definition** ةريشع ةد ابإ وأ ءاوتحإ ، د امخإ _ ةف
English	
	control (of a pest) **Definition** Suppression, containment or eradication of a pest population.
Spanish	
	control (de una plaga) **Definition** La supresión, contención o erradicación de una población de plagas.
French	
	lutte (contre un organisme nuisible) **Definition** Suppression, enrayement ou éradication de la population d'un organisme nuisible.
Chinese	
	病虫害控制 病虫害防治 **Source** Chinese Academy of Agricultural Sciences, CAAS, Beijing, FAO Language Resources Project 2005. Ref.1: English-Chinese Dictionary of Agriculture, the First Edition, 1998, China Agricultural Press, Beijing. (www.ccap.com.cn).
Record info	
Entry Number:17532	
Category	**Terminology**
Status	Validated
Reliability	5
Source Language	en
Source	IPPC GLOSSARY
Subject	PLANT PRODUCTION
Glossint	Phytosanitary Terms

Table 12. Linguistic Information related to Area of “control (of a pest)”

As we could see this technique may be useful only for a limited number of languages, mostly the 5 official FAO languages. However, for Italian no translations have been yet identified. Domain experts at this point will preferably refer to specialized dictionaries, online or printed. Thus, for example we obtain the equivalent translations for the term “pest control” in Italian shown in Table 13.

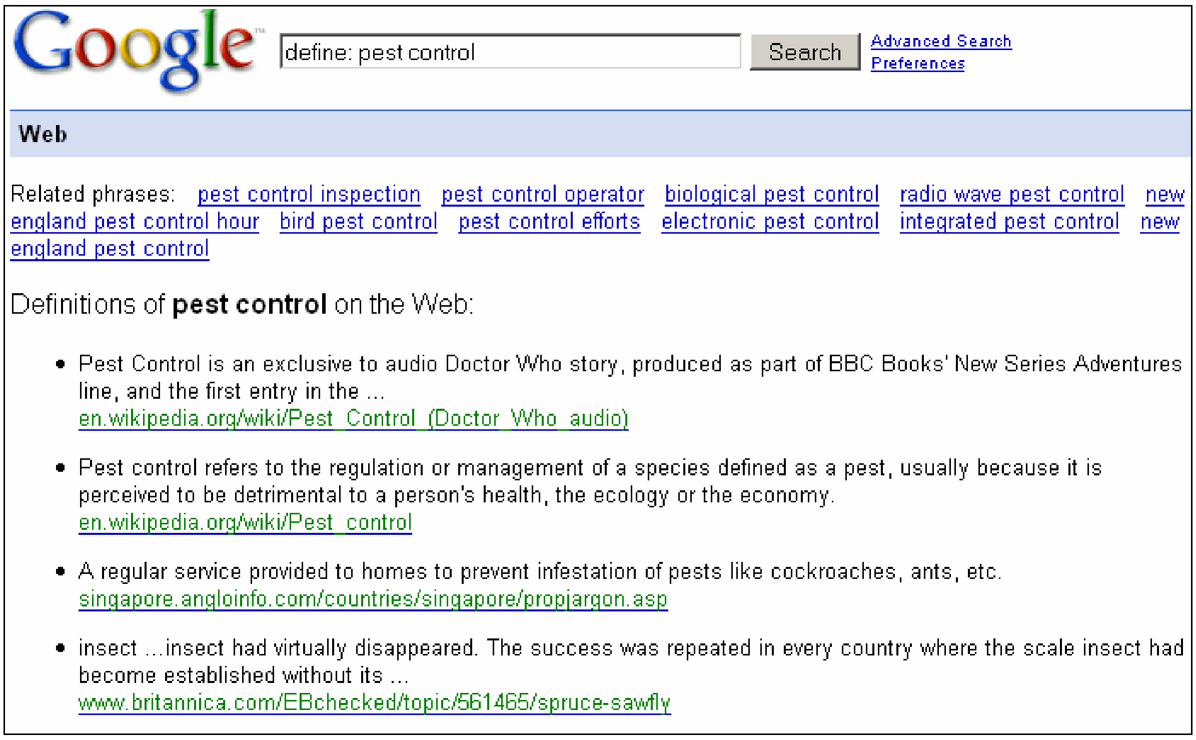
Nebulizzazione postraccolta
Difesa dei prodotti immagazzinati

Difesa postraccolta
Trattamento postraccolta
Controllo dei parassiti (postraccolta)

Table 13. Equivalent Translations in Italian of the Term “pest control”

Discovering the possible senses or definitions of the translations

For discovering the definitions we use Google define functionality. In Figure 18 we show a sample of the definitions obtained for the term “pest control”.



Related phrases: pest control inspection pest control operator biological pest control radio wave pest control new england pest control hour bird pest control pest control efforts electronic pest control integrated pest control new england pest control

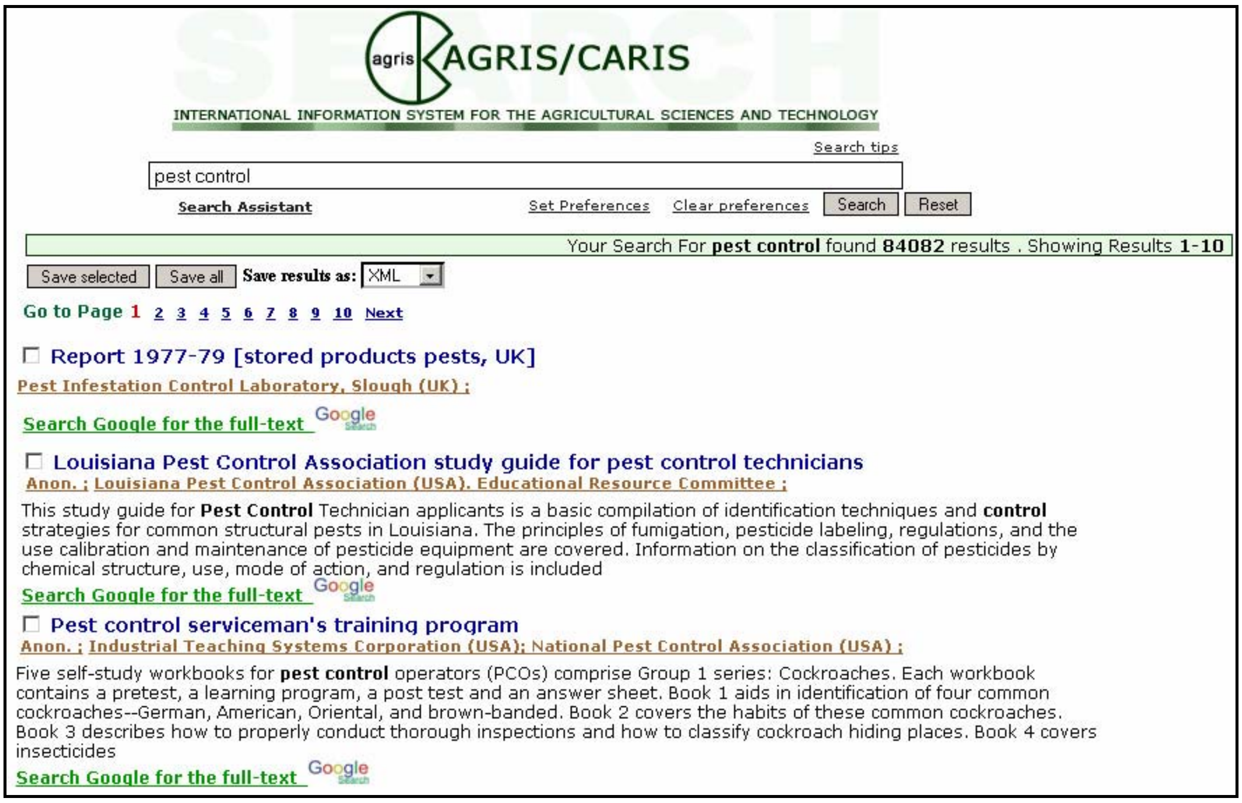
Definitions of **pest control** on the Web:

- Pest Control is an exclusive to audio Doctor Who story, produced as part of BBC Books' New Series Adventures line, and the first entry in the ...
  en.wikipedia.org/wiki/Pest_Control_(Doctor_Who_audio)
- Pest control refers to the regulation or management of a species defined as a pest, usually because it is perceived to be detrimental to a person's health, the ecology or the economy.
  en.wikipedia.org/wiki/Pest_control
- A regular service provided to homes to prevent infestation of pests like cockroaches, ants, etc.
  singapore.angloinfo.com/countries/singapore/propjargon.asp
- insect ...insect had virtually disappeared. The success was repeated in every country where the scale insect had become established without its ...
  www.britannica.com/EBchecked/topic/561465/spruce-sawfly

Figure 18. Google Definitions of the Ontology Term “pest control”

Additionally, we check the use of the term “pest control”, and possibly translated documents that make use of its translations in the desired languages using the AGRIS/CARIS resource[25]. Figure 19 shows a screenshot of the document related with the sample term “pest control”.

[25] http://www.fao.org/agris/search/search.do



Your Search For **pest control** found **84082** results . Showing Results **1-10**

Go to Page 1 2 3 4 5 6 7 8 9 10 Next

Report 1977-79 [stored products pests, UK]
Pest Infestation Control Laboratory, Slough (UK) ;
Search Google for the full-text

Louisiana Pest Control Association study guide for pest control technicians
Anon. ; Louisiana Pest Control Association (USA). Educational Resource Committee ;
This study guide for **Pest Control** Technician applicants is a basic compilation of identification techniques and **control** strategies for common structural pests in Louisiana. The principles of fumigation, pesticide labeling, regulations, and the use calibration and maintenance of pesticide equipment are covered. Information on the classification of pesticides by chemical structure, use, mode of action, and regulation is included
Search Google for the full-text

Pest control serviceman's training program
Anon. ; Industrial Teaching Systems Corporation (USA); National Pest Control Association (USA) ;
Five self-study workbooks for **pest control** operators (PCOs) comprise Group 1 series: Cockroaches. Each workbook contains a pretest, a learning program, a post test and an answer sheet. Book 1 aids in identification of four common cockroaches--German, American, Oriental, and brown-banded. Book 2 covers the habits of these common cockroaches. Book 3 describes how to properly conduct thorough inspections and how to classify cockroach hiding places. Book 4 covers insecticides
Search Google for the full-text

Figure 19. Uses and "possibly" translated Documents of the Ontology Term "pest control"

Disambiguate the translation senses

With the information obtained in the previous steps we used a manual disambiguation process to rank the translations of the each ontology term. For example in Table 14 we show the ranked translation obtained for the sample term "pest control".

English	French	Italian
pest control	lutte contre les ravageurs	Nebulizzazione postraccolta
pest control	lutte phytosanitaire	Difesa dei prodotti immagazzinati
control (of a pest)	lutte (contre un organisme nuisible)	Difesa postraccolta
		Trattamento postraccolta
		Controllo dei parassiti (postraccolta)

Table 14. Ranked Translations of the Term "pest control" for French and Italian

Task 4. Evaluate term translation(s).

Based on the NeOn guidelines we would identify the following situation:

Terminology evaluation

NeOn guidelines identify clearly mistakes which may be done and allows to exactly identifying correct localization of terms. Table 15 shows the results of the terminology evaluation of the term "pest control" and their related terms. The columns two and three show the translations in French and Italian and the terminology evaluation comments of each translation are shown in the last column.

English	French	Italian	Terminology evaluation
Postharvest spraying	Pulvérisation après récolte	Nebulizzazione postraccolta	Plural test: ok Unchanged test: ok
Product protection (stored)	Protection des denrées (stockées)	Difesa dei prodotti immagazzinati	Plural test: ok Unchanged test: to verify
Postharvest control	Lutte après récolte	Difesa postraccolta	Plural test: ok Unchanged test: to verify
Postharvest treatment	Traitement postrécolte	Trattamento postraccolta	Exact match for all Languages Plural test: ok Unchanged test: ok
Pest control (postharvest)	Lutte antiparasite (après récolte)	Controllo dei parassiti (postraccolta)	The Italian term “Lotta antiparassitaria (postraccolta)” is not used as exact translation of the French term Plural test: ok Unchanged test: failed

Table 15. Terminology Evaluation Results

Semantic fidelity evaluation

In order to evaluate the semantic fidelity of the translation we would implement the “Backward Translation” criteria. Table 16 shows the semantic fidelity evaluation results (only few cases have been analyzed) for some terms translated to French and Italian. As we can see, in many cases the translation do not match exactly the original meaning, but in a deeper analysis, taking in consideration the context and the topics (agriculture) we identified that the semantic fidelity is covered 100% while the syntactic fidelity is not ensured.

Original Term (EN)	Translation	Backward Translation (EN)
Postharvest spraying	Pulvérisation après récolte (FR)	Post-harvest spray Postharvest spraying
Postharvest control	Lutte après récolte (FR)	Postharvest fight Postharvest control
Postharvest spraying	Nebulizzazione postraccolta (IT)	Postharvest haze Postharvest spraying
Postharvest treatment	Trattamento postraccolta (IT)	Postharvest treatment
Postharvest control	Difesa postraccolta (IT)	Postharvest defence Postharvest protection Postharvest control

Table 16. Semantic Fidelity Evaluation Results

Stylistic evaluation

In this case we would check elements such as acronyms, the use of multiple words, capitalizations, etc.

For the mentioned use case no particular problems arises but the use of the parenthesis: for example, the English term “Product protection (stored)” appear to be translated in Italian as “Difesa dei prodotti immagazzinati”. Based on the tested guidelines for localization, in this case we would identify that a more correct translation would appear to be “Difesa dei prodotti (immagazzinati)”.

In other cases instead the proposed translations are consistent as shown in Table 17.

English	French	Italian
Pest control (postharvest)	Lutte antiparasite (après rècolte)	Controllo dei parassiti (postraccolta)

Table 17. Stylistic Evaluation Results

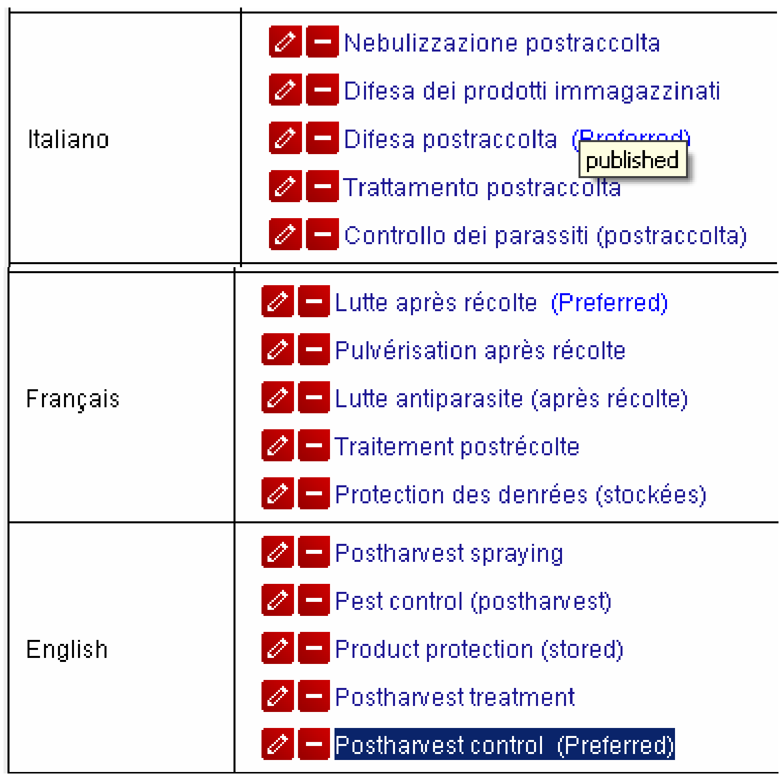
Task 5. Ontology update.

In this task for the mentioned use case we would chose the “Modular strategy” proposed by the guidelines. In this case all lexicalizations belonging to the same concepts are managed with a specific ontological model in which:

- all lexicalizations are represented as instances of the class “c_noun”;
- the concept corresponding to those lexicalization is represented as a class under a top level concept called “c_domain_concept”;
- lexicalizations are attached to the corresponding concept through an objectProperty relationships called “hasLexicalization”.

This model (known as the AGROVOC Concept Server model) has been implemented through the AGROVOC Concept Server Workbench tool, which allows users to easily update the ontology.

The final ontology will contain at least the terminology shown in Figure 9.



Italiano	Nebulizzazione postraccolta Difesa dei prodotti immagazzinati Difesa postraccolta (Preferred) Trattamento postraccolta Controllo dei parassiti (postraccolta)
Français	Lutte après récolte (Preferred) Pulvérisation après récolte Lutte antiparasite (après récolte) Traitement postrécolte Protection des denrées (stockées)
English	Postharvest spraying Pest control (postharvest) Product protection (stored) Postharvest treatment Postharvest control (Preferred)

Figure 20. Final Ontology using Modular Strategy

8.3.2 Pest control ontology localization with LabelTranslator

The objective of this example is to localize some terms of the Pest control ontology from English to Spanish using the LabelTranslator system [32].

LabelTranslator has been designed with the aim of automating ontology localization, and has been implemented in the ontology editor NeOn Toolkit as a plug-in. In its current version, it can localize ontologies in English, German and Spanish. In its design, the methodological guidelines proposed in Section 8.2 have been followed, and some of the techniques and tools described in Section 8.1 have been used.

In the following, we briefly describe how the tasks are performed by LabelTranslator system, and which are the techniques and tools used for each end.

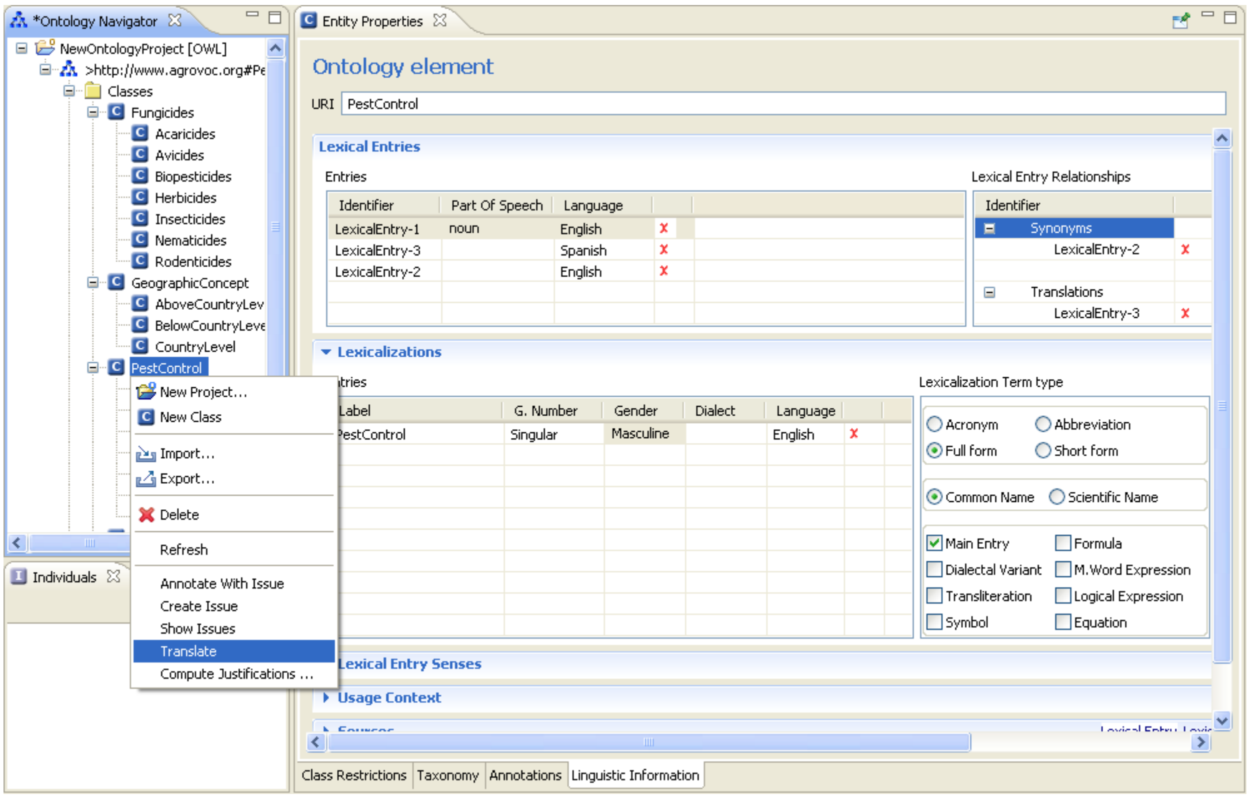
Task 1. Select the most appropriate linguistic assets.

The linguistic assets accessed by the current version of the LabelTranslator NeOn plug-in to perform the ontology localization are multilingual linguistic resources (EuroWordNet, Wiktionary, or IATE) and translation web services (GoogleTranslate, BabelFish, etc.). The addition of further domain specific resources is foreseen for domain ontologies.

Task 2. Select ontology term(s) to be localized.

Once an ontology has been created or imported in NeOn, LabelTranslator allows users and domain experts to manually/automatically sort out the ontology elements that should undergo localization. For each ontology element, LabelTranslator retrieves its local context, which is interpreted by the system using a structure-level approach.

For this sample we manually extracted some concept terms that will be localized into Spanish. Figure 21 shows a screenshot of both Ontology Navigator and the Entity Properties View with information of the sample term “PestControl”.



- NewOntologyProject [OWL]
  - >http://www.agrovoc.org#Pe
    - Classes
      - Fungicides
        - Acaricides
        - Avicides
        - Biopesticides
        - Herbicides
        - Insecticides
        - Nematicides
        - Rodenticides
      - GeographicConcept
        - AboveCountryLev
        - BelowCountryLeve
        - CountryLevel
      - PestControl

Ontology element

URI PestControl

Lexical Entries

Entries

Identifier	Part Of Speech	Language
LexicalEntry-1	noun	English
LexicalEntry-3		Spanish
LexicalEntry-2		English

Lexical Entry Relationships

Identifier
Synonyms
LexicalEntry-2
Translations
LexicalEntry-3

Lexicalizations

Entries

Label	G. Number	Gender	Dialect	Language
PestControl	Singular	Masculine		English

Lexicalization Term type

Acronym Abbreviation [✓ Full form] Short form

[✓ Common Name] Scientific Name

[✓ Main Entry] Formula Dialectal Variant M.Word Expression Transliteration Logical Expression Symbol Equation

Figure 21. Screenshot of the NeOn Toolkit Views used by the LabelTranslator Plug-in

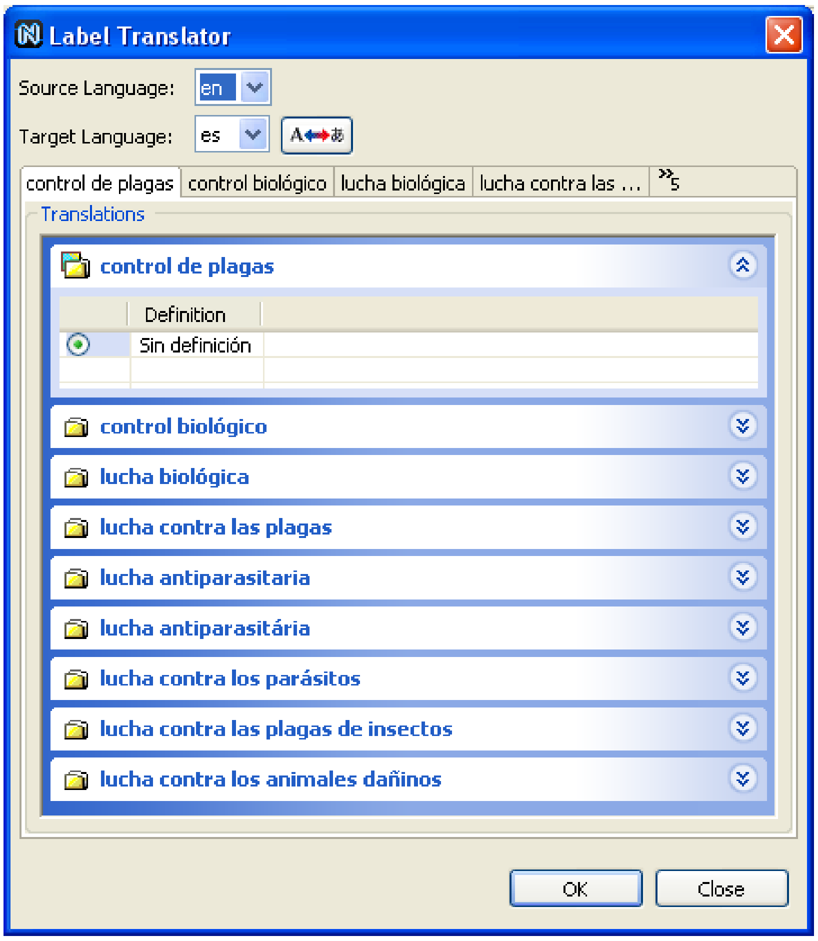
Task 3. Obtain ontology term translation(s).

In order to obtain the most appropriate translation for each ontology element in the target language LabelTranslator uses the following techniques in the indicated order:

- In step 1 the system obtains equivalent translations for all selected labels by accessing the linguistic assets listed in task 1.
- In step 2 the system retrieves a list of semantic senses for each translated label, querying different third-party knowledge pools: Watson[26], which indexes many ontologies available on the Web, and remote lexical resources as EuroWordnet.
- In step 3 the senses of each context label are as well discovered as in step 2.
- In step 4 the system uses a disambiguation method to sort the translations according to their context. LabelTranslator carry out this task in relation to the senses of each translated label and the senses of the context labels. At this stage, domain and linguist experts may decide to choose the most appropriate translation of the ones in the ranking. In default of this, the system will consider the one in the highest position.

In Figure 22 we show a sample of the equivalent translations obtained for the term “PestControl”. Notice that the obtained translations are ranked according to ontology context.

[26] http://watson.kmi.open.ac.uk/WatsonWUI/


Label Translator

Source Language: en

Target Language: es

control de plagas | control biológico | lucha biológica | lucha contra las ... | »5

Translations

control de plagas

	Definition
●	Sin definición

control biológico

lucha biológica

lucha contra las plagas

lucha antiparasitaria

lucha antiparasitária

lucha contra los parásitos

lucha contra las plagas de insectos

lucha contra los animales dañinos

Figure 22. Equivalent Translations for the Term "PestControl"

Task 4. Evaluate term translation(s).

The current version of LabelTranslator does not provide a method for semi-automatically evaluate the translations obtained in the previous step. Therefore, we used a manual evaluation to perform this task.

Based on the NeOn guidelines we would identify the following situation:

Terminology evaluation

Table 18 shows the results of the terminology evaluation of some ontology terms. The middle column shows the translations obtained by LabelTranslator in Spanish. The terminology evaluation comments of each translation are shown in the last column.

English	Spanish	Terminology evaluation
Pest control	Control de Plagas	Plural test: the correct plural form has been used Unchanged test: ok
Product protection (stored)	Protection des denrées (stockées)	Plural test: ok Unchanged test: to verify
Biopesticides	Bioplagicidas	Plural test: ok Unchanged test: to verify
Postharvest treatment	Traitement postrécolte	Exact match for all Languages Plural test: ok Unchanged test: ok

Table 18. Terminology Evaluation Results

Semantic fidelity evaluation

In order to evaluate the semantic fidelity of the translation we would implement the “Backward Translation” criteria. Table 19 shows the semantic fidelity evaluation results (only few cases have been analyzed) for some terms translated to Spanish.

Original Term (EN)	Translation (ES)	Backward Translation (EN)
Pest control	Control de plagas	Pest control Pest management
Brush control	Cepillo de Control	Brush of control
Insect control	Desinsectación	Disinsecting

Table 19. Semantic Fidelity Evaluation Results

As we could see, in many cases the translation does not match exactly the original meaning. In a deeper analysis, taking in consideration the context and the topics (agriculture) we identified that the translations “cepillo de control” and “desinsectación” do not match the original meaning. The remainder translations are covered 100%.

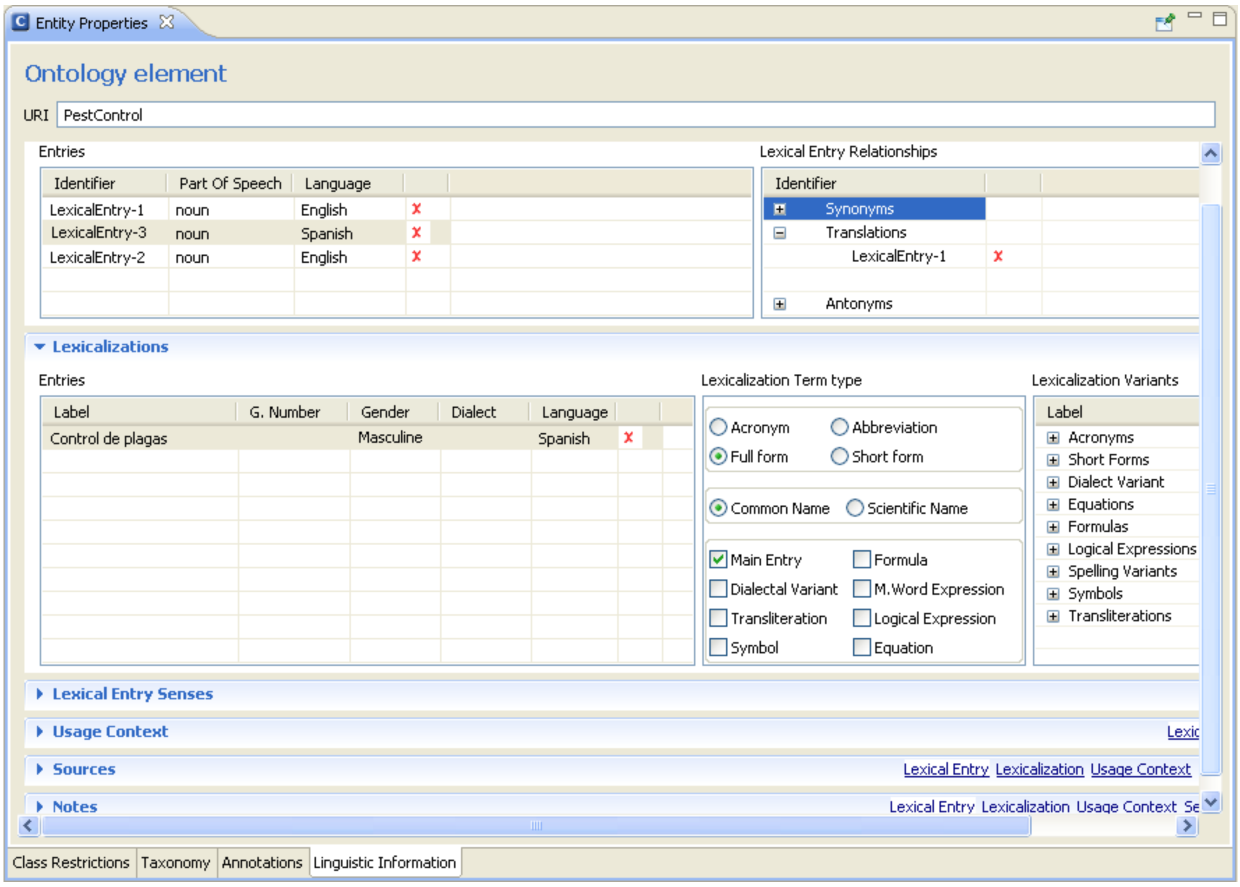
Stylistic evaluation

Within this sample we did not carry out this step.

Task 5. Ontology update.

The ontology is updated with the resulting linguistic data, which is stored in the LIR model, a separate module adopted by the LabelTranslator NeOn plug-in for organizing and relating linguistic information within the same language and across languages to domain ontologies.

Figure 23 shows the Linguistic Information page of the sample term “PestControl”. The linguistic page uses a model based on modular approach to store the linguistic information associated to each ontology term.



Entity Properties

Ontology element

URI PestControl

Entries

Identifier	Part Of Speech	Language	
LexicalEntry-1	noun	English	✗
LexicalEntry-3	noun	Spanish	✗
LexicalEntry-2	noun	English	✗

Lexical Entry Relationships

Identifier	
⊞ Synonyms	
⊟ Translations	
LexicalEntry-1	✗
⊞ Antonyms	

Lexicalizations

Entries

Label	G. Number	Gender	Dialect	Language	
Control de plagas		Masculine		Spanish	✗

Lexicalization Term type

Acronym [✓ Full form] Abbreviation Short form

[✓ Common Name] Scientific Name

[✓ Main Entry] Formula Dialectal Variant M.Word Expression Transliteration Logical Expression Symbol Equation

Lexicalization Variants

Label
⊞ Acronyms
⊞ Short Forms
⊞ Dialect Variant
⊞ Equations
⊞ Formulas
⊞ Logical Expressions
⊞ Spelling Variants
⊞ Symbols
⊞ Transliterations

Lexical Entry Senses

Usage Context — Lexic

Sources — Lexical Entry Lexicalization Usage Context

Notes — Lexical Entry Lexicalization Usage Context Se

Class Restrictions | Taxonomy | Annotations | Linguistic Information

Figure 23. Linguistic Information associated to Ontology Term "PestControl"

8.4. Future Work

In this chapter we have presented preliminary methodological guidelines for carrying out the ontology localization activity. These guidelines have been thought for a scenario which starts from an ontology already conceptualized without taking into account the multilingual and localization aspects.

Further work related with this activity includes: the design of guidelines to support the scenario when the ontology is built using a process that allow handle multiple languages and cultural conventions without the need of re-design it.

9. Conclusions and Future Work

As already mentioned in D5.4.1 [80], our aim within the NeOn project is to create the *NeOn methodology for building ontology networks* covering the drawbacks presented in the three analyzed methodologies (METHONTOLOGY, On-To-Knowledge, and DILIGENT), and benefiting from the advantages included in such methodologies, with respect to the aforementioned characteristics.

Therefore, deliverable D5.4.1 [80] included the first version of the NeOn methodology for building ontology networks presenting the following contributions:

- Analysis of how argumentation and collaboration dimensions are related to the different nine identified scenarios for collaboratively building network of ontologies.
- Prescriptive methodological guidelines for carrying out the ontology specification activity, including three examples on how to apply the proposed methodological guidelines.
- Methodological guidelines for reusing and re-engineering non-ontological resources.
- Prescriptive methodological guidelines for reusing ontological resources, focused on general or common ontologies, domain ontologies as a whole, and ontology statements.
- Methodological guidelines for reusing ontology design patterns by naive users.

The second version of the NeOn methodology is included in this deliverable (D5.4.2) and is focused on:

- improving and extending the methodological guidelines proposed in D5.4.1 [80];
- reusing ontology design patterns;
- modularizing ontology networks;
- evaluating ontology networks;
- evolving ontology networks; and
- localizing ontology networks.

Furthermore, future methodological work (methods, techniques and tools) for continuing the presented step forward will be included in deliverable D5.4.3 focusing on:

- selecting, comparing and combining non-ontological resources, ontological resources, and ontology design patterns for building ontology networks; and
- mappings between ontological resources.

We are also analyzing the possibility of providing in the next version of the NeOn methodology (that is, deliverable D5.4.3) methodological guidelines for:

- deciding which implementation language is better for each type of ontologies;
- deciding which knowledge should be represented as concepts, as relationships, etc.;
- using naming conventions in the ontology (network) development.

References

1. H. Alani, S. Harris, and B. O'Neil. Winnowing Ontologies based on Application Use. Proceedings of 3rd European Semantic Web Conference (ESWC'06), Montenegro (2006).

2. S. Angeletou, A. García-Silva, A. Gómez-Pérez, D. Maynard, M.C. Suárez-Figueroa, W. Peters, B. Villazón-Terrazas. NeOn Deliverable D2.2.2 Methods and tools supporting re-engineering. NeOn Project. http://www.neon-project.org. December, 2007.

3. C. F. Baker, C. J. Fillmore, and J. B. Lowe, “The Berkeley FrameNet project,” Proceedings of the COLING-ACL (1998).

4. J. Banerjee, W. Kim, H. Kim, H. Korth. Semantics and implementation of schema evolution in object-oriented databases. Proc. ACM SIGMOD Conf. Management of Data, 16(3), pp. 311-322 (1987).

5. K. Beck. Embracing Change with Extreme Programming. In: IEEE Computer, (Cover Feature), October 1999.

6. K. H. Bennett, V. Rajlich. Software maintenance and evolution: a roadmap. In ICSE - Future of SE Track, pages 73–87, 2000.

7. S. Bloehdorn. Ontology Evolution. Semantic Web Technologies - Trends and Research in Ontology-based Systems (John Wiley & Sons, 2006), 51-70.

8. E. Blomqvist. Pattern Ranking for Semi-automatic Ontology Construction. In: Proccedings of SAC'08: Track on Semantic Web and Applications (SWA), Fortaleza, Brazil, March 16-20, 2008.

9. D. Bonino, F. Corno, L. Farinetti, A. Ferrato. Multilingual Semantic Elaboration in the DOSE platform. In SAC 2004, ACM Symposium on Applied Computing, 2004.

10. J. Brank, M. Grobelnik, D. Mladenic. A survey of ontology evaluation techniques. In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005), Ljubljana, Slovenia, 2005.

11. P. Buitelaar, M. Sintek, M. Kiesel. A multi-lingual/multimedia lexicon model for ontologies. In Proc. ESWC'06, Budva, Montenegro., 2007.

12. C. Brewster, H. Alani, S. Dasmahapatra, Y. Wilks. Data driven ontology evaluation. Proceedings of Int. Conf. on Language Resources and Evaluation, Lisbon, Portugal, 26–28 May 2004.

13. C. Caracciolo, J. Euzenat, L. Hollink, R. Ichise, A. Isaac, V. Malaise, C. Meilicke, J. Pane, P. Shvaiko, H. Stuckenschmidt, O. Svab-Zamazal, V. Svatek. Results of the Ontology Alignment Evaluation Initiative 2008. OAEI-2008 at the ISWC ontology matching workshop, Karlsruhe, 2008.

14. C. Catenacci, A. Gangemi, J. Lehmann, M. Nissim, V. Presutti, G. Steve. D2.1.1 Design rationales for collaborative development of networked ontologies – State of the art and the Collaborative Ontology Design Ontology. NeOn Project. Available at: http://www.neon-project.org. February 2007.

15. P. Cimiano, P. Haase, M. Herold, M. Mantel, P. Buitelar. Lexonto: A model for ontology lexicons for ontology-based NLP. In OntoLex'07, Busan, South Corea., 2007.

16. B. Cuenca Grau, B. Parsia, E. Sirin, A. Kalyanpur. Automatic par titioning of owl ontologies using -connections. In Description Logics, 2005.

17. L. Chengye, X. Yue, G. Shlomo. Translation disambiguation in web-based translation extraction for english-chinese CLIR. In Workshop of SIGIR 2006, 2006.

18. P. J. Cheng. Translating unknown queries with web corpora for cross-language information retrieval. In Proc. of the 27th annual international conference on Research and development in information retrieval, 2004.

19. M. d'Aquin, A. Schlicht, H. Stuckenschmidt, M. Sabou. Criteria and Evaluation for Ontology Modularization Techniques. To appear in "Ontology Modularization", Christine Parent, Stefano Spaccapietra, Heiner Stuckenschmidt (editors.) Springer, (In press) 2009.

20. M. d’Aquin, P. Haase, C. Le Duc, A. Zimmermann. D1.1.4 NeOn Formalism for Modularization: Implementation and Evaluation. NeOn Deliverable 2008.

21. M. d’Aquin, P. Haase, S. Rudolph, J. Euzenat, A. Zimmermann, M. Dzbor, M. Iglesias, Y. Jacques, C. Caracciolo, C. Buil Aranda, J. M. Gomez. D1.1.3 NeOn formalisms for modularization: Syntax, semantics, algebra. NeOn Deliverable 1.1.3, 2008.

22. M. d'Aquin, P. Doran, E. Motta, V. Tamma. Towards a Parametric Ontology Modularization Framework Based on Graph Transformation. International Workshop on Modular Ontologies, K-CAP 2007.

23. M. d’Aquin, A. Schlicht, H. Stuckenschmidt, M. Sabou. Ontology modularization for knowledge selection: Experiments and evaluations. In Roland Wagner, Norman Revell, and Gu□nther Pernul, editors, Database and Exper t Systems Applications, 18th International Conference, DEXA 2007, Regensburg, Germany, September 3-7, 2007, Proceedings, volume 4653 of Lecture Notes in Computer Science, pages 874–883. Springer, 2007.

24. M. d’Aquin, M. Sabou, E. Motta. Modularization: a key for the dynamic selection of relevant knowledge components. In Workshop on Modular Ontologies, 2006.

25. P. De Leenheer, T. Mens. Ontology Management. Semantic Web, Semantic Web Services, and Business Applications, Chapter Ontology Evolution. State-of-the-art and Future Directions. Springer, 2007.

26. P. De Leenheer, A. de Moor, R. Meersman. Context Dependency Management in Ontology Engineering: a Formal Approach. Journal on Data Semantics VIII, LNCS 4380, Springer-Verlag, pp. 26-56 2007.

27. K. Dellschaft, H. Engelbrecht, J. MonteBarreto, S. Rutenbeck, S. Staab. (2008). Cicero: Tracking Design Rationale in Collaborative Ontology Engineering. Proceedings of the ESWC 2008 Demo Session.

28. P. Doran, I. Palmisano, V. Tamma. SOMET: Algorithm and Tool for SPARQL Based Ontology Module Extraction. International Workshop on Ontologies: Reasoning and Modularity (WORM-08), ESWC 2008.

29. P. Doran, V. Tamma, L. Iannone. Ontology module extraction for ontology reuse: An ontology engineering perspective. In Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management, 2007.

30. M. Dzbor, M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Blomqvist, H. Lewen, and M. Espinoza. D5.6.2 Experimentation with parts of NeOn methodology. Technical report, NeOn Project, 2009. Available at: http://www.neon-project.org.

31. T. Declerck, A. Gómez-Pérez, O. Vela, Z. Gantner, D. Manzano-Macho. Multilingual lexical semantic resources for ontology translation. In Proceedings of LREC 2006, 2006.

32. M. Espinoza, A. Gómez-Pérez, E. Mena. Enriching an ontology with multilingual information. In Proc. of 5th European Semantic Web Conference (ESWC'08), Tenerife, (Spain), June 2008.

33. M. Espinoza, A. Gómez-Pérez, E. Mena. Labeltranslator - automatically localizing an ontology. In Proc. of 5th European Semantic Web Conference (ESWC'08), Tenerife, (Spain), June 2008.

34. C. Fellbaum, Wordnet: An Electronic Lexical Database (MIT Press, 1998).

35. M. S. Fox, M. Barbuceanu, M. Gruninger, J. Lin. An organization ontology for enterprise modelling. In: M. Prietula et al. (eds.), Simulating organizations: Computational models of institutions and groups, AAAI/MIT Press, 1998, pp. 131–152.

36. A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann. Modelling Ontology Evaluation and Validation. In Proceedings of the Third European Semantic Web Conference, volume 4011 of LNCS. Springer, 2006.

37. A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann. (2005). Ontology evaluation and validation - An integrated formal model for the quality diagnostic task. *Technical Report*, available at http://www.loa-cnr.it/Publications.html.

38. J. Hartmann, Y. Sure, A. Giboin, D. Maynard, M. C. Suárez-Figueroa, and R. Cuel. Methods for ontology evaluation. Knowledge Web Deliverable D1.2.3. 2005.

39. P. Haase, S. Rudolph, Y. Wang, S. Brockmans, R. Palma, J. Euzenat, M. d'Aquin. NeOn Deliverable D1.1.1. Networked Ontology Model. November 2006. Available at: http://www.neon-project.org/.

40. P. Haase, L. Stojanovic. Consistent Evolution of OWL Ontologies. Proceedings of the Second European Semantic Web Conference, Heraklion, Greece, 2005 (2005): 182-197. LNCS 3532, Springer.

41. N. Ide, J. Veronis. Introduction to the special issue on word sense disambiguation: The state of the art. Computational Linguistics. Special Issue on Word Sense Disambiguation, 1998.

42. M. Klein. Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit, Amsterdam), 2004.

43. M. Klein, N. Noy. A Component-based Framework for Ontology Evolution. In Proceedings Workshop on Ontologies and Distributed Systems, IJCAI 2003 (Acapulco, Mexico).

44. M. Klein, D. Fensel. Ontology versioning for the Semantic Web. 2001.

45. S. Kohler, J. Euzenat, G. Herrero, C. Caracciolo. Evaluation of Mapping. NeOn deliverable D3.4.1, 2009.

46. A. Kubias, S. Schenk, S. Staab. SAIQL Query Engine - Querying OWL Theories for Ontology Extraction. In Poster Session of ESWC 2007.

47. K. Linden. Word sense discovery and disambiguation. Academic Dissertation, University of Helsinki, Faculty of Arts, Department of General Linguistics, 2005.

48. V. López, E. Motta, M. Dzbor, M. d'Aquin, S. Peroni, D. Guidi. D8.6. Final Version of the Question Answering System. OpenKnowledge Deliverable, 2009.

49. A. Lozano-Tello, A. Gómez-Pérez. Ontometric: A method to choose the appropriate ontology. Journal of Database Management, 15(2):1–18 (2004).

50. B. MacCartney, S. McIlraith, E. Amir, T.E. Uribe. Practical Par tition-Based Theorem Proving for Large Knowledge Bases. In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI), 2003.

51. A. Maedche, S. Staab. Measuring similarity between ontologies. Proceedings of the 13th Conference on Information and Knowledge Management (2002). LNAI Vol. 2473.

52. D. Maynard, N. Aswani, W. Peters, F. Zablith, M. d'Aquin, NeOn Deliverable D1.5.3. Advanced Methods for Change Propagation between Networked Ontologies and Metadata (May 2009).

53. T. Menzies. Object-oriented patterns: Lessons from expert systems. Software - Practice and Experience, 1(1), December 1997.

54. E. Montiel-Ponsoda, G. Aguado, A. Gómez-Pérez, W. Peters. Modelling multilinguality in ontologies. In Coling 2008: Companion volume - Posters and Demonstrations, Manchester, UK, 2008.

55. V. Novacek, L. Laera, S. Handschuh. Semi-automatic Integration of Learned Ontologies into a Collaborative Framework. International Workshop on Ontology Dynamics (IWOD-07) (2007).

56. N. Noy, A. Chugh, W. Liu, M. Musen. A framework for ontology evolution in collaborative environments. In International Semantic Web Conference, pages 544–558, 2006. Athens, Georgia, USA.

57. N.F. Noy, M.A. Musen. Specifying Ontology Views by Traversal. In Proc. of the International Semantic Web Conference (ISWC), 2004.

58. N. Noy, S. Kunnatur, M. Klein, M. Musen. Tracking changes during ontology evolution. In International Semantic Web Conference, 2004.

59. D. Oliver. Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary. 2000. Stanford University.

60. K. Ottens, M. P. Gleizes, P. Glize. A multi-agent system for building dynamic ontologies. Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (Honolulu, Hawaii: ACM, 2007), 1-7.

61. M. T. Pazienza, A. Stellato. Exploiting linguistic resources for building linguistically motivated ontologies in the semantic web. In Second Workshop on Interfacing Ontologies and Lexical Resources for Semantic Web Technologies (OntoLex2006), held jointly with LREC2006, May 24-26, 2006, Genoa, (Italy), 2006.

62. R. Palma, P. Haase, Ó. Corcho, A. Gómez-Pérez and Q. Ji. An Editorial Workflow Approach for Collaborative Ontology Development. Springer. ASWC. 2008.

63. R. Palma, J. Hartmann, P. Haase. OMV - Ontology Metadata Vocabulary for the Semantic Web, 2008. v. 2.4, available at http://omv.ontoware.org/.

64. R. Palma, P. Haase, Q. Jiu. D1.3.2 Evaluation of propagation models and strategies. Technical Report D1.3.2; NeOn Deliverable, December 2008.

65. R. Palma, P. Haase, Y. Wang, M. d'Aquin. D1.3.1 propagation models and strategies. Technical Report D1.3.1; NeOn Deliverable, November 2007.

66. R. Porzel, R. Malaka. A task-based approach for ontology evaluation. Proceedings of the ECAI 2004 Workshop on Ontology Learning and Population. PP. 9–16, 2004.

67. V. Presutti, A. Gangemi, S. David, G. Aguado de Cea, M. C. Suárez-Figueroa, E. Montiel-Ponsoda, M. Poveda. D2.5.1: A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies. NeOn Project. Available at: http://www.neon-project.org. February 2008.

68. A. L. Rector. Normalisation of ontology implementations: Towards modularity, re-use, and maintainability. Proceedings Workshop on Ontologies for Multiagent Systems (OMAS) in conjunction with European Knowledge Acquisition Workshop, Siguenza, Spain, 2002.

69. M. Sabou, M. d'Aquin, E. Motta. Exploring the Semantic Web as Background Knowledge for Ontology Matching. Journal on Data Semantics, no. XI (2008).

70. M. Sabou, S. Angeletou, M. d'Aquin, J. Barrasa, K. Dellschaft, A. Gangemi, J. Lehmann, H. Lewen, D. Maynard, D. Mladenic, M. Nissim. NeOn D2.2.1. Methods for Selection and Integration of Reusable Components from Formal or Informal User Specifications. NeOn Project. Available at: http://www.neon-project.org. May 2007.

71. F. Scharffe, Y. Ding, D. Fensel. Towards correspondence patterns for ontology mediation. In: Proceedings of The Second International Workshop on Ontology Matching, 2007.

72. J. Seidenberg, A. Rector. Web ontology segmentation: Analysis, classification and use. In Proceedings of the World Wide Web Conference (WWW), Edinburgh, June 2006.

73. P. Shvaiko, J. Euzenat, F. Giunchiglia, H. Stuckenschmidt. Proc. 3rd Intl. Workshop on Ontology Matching, Karlsruhe, 2008.

74. L. Stojanovic. Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe (TH), Germany, August 2004.

75. H. Stuckenschmidt. Toward Multi-Viewpoint Reasoning with OWL Ontologies. In Proc. of the European Semantic Web Conference (ESWC), 2006.

76. H. Stuckenschmidt, M. C. A. Klein. Structure-based par titioning of large concept hierarchies. In International Semantic Web Conference, pages 289–303, 2004.

77. M. C. Suárez-Figueroa, M. Fernández-López, A. Gómez-Pérez, K. Dellschaft, H. Lewen, M. Dzbor. *NeOn D5.3.2. Revision and Extension of the NeOn Development Process and Ontology Life Cycle*. NeOn project. http://www.neon-project.org. November 2008.

78. M. C. Suárez-Figueroa, A. Gómez-Pérez. First attempt towards a standard glossary of ontology engineering terminology. Proceedings of 8th International Conference on Terminology and Knowledge Engineering (TKE'08), 2008.

79. M. C. Suárez-Figueroa, A. Gómez-Pérez. Towards a Glossary of Activities in the Ontology Engineering Field. 6th Language Resources and Evaluation Conference (LREC 2008). Marrakech (Morocco). 2008.

80. M. C. Suárez-Figueroa, G. Aguado de Cea, C. Buil, K. Dellschaft, M. Fernández-López, A. García, A. Gómez-Pérez, G. Herrero, E. Montiel-Ponsoda, M. Sabou, B. Villazon-Terrazas, Z. Yufei. *NeOn D5.4.1. NeOn Methodology for Building Contextualized Ontology Networks*. NeOn project. http://www.neon-project.org. February 2008.

81. M. C. Suárez-Figueroa, G. Aguado de Cea, C. Buil, C. Caracciolo, M. Dzbor, A. Gómez-Pérez, G. Herrero, H. Lewen, E. Montiel-Ponsoda, V. Presutti. *NeOn Deliverable D5.3.1. NeOn Development Process and Ontology Life Cycle*. NeOn Project. http://www.neon-project.org. August 2007.

82. M. C. Suárez-Figueroa, S. Brockmans, A. Gangemi, A. Gómez-Pérez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou. D5.1.1 NeOn Modelling Components. NeOn Project. Available at: http://www.neon-project.org. March 2007.

83. F. M. Suchanek, G. Kasneci, G. Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. Elsevier Journal of Web Semantics, 2008.

84. E. Sunagawa, K. Kozaki, Y. Kitamura, R. Mizoguchi. An environment for distributed ontology development based on dependency management. In International Semantic Web Conference, pages 453–468, 2003.

85. P. Starren, M. Thelen. General dictionaries and students of translation: A report on the use of dictionaries in the translation process. In Proceedings BudaLEX88, 1988.

86. R. Trillo, J. Gracia, M. Espinoza, E. Mena. Discovering the semantics of user keywords. Journal on Universal Computer Science. Special Issue: Ontologies and their Applications, 2007.

87. M. Völkel. D2.3.3.v2 SemVersion: Versioning RDF and Ontologies. Technical report, University of Karlsruhe, January 2006.

88. J. Völker, E. Blomqvist. D3.8.1 Prototype for Learning Networked Ontologies. NeOn Project. Available at: http://www.neon-project.org. February 2008.

89. D. Vrandecic. The DILIGENT knowledge processes. Journal of Knowledge Management 9, no. 5 (2005): 85-96.

90. Y. Zhang and P. Vines. Using the web for automated translation extraction incross-language information retrieval. In Proceedings of the 27th annual international ACM SIGIR conference, 2004.

91. S. Bohner, R. Arnold. Software change impact analysis. IEEE Computer Society Press. 1996.

92. P. Plessers. An Approach to Web-based Ontology Evolution. PhD Thesis, Department of Computer Science, Vrije Universiteit Brussel, Brussel, Belgium. 2006

93. E. J. Chikofsky, J. H. Cross. Reverse engineering and design recovery: A taxonomy. IEEE Software 7(1), pp. 13-17. 1990.

94. H. A. Proper, T. A. Halpin. Conceptual Schema Optimisation: Database Optimisation before sliding down the Waterfall. Technical Report 341, Department of Computer Science, University of Queensland, Australia. 1998.

95. J. Banerjee, W. Kim, H. Kim, H. Korth. Semantics and implementation of schema evolution in object-oriented databases. Proc. ACM SIGMOD Conf. Management of Data, 16(3), pp. 311-322. 1987.

96. B. Parsia, E. Sirin, A. Kalyanpur. Debugging OWL ontologies. Proc. 14th Int'l Conf. World Wide Web. ACM Press, pp. 633-640. 2005.

97. E. Bozsak, Marc Ehrig, Siegfried Handschuh, Andreas Hotho, Alexander Maedche, Boris Motik, Daniel Oberle, Christoph Schmitz, Steffen Staab, Ljiljana Stojanovic, Nenad Stojanovic, Rudi Studer, Gerd Stumme, York Sure, Julien Tane, Raphael Volz, Valentin Zacharias. KAON - Towards a large scale Semantic Web. In Proceedings of E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002. pp 304-313. Springer.

98. C. Caracciolo, J. Heguiabehere. NeOn Deliverable D7.2.3. Initial Network of Fisheries Ontologies. NeOn Project, March 2009.

99. C. Caracciolo, A. Gangemi. NeOn Deliverable D7.2.2. Revised and Enhanced Fisheries Ontologies. NeOn Project, August 2007.